Dipartimento di Informatica e
Scienze dell'Informazione

From irregular meshes to structured models

by

Daniele Panozzo

Theses Series **DISI-TH-2012-03**

DISI, Università di Genova

v. Dodecaneso 35, 16146 Genova, Italy http://www.disi.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica e Scienze dell'Informazione

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

From irregular meshes to structured models

by

Daniele Panozzo

February, 2012

Dottorato di Ricerca in Informatica
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova

DISI, Univ. di Genova
via Dodecaneso 35
I-16146 Genova, Italy
`http://www.disi.unige.it/`

Ph.D. Thesis in Computer Science (S.S.D. INF/01)

Submitted by Daniele Panozzo
DISI, Univ. di Genova
`panozzo@disi.unige.it`

Date of submission: February 2012

Title: From irregular meshes to structured models

Advisor: Enrico Puppo
Univ. of Genova
`puppo@disi.unige.it`

Ext. Reviewers:
Tamy Boubekeur
Telecom Paristech
`tamy.boubekeur@telecom-paristech.fr`

Olga Sorkine
ETH Zurich
`sorkine@inf.ethz.ch`

Abstract

Surface manipulation and representation is becoming increasingly important, with applications ranging from special effects for films and video-games to physical simulation on the hulls of airplanes. Much research has been done to understand surfaces and to provide practical and theoretical tools suitable for acquiring, designing, modeling and rendering them.

This thesis contributes to fill the gap that exists between acquisition of surfaces from 3D scanners and their use in modeling. The problem has been studied from different perspectives, and our contributions span the entire modeling pipeline, from decimation and parametrization to interactive modeling. First and foremost, we propose an automatic approach that converts a surface represented as a triangle mesh to a base domain for the definition of a higher order surface. This allows us to have the advantages of a structured base domain, without the need of defining it by hand. The algorithm performs a series of local operations on the provided triangulation to transform it into a coarse quad mesh, minimizing in a greedy way a functional that keeps the newly computed smooth surface as close as possible to the original triangle mesh.

The same problem is also approached from a different angle, by proposing an algorithm that computes a global parametrization of the surface, using an automatically costructed abstract mesh as domain. The problems are related because whenever a global parametrization of a surface is known, it is possible to produce a quad mesh by imposing a regular grid over the parametrization domain, which is usually a plane or a collection of planes, and mapping it to the surface using the parametrization itself. It is then possible to use surface fitting methods to convert the quad mesh to a base domain for a high-order surface. Our contribution is an algorithm that is able to start from a cross-field defined on a surface, simplify its topology and then use it to compute a global parametrization that is especially suitable for re-meshing purposes. It is also possible to use it for other usual applications of a parametrization, like texturing or non-photorealistic rendering.

Since most objects in the real-world are symmetric, we studied robust methods to extract the symmetry map from acquired models. For extrinsic symmetries, we propose a simple and fully automatic method based on invariants usually used for image analysis. For intrinsic symmetries, we introduce a novel topological definition of symmetry and a novel algorithm that starting from a few correspondences is able to extract a high-quality symmetry map for the entire shape. The extracted symmetric map is then used to produce symmetric remeshing of existing models, as well as symmetric non-photorealistic rendering and parametrization.

We also introduce an innovative parametrization algorithm for the special case of mapping a rectangular subset of the plane to another subset of different size. This case is of special interest for the task of interactive image retargeting, where the aspect ratio of an image is changed without distorting the content in interesting areas. Our algorithm searches for the parametrization function in the restricted subset of axis-aligned deformations, by minimizing a convex functional. This allows us to achieve robustness and real-time performances even on mobile devices with low processing power. A user-study with 305 participants shows that our method produces high-quality results.

Starting from a structured model, we consider the problem of refining it in an adaptive way. We present a way to encode an arbitrary subdivision hierarchy in an implicit way, requiring an amount of additional space that is negligible with respect to the size of the mesh. The core idea is general, and we present two different instantiations, one for triangle and one for quad meshes. In both cases, we discuss how they can be implemented on top of well-known data structures and we introduce the concept of topological angles, that allows to efficiently navigate in the implicit hierarchy. Our adaptive framework can be used to define adaptive subdivision surfaces and for generating semi-regular remeshing of a given surface.

Finally, we extend common geometric modeling algorithms to prevent intersections. We show that it is possible to extend them to produce interesting deformations, which depend on the modeling algorithm used, to avoid self-intersections during interactive modeling. Self-intersections are a common problem, since they usually represent unrealistic scenarios and if a mesh contains intersections it is hard to run any kind of physical simulation on it. It is thus impossible to realistically model clothes or hair on self-intersecting meshes, and the manual cleaning of these models is time-consuming and error-prone. Our proposal allows us to produce models with the guarantee that self-intersections cannot appear and can be easily integrated into existing modeling software systems.

To my family

Do not worry about your difficulties in Mathematics. I can assure you mine are still greater.

(Albert Einstein)

Acknowledgements

I want to thank my advisor for his guidance and for all the interesting discussions. He always supported me on all my projects and ideas, and taught me how to do research and how to have fun while doing it. Thanks to all the people I met during my visit at the Courant Institute of Mathematical Science in New York, especially Olga Sorkine, Denis Zorin, David Harmon, Alec Jacobson, Ofir Weber, Ashish Myles, Qingnan Zhou and Denis Kovacs. This visit has been a very important step both in my life and in my career and it has been a success thanks to them. I also want to thank my friends and colleagues Paolo Gasti, Danilo Roascio, Maddalena Strumia, Yaron Lipman, Olga Diamanti, Chiara Olivieri, Luigi Rocca, Marco Tarini, Nico Pietroni and Paolo Cignoni. Last but not least, I thank my family for their patience and support.

Table of Contents

Chapter 1

Introduction

Surface manipulation and representation is becoming increasingly important, with applications ranging from special effects for films and video-games to physical simulation on the hulls of air and space-crafts. Much research has been done to understand surfaces and to provide practical and theoretical tools suitable for acquiring, designing, modeling and rendering them. This thesis contributes to fill the gap that exists between acquisition of surfaces from 3d scanners and their use in modeling.

A surface, that for the subsequent discussion has to be considered a two-dimensional manifold embedded in $\mathbb{R}^3$, is usually represented using two different families of representations. It can be represented as a large collection of simple objects, or as a small collection of high-order surface patches. In the former case, the surface is most often represented by a large collection of triangles, every triangle locally approximating the surface with a plane. In the latter, every part of the surface is represented using higher-order approximations; since the high-order patches are more expressive than planes, the same surface can be encoded with the same "approximation quality" with less patches than in the former case.

Triangular meshes are commonly produced by a mesh scanner and they are the standard de facto for interactive rendering. Their advantages are their simplicity and lack of any global structure. Every triangle directly represents a part of the surface approximated by a plane. They are a good linear discretization of continuous surfaces, and the continuous differential operators can be discretized and defined on them [BKP$^+$10]. Unfortunately, they are not always suitable for modeling purposes, since they have a lot of degrees of freedom and modeling a simple deformation like the bending of an arm involves moving hundreds or thousands of vertices at the same time.

Approximating surfaces with high-order patches [Far88], provides advantages in term of quality and in ease and speed of control during modeling. The surface is represented by a coarse control mesh that is used as domain for defining parametric surfaces that form the final shape when stitched together. In this representation, the base domain is usually a mesh with quadrilateral faces, i.e. every face is a quadrilateral defined by the positions of four 3D points, since this domain is better suited for the definition of spline surfaces. The base domains are usually designed from scratch using geometric modeling systems, and their definition usually requires long editing sessions.

In Chapter 2, an automatic approach that converts a surface — represented as a triangle mesh — into a base domain for the definition of a high-order surface representation is presented. This allows to have the advantages of a structured base domain, without the need of defining it by hand. Starting from a mesh of a real 3D object, we compute a coarse base domain whose limit surface approximates the real object arbitrarily well. The algorithm performs a series of local operations on the provided triangulation to transform it into a coarse quad mesh, minimizing in a greedy way a functional that keeps the newly computed smooth surface as close as possible to the original triangle mesh. The local operations are divided in two groups: simplification operations, that reduce the complexity, and optimization operators, that increase the quality of the quad mesh. The application of these two operators is guided by two automatically computed scalar fields called Fitmaps, which guide the simplification and provide a stopping criterion that depends on a single parameter. The first part of the algorithm can also be used to build simple quad meshes, where every face represents a bi-linear patch. We show results computed on scanned and synthetic models.

In Chapter 3, we approach the same problem from a different perspective. We present two novel global parametrization algorithms for surfaces. Global parametrization is clearly related with the problem tackled in Chapter 2: whenever a global parametrization of a surface is known, it is possible to produce a quad mesh by imposing a regular grid over the parametrization domain, which is usually a subset of the plane, and mapping it to the surface using the parametrization itself. It is then possible to use surface fitting methods to convert the quad mesh into a base domain for an high-order surface. Our contribution is an algorithm that is able to start from a cross-field defined on a surface, simplify its topology and then use it to compute a global parametrization that is especially suitable for re-meshing purposes. An interesting property of our parametrization is that instead of continuously mapping a subset of $\mathbb{R}^2$ onto the surface, it uses an abstract cell complex as its domain. Advantages and disadvantages of this approach are thoroughly discussed, and results are shown.

The second parametrization algorithm aims at producing symmetric fields and meshes that adhere with the symmetry already present in shapes. This is a very natural requirement, and in fact most of the models manually designed by artists exhibit clear symmetries that are exploited by them to speedup the modeling and to increase the quality of the results. The detection of symmetries on shapes has been studied extensively in the last years, but it has never been applied to the generation of symmetric fields on surfaces. All existing intrinsic symmetry detection methods rely on the assumption that the symmetry map should be an isometry on the surface. We show that this definition is not suitable for most real shapes, and we propose a new topological definition that is more general. We present an efficient algorithm that is able to extract the intrinsic symmetry map starting from a sparse set of landmarks. For the extrinsic case, we extend the concept of symmetric invariants usually used in image processing to 3D meshes, and we present a trivial algorithm that is able to find bilateral extrinsic symmetries without user input and without parameters. We use the symmetry map to produce symmetric parametrizations, quadrangulations and non-photorealistic renderings on a wide array of models. The method fixes as hard constraints in a global optimization all the parts of the surface that are symmetric with themselves, i.e. the part of the surface that lies on the stationary set of the symmetry map. It then uses the symmetry map to average the directions of an arbitrary cross-field defined on the surface, producing a new cross-field that respects the symmetry and that is then used to compute the global parametrization. The method is general and can be used to symmetrize any scalar, vector, line, cross or tensor field defined on a surface.

We also present an innovative parametrization algorithm for the very special case of parametrization from a rectangular subset of $\mathbb{R}^2$ to another. This case is of special interest for the task of interactive image retargeting, where the aspect ratio of an image is changed without distorting the content in interesting areas. Our algorithm searches for a parametrization function in the restricted subset of axis-aligned deformations, by minimizing a convex functional. Our method is fast and robust and we provide comparisons with 8 methods in the state of the art. A user study with 305 participants shows that our method produces high-quality results. The efficiency of our method allows real-time retargeting of images even on mobile devices, thus opening a lot of possible applications in mobile computing where retargeting is particularly relevant since the devices screens are usually small.

In Chapter 4, we assume to have obtained a coarse base domain (either made of triangles or quads) and we consider the problem of refining it in an adaptive way. We present a way to encode an arbitrary subdivision hierarchy in an implicit way, requiring an amount of additional space that is negligible with respect to the size of the mesh. The core idea is general, and we present two different instantiations, one for triangle and one for quad meshes. In both cases, we discuss how they can be implemented on top of well-known data structures and we introduce topological angles, that allow us to efficiently navigate in the implicit hierarchy. We present benchmarks of the implementation for both of them, showing that the method is efficient and robust. Our adaptive framework can be used for adaptive subdivision surfaces, when proper subdivision rules are used, or for producing an adaptive re-meshing of a surface.

Finally, in Chapter 5, we extend common geometric modeling algorithms to handle intersections. We show that it is possible to extend them to produce interesting deformations, that depend on the modeling algorithm used, to avoid self-intersections during modeling. Self-intersections are a common problem during modeling, since they usually represent unrealistic scenarios and it is usually impossible to run any kind of physical simulation on them. It is thus impossible to realistically model cloth or hairs on meshes that contains intersections, and this usually requires manual cleaning of the overlapping parts. Our proposal allows us to produce models with the guarantee that self-intersections cannot appear and can be easily integrated in existing modeling software system.

1.1 Publications

All the scientific contributions of this thesis have been published on international journals or presented at international conferences.

The quad mesh simplification algorithm presented in Chapter 2 has been presented at EUROGRAPHICS 2010 and published in Computer Graphics Forum [TPC+10]. The second part of Chapter 2, which extend the algorithm to produce Catmull-Clark control grids, has been published in IEEE Transaction on Visualization and Computer Graphics [PPT+11]. Both works are the result of an ongoing collaboration between our group in Genova and Marco Tarini, Nico Pietroni and Paolo Cignoni from the ISTI-CNR of Pisa.

The first section of Chapter 3 has been presented at ACM SIGGRAPH ASIA 2011 and published in ACM Transaction on Graphics [TPP+11]. The core idea was the natural continuation of Chapter 2 and the work is another product of the collaboration with the ISTI-CNR of Pisa. Section 3.3 contains a novel symmetry-aware parametrization algorithm invented during my visit at New York University. I worked with Denis Zorin (New York University), Yaron Lipman (Princeton University) and my advisor on this project. The contribution will be presented at ACM SIGGRAPH 2012 and has been accepted for publication in ACM Transaction on Graphics [PLPZ12]. Section 3.4 has been presented at EUROGRAPHICS 2012 in May 2012 and published in Computer Graphics Forum [PWS12]. The work has been realized in collaboration with Ofir Weber (New York University) and Olga Sorkine (ETH Zurich).

Chapter 4 contains a framework for adaptive subdivision of meshes, which was invented during the first two years of my PhD in collaboration with my advisor. A first version, which is specialized for triangular meshes, has been published in IEEE Transaction on Visualization and Computer Graphics [PP09b]. The concept of topological angles has been presented later in the same year at GRAPP 09 [PP09a], together with the extension of the framework to support the butterfly subdivision scheme. The natural extension of the framework to quadrilateral meshes has been presented at Afrigraph 2010 [PP10] and published in Computer Graphics Forum [PP11].

During my visit at New York University, I collaborated with David Harmon (NYU), Denis Zorin (NYU) and Olga Sorkine(ETH Zurich) on extending common geometric modeling algorithms to handle intersections. The results of our research (Chapter 5) have been presented at SIGGRAPH ASIA 2011 and published in ACM Transactions on Graphics [HPSZ11].

Chapter 2

Reverse Catmull-Clark Subdivision

We present an automatic method to produce a Catmull-Clark subdivision surface that fits a given input mesh. Its control mesh is coarse and adaptive, and it is obtained by simplifying an initial high resolution mesh. Simplification occurs progressively via local operators and addresses both the quality of the surface and its faithfulness to the input shape throughout the whole process. The process is based on a novel set of strictly local operations which preserve quad structure, it is robust and performs well on rather complex shapes. The decimation process is interleaved with smoothing in tangent space. The latter strongly contributes to identify a suitable sequence of local modification operations, leading to a tessellation that adapts well to geometric features. The method is naturally extended to manage preservation of feature lines (e.g. creases) and varying (e.g. adaptive) tessellation densities. Displacement mapping or normal mapping can be applied to approximate the input shape arbitrarily well.

2.1 Reverse Subdivision

Subdivision surfaces have become of central interest during the last few years, because of their high potential in shape design and rendering. A major breakthrough in this context comes from methods for fast evaluation of subdivision surfaces [LS08] that, combined with recent advances in GPUs [Gee08], may support rendering with unprecedented quality and speed [EML09]. Fine detail can be added via displacement mapping [Cas08]. With these methods, high resolution meshes are produced directly in the graphics card, bypassing the limit of throughput from main memory to GPU. Expectations are that subdivision-based modeling will soon replace polygonal modeling even for real-time applications such as videogames.

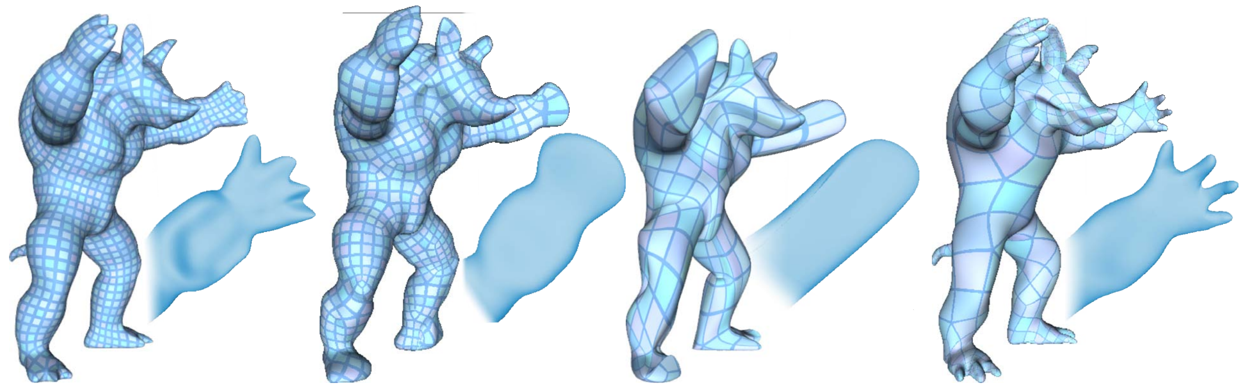


Figure 2.1: *Subdivision surfaces approximating the Armadillo (original mesh 345k triangles) obtained from control meshes produced with different methods. Our approach gives a much better approximation by using fewer patches, thanks to adaptivity.*

This approach is relatively straightforward as long as shapes are created directly as subdivision surfaces [Cas08]. However, many shapes come as polygonal meshes at high resolution, e.g., built by range-scanning real-world objects, or as iso-value surfaces, or by tessellating implicit surfaces, etc. Range scanning is customary in cultural heritage applications; and during production of animation movies and videogames, plaster mock-ups are created by artists prior to being modeled through CAD.

Hence, the problem of converting an input mesh at high resolution into a subdivision surface with a coarse control mesh, possibly enriched with a displacement map or normal map. In this chapter we tackle two subproblems, which have been studied independently in the literature: creating a coarse mesh to approximate a given shape; and fitting a surface to a shape, for a given connectivity of its control mesh. Displacement or normal mapping also require that the output surface can be projected to the input shape. In this work we show that tackling the original problem as a whole leads to better results than combining techniques aimed at resolving each of the sub-problems.

2.1.1 Adaptivity vs regularity

The primary application we address here is fast GPU rendering of displaced subdivision surfaces. To this aim, it is important that the control grid of the subdivision surface is made of as few patches as possible, while preserving the overall shape, and that displacement mapping is used just to add fine detail. For natural objects with details at different scales, such as the Armadillo shown in Figure 2.1, the contrasting objectives of having a good fit and a coarse control mesh can be achieved only if the mesh is adaptive. While techniques for adaptive triangle-based meshing can be considered a consolidated subject, producing an adaptive quad-only mesh for a given shape is still a challenging task: not only the mesh (in fact, its limit surface in our case) should fit the input shape, but also its connectivity, as well as the shape of its patches, should be as regular as possible. Adaptivity and regularity are highly contrasting objectives: transition from coarse to fine patches requires introducing some irregular vertices, or warping the shape of some quads, or both. Several works have been proposed in the literature recently that address the problem of producing a quad mesh that approximates a given input shape well while being made of regular faces, which are possibly aligned either with lineal features, or with a cross field defined on the surface [BZK09, DBG$^+$06, KNP07, RLL$^+$06]. Such methods achieve very good results in terms of regularity of the mesh (see, e.g., the first two examples on the left in Figure 2.1), but they are intrinsically not adaptive, thus requiring many more faces to achieve the same quality of approximation. This is especially true for objects with intricate geometries that are difficult to represent via displacement mapping (see, e.g., hair curls in the David head in Figures 2.18, 2.20): lack of adaptivity leads either to artifacts, or to meshes with many tiny faces even on smooth regular areas. For these reason, in this work we trade some regularity for adaptiveness. Our method combines tools for progressive mesh simplification based on local operations, together with tangential smoothing to keep a regular shape of patches, and subdivision surface fitting to keep the limit surface close to the input shape. Such ingredients are not used independently in separate stages, but they are combined in the context of an integrated framework, in order to take into account all aspects of the problem throughout the whole process.

2.1.2 Objectives

Given a triangular input mesh M, we strive to build a quad-based control mesh K with a limit surface S_K through the Catmull-Clark (CC) subdivision scheme [CC78], such that surface S_K approximates surface M, with the following requirements:

(I) *Conciseness*: the number of faces of K is small;

(II) *Regularity*: most vertices have valence four, and patches have nearly right angles;

(III) *Accuracy*: the difference between S_K and M is small;

(IV) *High projectability*: The great majority of points of M can be reached from S_K by projection along the surface normal.

Conciseness relates to rendering efficiency, ease of editing, minimization of memory consumption, etc. *Regularity* relates to the quality of the meshing that can be obtained with a regular sampling of S_K (e.g., performed at rendering times). *Accuracy* relates to the quality of generated geometrical approximation: it implies not only that positions of vertices of K are carefully chosen, but also that K is tessellated adaptively. Finally *projectability* is very important for several techniques in computer graphics: in practice it means that any attribute of M (e.g., per vertex color or normal) can be stored as texture maps associated to faces of K. Moreover it also means that M can be faithfully reproduced from S_K via displacement mapping in the normal direction, which requires storing just a texture of scalar offsets for each patch. Ideally, one wish to have perfect projectability, i.e., the projection along surface normal defining a bijection between S_K and M. However, since the input can be affected by noise and/or contain detail at arbitrarily high frequency, such a strict requirement may prevent building coarse control meshes. For this reason, we allow for some loss of projectability, which can be controlled by the unique parameter used in our method, as a trade-off for improving adaptivity and coarseness of the output mesh.

2.1.3 Contribution

Our contribution can be summarized as follows:

- We present an integrated approach that incorporates both the automatic construction of a pure-quad, concise and semi-regular control mesh, and the optimization of its related CC subdivision surface, in terms of both projectability and accuracy with respect to the input shape. To the best of our knowledge, this is the first method to produce a pure-quad control mesh while addressing quality of subdivision fit during mesh construction. The only other attempts in this direction have been proposed in [LMH00] for triangle meshes and Loop surfaces and in [MPKZ10] for T-meshes and T-splines.
- Our method is fully automatic and "one-click": it takes in input a geometric mesh and it builds a coarse CC subdivision surface through progressive simplification of its control mesh.
- Mesh simplification is driven from an innovative and effective heuristic, which is based on an static analysis of the input made during preprocessing: a pair of scalar fields, called a *Fitmap*, are computed on the input mesh, which roughly estimate for each vertex of the surface, how well the mesh can be locally modeled with patches, in terms of both geometric error and projectability. This allows us to avoid performing cumbersome geometric tests during the simplification process. This approach is fairly general and probably it can be adapted to any other form of parametric surfaces, as well as to the simpler case of polygonal mesh simplification.
- We experiment our method on several meshes, representing objects with various topologies and details at different scales. We compare our results to CC subdivision surfaces obtained by first building a quad mesh with other state-of-the-art methods, and then using such meshes for subdivision surface fitting and displacement mapping.

Our results clearly outperform those obtained with the other methods.

- Our control meshes contain about two orders of magnitude less faces than the input meshes, thus our method works also as a shape compressor. In fact, displaced subdivision surfaces can be efficiently encoded as a control mesh plus single channel, highly compressible, displacement textures.

This chapter is divided in three sections. We will first review the state of the art in mesh simplification, remeshing and reverse subdivision. In Section 2.3, we will tackle the problem of simplifying a dense quad-mesh, that is a sub-problem of our reverse subdivision algorithm that is interesting by itself. The simplification algorithm will be extended in Section 2.4 to produce control meshes for a catmull-clark surface. Even if the energy minimized is different, both algorithms are based on a common set of local operations and they tackle the two problems in similar ways.

2.2 Related work

Triangle mesh simplification. Simplification of triangle meshes has been studied in depth during the Nineties and can now be considered a mature technology. Good algorithms for simplifying triangle meshes are available in common modeling packages like Maya, Blender or MeshLab [CCR08].

Most triangle mesh simplification algorithms focus on adaptive meshing, with the primary goal of obtaining a good approximation of the original shape with a small number of triangles [CMS97, LRC$^+$02]. Many such algorithms work by means of local modifications, which are iterated until the required LOD is obtained. This approach lends itself naturally to the construction of Continuous LOD (CLOD) models. Local operators can also be useful in a variety of contexts (e.g., mesh editing). Hoppe et al. [HDD$^+$93] introduced the use of additional local operators, such as edge flips, which improve the quality of tessellation rather then reducing mesh complexity. They also introduced the idea to drive the choice of local operations aimed at minimizing of an objective function. We reformulate these ideas for the case of quad meshes.

Quad mesh simplification. Simplification algorithms targeting quad meshes have been developed only recently, and they pose extra difficulties. Collapse of a quad diagonal (a.k.a. quad-close [Kin97], quad-collapse [DSSC08], quad-vertex merge [DSC09a]) is recognized as a valid local operation that preserves quad structure, but a simplification algorithm cannot be based just on it. The standard approach is to use also operations affecting larger areas, so that the quad structure and the overall quality of the mesh are preserved.

Following this direction, the poly-chord collapse has been adopted in [DSSC08], adapting it from the ring collapse introduced in [BBS02]. In a poly-chord collapse, an entire line of side-to-side quads is removed, so that quad regularity is maintained. Poly-chords are alternated to diagonal collapses, striving to maximize the number of regular vertices. Global operations are inherently unpractical: not only they make the change in resolution less continuous, but also their all-or-nothing nature makes them a clumsy tool to maximize any sought objective: an operation with a large footprint (like poly-chord collapse) is likely to have opposite effects on quality in different subparts of the affected area, and still it must be either performed or discarded as a whole. Moreover, the lack of locality makes it difficult, for example, to selectively decimate only an area of the mesh, leaving the rest unaffected, or to be used in a quad-only region of a mixed mesh. Finally, local operations lends themselves better to out-of-core adaptations of the simplification algorithm, being possible to be performed in a small region even if the mesh is constrained elsewhere.

To alleviate the problem linked to global operations, in [SDW+09], rings to be collapsed are "steered" to constrain the affected area inside a user defined subregion. In [DSC09a] poly-chord collapses are split into smaller independent sub-steps, resulting in the first local-only framework for quad meshes. Our scheme is also local-only, but it uses finer grade operations, see Sec. 2.3.2.1. Exclusive use of local operations tends to produce lower quality meshes, though. To improve them, tangent space smoothing is applied to the final result in [DSC09a]. This however has no effect on connectivity. In our proposal, tangent space smoothing is interleaved to local operations at each iteration, and it helps selecting the next operation to be performed.

Local operations have been proposed for improving the quality of 2D quad meshes in [Kin97] and they have been used also to produce quad meshes from 2D triangle meshes in [OSCS99]. However, the problem of optimizing meshes in 3D is quite different from the 2D case.

In the context of an unrelated application, the problem of obtaining a triangle mesh with edges of constant length, similarly to what we propose, was addressed in [IGG01].

Quad-remeshing. A related yet different topic is remeshing. The aim of remeshing is to obtain a completely new mesh, not necessarily at lower complexity, which represents the input shape well and has a superior quality. Again, the focus here is on quality of the mesh, but remeshing is inherently not progressive: the output is built from scratch, using the input mesh just as the reference shape.

A few algorithms for remeshing of quad meshes have been proposed in the literature. Methods proposed in [ACSD+03, LKH08] use alignment to principal curvatures to drive remeshing, while those proposed in [DBG+06, DKG05, HZM+08a, TACSD06] resort to mesh parametrization and uniform sampling in parameter space. Either strategy imposes to solve difficult problems. The methods proposed in [BZK09, RLL+06] belong to both groups, because they use the principal curvatures within a parametrization approach.

Since the objectives of the two tasks (simplification and remeshing) are similar, it is worth to underline when one should be preferred to the other. Most considerations that makes, in some context, local-operation based simplification more attractive than global-operation based one, discussed above, are stronger when applied to remeshing, which is inherently a global operation performed on the entire mesh (for example, it does not lend itself either to the construction of CLOD models, or to local editing). Another issue relates to robustness: in general, remeshing requires solving more complex sub-problems compared to mesh simplification (like parametrization, or identification of principal curvature direction) which are difficult to tackle robustly; remeshing is often less robust to noise, or requires clear, well distanced cone singularities. Also it is hard to produce extremely low resolutions meshes (which can also serve as base domains or control meshes). On the other hand, remeshing can benefit from a global, analyze-then-process, top-down approach, and thus produces output meshes with superior quality. For example, it is possible identify cone singularities [BZK09] and to explicitly enforce a correspondence between vertex valencies of output mesh and gaussian curvature of original mesh [LKH08], which is ideal with man-made objects and CAD models. Notwithstanding that, the proposed local, greedy simplification approach performs almost comparably with [BZK09] even in these cases (Tab. 2.1).

Remeshing algorithms, such as those proposed in [BZK09, RLL$^+$06, DBG$^+$06, KNP07], replace an input mesh with a semi-regular quad (or quad-dominant) mesh, which can represent the same shape with arbitrarily good accuracy. Since all these methods are aimed at achieving face uniformity, they are inherently not adaptive, thus it is hard to apply them with the purpose of drastic simplification. For instance, in the Mixed Integer method [BZK09] grid step is set by the smallest distance between cone singularities of a smooth guidance field computed on the input. Such a distance is likely to be quite small on complex natural shapes. In these cases, drastic simplification can be achieved only if cone singularities are placed manually (see also Section 2.4.2). Myles et al. [MPKZ10] use T-meshes to obtain a small number of adaptive patches while preserving regularity and alignment, but the representation scheme (T-splines) is much more complex.

Algorithms for progressive simplification of quad meshes [DSC09a, DSC09b, TPC$^+$10] are based on sequences of small operations combined with tangent space smoothing. Such algorithms usually aim at obtaining a mesh with good quality, in terms of shape of faces and vertex valencies, while quality of approximation and adaptiveness are usually addressed only indirectly.

We would like to remark that all the above methods are designed to produce general coarse polygonal meshes, rather than control meshes meant to be subdivided. As such, they provide direct control neither on the quality of the subdivision surface, nor on normal projectability to the input data.

Fitting subdivision surfaces: The literature on surface fitting is immense and its review is beyond the scope of this chapter. Dozens of algorithms tackle exclusively the task of geometric fitting a subdivision surface starting with a control mesh with known connectivity. See, e.g., literature reviews in [CWQ$^+$04, LD09, MK04]. We adopt a rather standard approach for solving this problem [HL93], which is orthogonal to our contribution. In this review we rather focus on methods that address the problem of automatically building a suitable control mesh.

Simplification and fitting The problem of finding a coarse subdivision surface that fits an input mesh has been studied in the past for the case of triangle-based subdivision surfaces. Hoppe et al. [HDD$^+$94] first simplify a triangle mesh, then they build a control mesh for Loop subdivision by optimizing vertex positions; further simplification is performed by alternating local operators for mesh simplification and geometric optimization. Later on, Lee et al. [LMH00] combine this approach with displacement mapping. To this aim, they address approximated projectability during simplification.

Similar simplify-then-fit approaches have been proposed in [CWQ$^+$04, MMTP04, MK04]. Beside simplification, some of these methods consider refining the control mesh via local operations, such as edge split and edge swap, to improve portions of surface affected by large error. Conversely, Suzuki et al. [STKK99] adopt a semi-automatic approach based on refinement of a given, manually built, coarse mesh. Kanai [Kan01] modifies the QEM method [GH97] to perform mesh simplification by taking into account the position of points sampled from the limit surface of a Loop subdivision.

It is not clear how such algorithms could be extended to work on quad-based subdivision schemes. However, following the same simplify-then-fit approach, other known techniques could be adopted to produce a coarse control mesh of quads.

Boier-Martin et al. [BMRJ04] and Cohen-Steiner et al. [CSAD04] propose clustering algorithms that generate coarse and adaptive polygonal meshes. Such algorithms take into account projectability to some extent, as clustering is driven from alignment of face normals. Resulting meshes can be quite coarse, but also irregular, containing faces with concave boundaries and many edges. Such faces can be decomposed further to obtain semi-regular quad meshes, but this process usually increases their number for about one order of magnitude (see also Section 2.4.2).

Similarly Marinov and Kobbelt [MK05] use a face-merge method to compute a coarse polygonal mesh, by also taking into account normal projectability to the input. Again, they diagonalize each polygonal face at the end of the process, to obtain a quad-dominant control mesh for Catmull-Clark subdivision, and they compute a normal displacement map from it. Lavoué and Dupont [LD09] use Variational Shape Approximation (VSA) [CSAD04] to build a polygonal control mesh of hybrid tri-quad subdivision surfaces for mechanical objects. Their algorithm is tailored for piecewise-smooth mechanical objects made of a relatively small set of patches joining at sharp edges.

Evaluation of subdivision surfaces: Although subdivision surfaces are defined as the limit of an infinite process of recursive subdivision of a mesh, methods have been developed in the literature that allow one to evaluate them at any point as parametric surfaces. Seminal works by Stam [Sta98a, Sta98b] provide rather complicated methods to evaluate the Catmull Clark and the Loop surfaces at arbitrary points. More recently, Schaefer and Warren [SW07] have proposed a fast and exact method to evaluate subdivision schemes at rational parameters. Later on, Loop and Schaefer [LS08] have proposed an even faster approximated method to evaluate Catmull-Clark surfaces at any parameter value. Their method has been used recently for real-time rendering of subdivision surfaces on the GPU [EML09] and it has also be extended to represent surfaces with sharp creases [KMDZ09]. We make use of this method to evaluate subdivision surfaces during simplification, as well as to compute displacement maps and to use them during rendering.

Mesh analysis: The issue of statically analyzing a mesh to get synthetic information is a vast subject beyond the scope of this chapter. We just discuss some of the topics that are more closely related to our approach. Discrete curvature measures for triangular meshes (see [GG06] for a recent survey) have long been used for feature identification, segmentation into regions, and many other purposes. many purposes in geometry processing. Curvature-based measures have been used to drive simplification of triangle meshes in [KKL02, WB01a]. Curvature analysis is also at the basis of *mesh saliency* [LVJ05], is defined by computing for mesh vertices the difference between mean curvatures filtered with a narrow and a broad Gaussian kernel; they use this scale dependent measure a scale-independent measure that has been used to weight the relevance of geometric error during triangle mesh simplification.

Gal et al. [GSCO07] define the Shape Diameter Function for watertight meshes, which measures how far each point of the mesh is with respect to its antipodal point on the surface. This function is used to define statistical descriptors for shape retrieval. The way it is practically computed (i.e., by shooting rays from each point towards the interior) could be related to our issue of estimating how well a portion of the mesh can be safely projected onto patches.

Principal curvature directions have been used by several authors to produce surface parametrization. In [LRL06], a curvature-based parametrization method is used to produce a control mesh for T-splines, addressing a problem that is closely related to the subject of this chapter.

We make minor use of curvature estimation in the computation of our Fitmaps.

2.3 Simplification of dense quadrilateral meshes

Quad meshes, i.e. meshes composed entirely of quadrilaterals, are important data structures in computer graphics. Several applications in modeling, simulation, rendering, etc. are better suited for quad meshes than for triangle meshes. In spite of this, much literature in geometry processing in the past has addressed more the case of triangle meshes, while similar problems for quad meshes are either relatively unexplored, or they have been addressed only very recently. This is the case of mesh simplification, i.e., the task of producing a low complexity mesh M' out of a high complexity one M.

Compared to the case of triangle meshes, simplification of quad meshes poses extra challenges, because quads are less adaptive and more delicate structures than triangles. The main goal here is to obtain a mesh with good quality, i.e., having almost flat and square faces, and most vertices with regular valence four. Quality of approximation and adaptiveness are usually addressed only indirectly.

In this Section, we present a novel approach to the problem of quad mesh simplification, striving to use practical local operations, while maintaining the same goal to maximize tessellation quality. We aim to progressively generate a mesh made of convex, right-angled, flat, equally sided quads, with a uniform distribution of vertices (or, depending on the application, a controlled/adaptive sample density) having regular valency wherever appropriate. The orthogonal objective of maximizing shape similarity between input and output surfaces can be achieved indirectly by enforcing line features and adaptive sampling.

The presented novel approach to quad mesh simplification is incremental, greedy, and based on local operations only. It includes a novel set of local operators preserving the quad structure, prioritized by a simple yet effective criteria, and interleaved with vertex smoothing in tangent space. We show that this approach is effective to solve the otherwise difficult problem of producing quad meshes with a good quality.

The system lends itself well to efficient implementation, and it is easily extended to reconstruct feature lines, or to progressively produce variable tessellation densities.

As a minor contribution we offer a short analysis of coherence preservation for local operations in quad meshes. We also discuss in which context configurations traditionally regarded as degenerate (doublets) are useful and can be kept.

The approach is completed with an original Triangle-to-Quad conversion algorithm that behaves well in terms of tessellation quality and, given a closed mesh with n triangles, always generates $n/2$ quads.

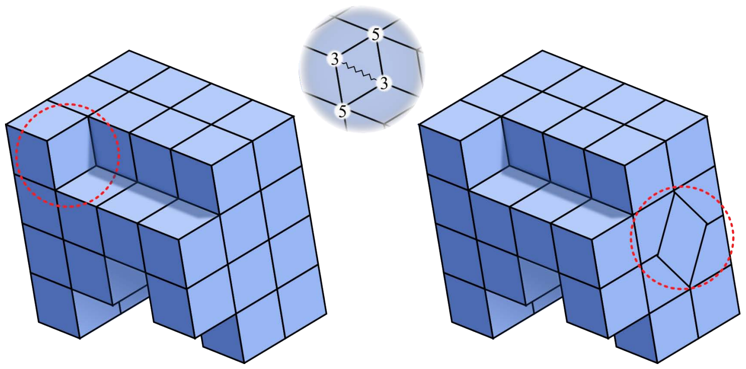


Figure 2.2: *Left: an example of surface allowing for a fully homeometric quad meshing (a polycube surface). The diag-collapse in the inset affects 4 irregular vertices and would make then all regular. However (right) such diag-collapse is beneficial in this case only.*

2.3.1 Overview of the method

Our proposal is to approach the problem of maximizing the quality of a quad mesh during simplification from a new, straightforward perspective. Consider an ideal two-manifold quad mesh composed entirely of flat, equally sided, regular squares. The surface of a regular poly-cube constitutes an example of surface that allows for this ideal tessellation (see Fig. 2.2). Note that this ideal condition can be enforced just by measuring lengths, i.e.: all edges of the mesh have the exact same length l, and all diagonals of faces have exactly length $l\sqrt{2}$. We call this condition on edges and diagonals *homeometry*.

Homeometry indirectly implies that vertex valencies depend on local surface shape: vertices in regions of zero Gaussian curvature have valence 4; vertices in regions of (high) positive Gaussian curvature have valence < 4; and vertices in regions of (high) negative Gaussian curvature (i.e., saddles) have valence > 4. This relation between valence and Gaussian curvature may be brought to an extreme by considering as profitable to have valence 2 in regions of extremely high positive curvature (Sec. 2.3.2.2). Also, a homeometric quad mesh is optimal in the sense that all the angles are right, all the faces flat, and the distribution of vertices is uniform.

Clearly, it is hardly possible that a general surface allows for a fully homeometric quad tessellation. However, homeometry gives us an easier objective to pursue, i.e., one that is only length based, works at all scales, and substitutes well, in practice, more complex criteria like the ones involving Gaussian curvature or vertex valencies.

We measure how far a given mesh M is from being homeometric by means of the variance of lengths of edges and diagonals:

$$\sum_{e \in M^E} (|e| - \mu)^2 + \sum_{d \in M^D} (|d| - \sqrt{2}\mu)^2 \tag{2.1}$$

where e and d span over the sets of edges M^E and diagonals M^D of M, respectively, and μ represents the ideal edge length, computed as the side of an ideal square quad of M:

$$\mu = \sqrt{Area(M)/|M|}, \tag{2.2}$$

where $|M|$ denotes the number of faces of M.

Our simplification method modifies the input mesh to reduce its complexity, while trying to minimize the objective function (2.1). Homeometry-driven simplification blends naturally the need for regular vertices (or, rather, valence matching curvature) with other desiderata, such as uniform vertex spacing. In fact, we believe this approach to be superior than trying to impose regular valence at all vertices, as, e.g., in [DSSC08, DSC09a]. As a clarifying, intuitive example, consider the situation in Fig. 2.2, left. A criterion trying to maximize regular vertices would see as beneficial the collapse of the marked quad diagonal. On the other hand, the initial situation containing irregular vertices is obviously optimal, due to the local discrete curvature, and such a collapse should be considered harmful. Note that this situation can happen at any scale and with unbounded frequency. On the right side of Fig. 2.2, the same connectivity configuration is present on an underlying geometry with zero Gaussian curvature, where valence four is always appropriate: here, the same diagonal collapse would be beneficial. This collapse would be favored in our method because that diagonal is shorter than the prescribed one, against the homeometry criterion, and not by identifying the curvature (which is a relatively complex task involving computation of discrete curvature at varying scale during the simplification process).

2.3.2 Conceptual algorithm

An input mesh M_0 is progressively coarsened by a sequence of either complexity-reducing or local optimizing operations, thus producing a sequence of meshes M_i until a user defined criterion is met (e.g. on the number of quads). One strength of our approach is the use of local operations only, which preserve the quad structure at all steps.

The proposed method is based on three mutually interacting ingredients: a novel set of local operations for quad-only meshes, which constitute the atomic steps of the framework; a heuristic, practical criterion to select the most promising (or less threatening) operation, which tends to maximize homeometry; a tangent smoothing operator, which displaces vertices over the surface of the mesh, without leaving it. The method can be summarized as follows:

0. [Convert input triangle mesh into quad mesh M_0] (Appendix A)
1. Initial global smoothing of mesh M_0 (Sec. 2.3.2.3)
2. Iteratively process mesh M_i to produce mesh M_{i+1} until user-defined criterion is met. In each loop:

   (a) for a fixed number of times:

      i. perform any profitable local optimizing-operation, until none is available (Secs. 2.3.2.1 and 2.3.2.2)

      ii. select and perform a local coarsening-operation (Secs. 2.3.2.1 and 2.3.2.2)

   (b) local smoothing (Sec. 2.3.2.3)

3. Final global smoothing of mesh M_n (Sec. 2.3.2.3)

Step 0 is applied only in case the input comes in the form of a triangle mesh. Only collapse operations applied during Step ii simplify the mesh, while the other operations are aimed on one hand at improving mesh quality in terms of both connectivity (Step i) and sample distribution (Step b), and on the other hand, at driving the selection of best coarsening operations to be performed next. In particular, during the final Step 3 the main purpose of smoothing is to improve the quality of the mesh, while elsewhere (in Step 1 and Step 2) it has a more crucial role: by modifying lengths of linear elements over the mesh, it effectively drives the selection of local operations to be performed at the next cycle.

This schema lends itself well to an efficient implementation. The resulting method is fully automatic and it depends only on a small number of parameters that are used mainly to control the tradeoff between accuracy and speed and do not need to be adjusted depending on specific input.

2.3.2.1 Local Operations

We define three kinds of local operations (see Fig. 2.3): *coarsening operations,* which reduce complexity; *optimizing operations,* which change local connectivity without affecting the number of elements; and *cleaning operations,* which resolve local configurations considered degenerate.

Optimizing operations (or rotations) *Edge rotate:* consider a non-border edge, shared by two quads, and dissolve it, leaving a hexagonal face. There are two other ways to split that face into a pair of quads. We substitute the deleted edge with either one of the two possibilities, calling the two alternatives a clockwise and a counterclockwise edge rotation, respectively. Thus, for each non-border edge in the current mesh, there are two potential edge-rotate operations.

Vertex rotate: consider a non-border vertex v. Each of the k quads sharing v can be split in 2 triangles along the diagonal emanating from v. The $2k$ triangles can be merged next along the former quad edges. Diagonals used to split the quads thus become the new edges. We call this operation a rotation because it can be seen as a rotation of edges around the

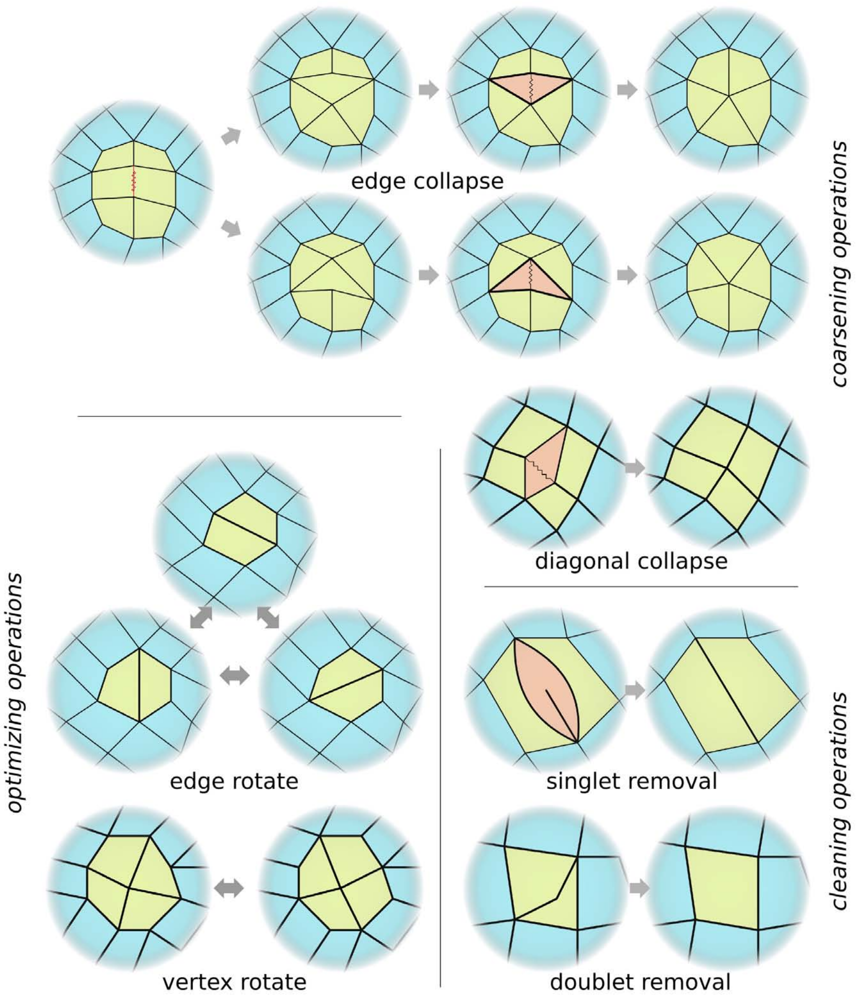


Figure 2.3: *The set of local operations.*

vertex, in either direction (like sails in a windmill). For each non-border vertex of the current mesh, there is one potential vertex-rotate operation.

Coarsening operators (or collapses) *Diagonal collapse:* a quad q can be collapsed on either diagonal, removing q from the mesh, and merging the two vertices at the end of the collapsing diagonal into one new vertex. The structure of the quad mesh is preserved, and its complexity is reduced by one quad, two edges and one vertex. For each quad in the current mesh, there are two potential diagonal collapse operations, one along each diagonal. This is the most widely used operation for quad-meshes. The position of the new vertex resulting from collapse is set so that the objective function (2.1) is minimized in its star.

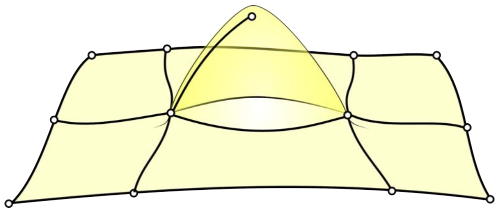


Figure 2.4: *The "singlet" degenerate configuration. Edges are shown as curved lines for illustration purposes.*

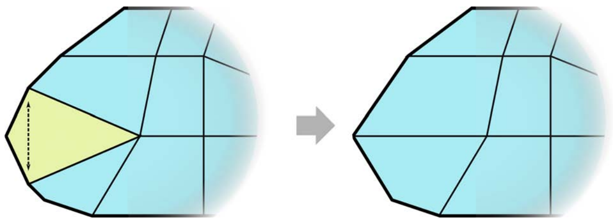


Figure 2.5: *Two border edges are removed by collapsing a quad on the border.*

Edge collapse: given a non-border edge e, we can perform a vertex rotation around either endpoint, turning e into a quad diagonal, and then collapse it. There are two alternatives, corresponding to which endpoint is rotated, producing two distinct potential edge-collapse operations for e (non border edges connecting two border vertices cannot be collapsed).

Cleaning operations (or removals) *Doublet removal:* a doublet is a well-known configuration where two adjacent quads share two consecutive edges. The situation can also be described, and it is best detected, as having a valency 2 non-border vertex. A doublet can be eliminated by dissolving the two shared edges, removing the vertex in the middle, merging the two quads into a single one.

Singlet removal: a singlet is a degenerate configuration where a quad is folded such that two consecutive edges become coincident (see Fig. 2.4). Singlets arise, for example, if two quads initially share three edges and then either one of the two consequential doublets is removed. A singlet is healed by removing the degenerate quad and substituting it with an edge. The valence 1 vertex is also removed, but everything else is kept unmodified.

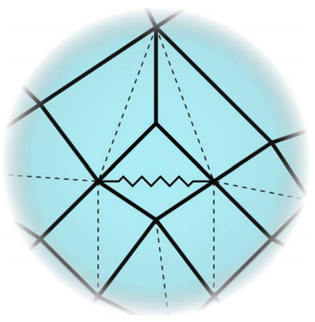
Discussion on the operation set At first, it may seem that border edges cannot be subject to any decimation operation. In fact, a border edge cannot be removed through edge-collapse, otherwise a single triangle would be produced and there would be no local operation to bring a pure quad configuration back. A mesh decimation process that keeps all the original border edges untouched would be clearly unusable. This problem is not a real one, because eventually quads will be generated having two consecutive edges on the border. Collapsing the corresponding diagonal of one such quad removes both border edges and the dangling vertex as a side effect. (see figure Fig. 2.5).

Redundancy: It is easy to check that vertex-rotations and edge-rotations are independent operations, meaning that neither one can be replaced by a sequence of the other. Moreover an edge-collapse could be seen as a combination of a vertex-rotation and a diag-collapse. However, it is convenient to consider it as an atomic operation, because it is often the case that its effect on mesh quality (homeometry) is very different from the effect of the first sub-operation alone.

Note that the edge operation described as a "qeMerge" (quad-edge merge) in [DSC09a] can be considered as the combination of two vertex rotations and four diagonal collapses. Using the latter represents a finer granularity (in this case, we can think of no apparent advantage in considering the sequence of all six operations to be atomic).

Doublet-removal can be seen as a special case of diagonal collapse, the only difference being how they affect geometry: in doublet removal the position of the new vertex is set as one of the two extremes of the collapsing diagonal.

Consistency: All the above operations preserve topology. There are only two potential inconsistencies arising from their application. Any operation creating a quad edge connecting a vertex to itself must be prevented. The only other problem is that of singlets (and possibly doublets, if they are to be considered degenerate: see discussion later). Detecting and removing them right after creation suffices to ensure consistency.



This approach to consistency preservation is an advancement over the practice to reduce the problem to the triangular case, i.e., splitting quads into triangles and then checking the consistency of the resulting triangle mesh, using [DEGN98], as for example in [DSSC08]. By explicitly considering the problem in terms of quads, one allows for legal operations that would be barred by using triangle mesh criteria. For instance, the potential diagonal collapse in the inset would be forbidden since it produces degenerate configurations in the triangle mesh including dotted edges, while it is legal in the quad mesh.

2.3.2.2 Prioritizing operations

Consider a typical closed mesh with n quads and, thus, with about n vertices and $2n$ edges. There is a total of $11n$ potential operations ($2n$ diagonal-collapses, $4n$ edge-collapses, and $4n$ edge-rotations, and n vertex-rotations), plus doublet and singlet removals that can be performed on such a mesh. Clearly, many operations would invalidate other potential operations and create the preconditions for other operations yet. For practical purposes, it is important that the choice of which operation to perform at every iteration is taken very efficiently.

We have seen how we reduced the problem of mesh quality in terms of homeometry. However, equation (2.1) is still a complex objective function, with multiple local minima, awkward to minimize explicitly (using the above or any other set of discrete operations). Finding the global optimum solution for a target number of quads is not practical. Instead, length based heuristics can be adopted that reach a good solution in a much shorter time. Good performance of this approach has been empirically demonstrated, measuring the objective function (2.1) (Sec. 2.3.5).

Prioritizing collapses: The proposed solution is to collapse the shortest element of the mesh. Since a collapse typically causes neighbor elements to expand, systematical removal of the shortest element (either edge, or diagonal) causes the population of survivors to have a similar length. We expect diagonals to be $\sqrt{2}$ times as long as the edges, so we divide their measured length by $\sqrt{2}$ for the purpose of identifying the shortest element.

This evaluation process is simple yet effective, depending only on lengths (which change locally), not on the value of μ (which changes globally) in (2.1); this allows for a practical and efficient implementation (Sec. 2.3.4).

Prioritizing optimizations: Contrary to collapses, rotations do not reduce complexity and are performed just to improve tessellation quality. Their role in our framework is similar to the one edge flips play in triangle mesh optimization [HDD+93]. It can be seen as dual to that of collapses: rotations shorten linear elements that are too long, whereas collapses remove elements that are too short, both contributing to achieve length uniformity.

Each potential rotation is assessed by its *profitability*, a value which is, in first approximation, correlated with the related change of function (2.1). We perform only rotations with a positive effect, starting from the most profitable. This criterion is stated only in terms of length changes, and again it does not depend on the value of μ.

A vertex rotation around v turns edges emanating from v into diagonals, and viceversa. We consider a vertex rotation to be profitable if, in the current star of v, the sum of the edge lengths overcomes the sum of the diagonals. The difference between the two quantities is the corresponding amount of profitability.

The purpose of edge rotations is to eliminate overlong elements (while short ones are collapsed). An edge rotation affects only one edge and two diagonals, each in a different

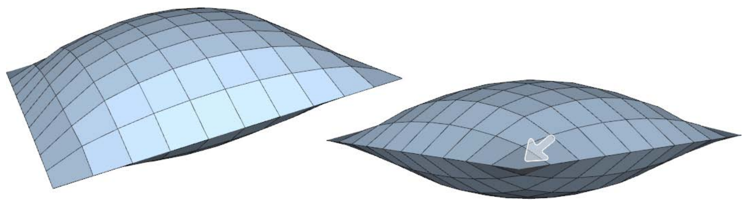


Figure 2.6: *The doublet in each corner of the "pillow" dataset (two views shown) can be considered a good meshing solution, because of local extreme Gaussian curvature.*

quad (the other diagonal is unaffected). We consider an edge rotation to be profitable if it shortens the rotated edge and both such diagonals. Profitability is the amount of shortening.

Prioritizing cleaning operations: Cleaning operations are not scheduled, but they are performed during both steps i and ii of the simplification algorithm as soon as any degenerate configuration is detected.

In the literature, doublets are considered degenerate configurations. In fact, in a fully homeometric mesh, a doublet necessarily corresponds to a pair of geometrically coincident faces with opposite orientations. However, a doublet can represent an optimal configuration in regions with extremely high positive Gaussian curvature, as depicted in Fig 2.6.

If application dependent considerations dictate that doublets are to be considered degenerate, then they are removed. This is also the best route in case the original mesh does not present regions with extreme positive Gaussian curvature. However, we have also the alternative of keeping "good" doublets, by inhibiting this cleaning operation. "Bad" doublets are neither detected explicitly (e.g., by measuring curvature), nor treated as a special cases: they are just removed with a diagonal collapse when their geometric shape makes that collapse to become the next operation.

Singlets instead are always degenerated configurations and, as such, they are removed as soon as they appear.

2.3.2.3 Tangent space smoothing

This operation consists in moving vertices so that they never leave the surface of the mesh and, at the same time, the overall homeometry (2.1) is increased. For a better match between the simplified model and the original mesh M_0, vertices are kept on M_0, rather than on current mesh M_i. Factor μ of equation (2.1) is computed for the current mesh M_i, to account for the minor area deformations occurring during coarsening.

Smoothing is performed through a relaxation process and it has two main purposes: first, by maximizing homeometry, it directly improves mesh quality; second, and more importantly, it helps selecting the best candidate operation to perform next. The rationale is that the elements that cannot be made homeometric by smoothing are good candidates for the next collapse/rotate operation. For example, when the number of quads incident at a vertex is too high with respect to what is required by the local Gaussian curvature, then, *even after smoothing*, one diagonal of each such quad will be shorter than the prescribed one ($\sqrt{2}\mu$). As such, that quad may be selected for collapse.

Depending on the initial data, mesh M_0 can be very far from being homeometric. Thus, global tangent smoothing is applied to mesh M_0 during Step 1 of the algorithm until convergence. Global tangent smoothing is also applied to improve the final mesh M_n during step 3, similarly to [DSC09a, DSSC08, SDW$^+$09].

Conversely, smoothing operations performed during Step b are localized to a small area, just around the regions of influence of the local operations preceding it. Vertices affected by local operations during Steps i and ii are initially scheduled for smoothing during Step b. In case any such vertex is moved during smoothing for an amount larger than a given threshold, then also its neighbors are scheduled for smoothing.

2.3.3 Extending the method

The method described in the previous section lends naturally to two useful extensions: creating meshes with varying, customizable tessellation densities (Sec. 2.3.3.1); and creating meshes that preserve feature lines (Sec. 2.3.3.2).

2.3.3.1 Customizable tessellation at variable density

Since both the smoothing phase and the selection phase are length based, it is easy to make the system produce tessellation densities that vary over the surface, according to a given importance function taken in input. This function $\lambda(p)$ is defined over the surface M_0 and determines the required tessellation density around each point p of the surface.

Since the average edge length is already implicitly defined by the number of elements $|M_i|$ and the total area of M_i, we make $\lambda(p)$ define the prescribed *ratio* between the edge lengths around p over average edge length (and similarly for the diagonals). This function is normalized so that the area-weighted average of $1/\lambda^2$ over all M_0 is 1.

During the smoothing phase, the objective function (2.1) is minimized using $\mu\lambda(p)$ instead of μ, p being the center of the given element. During the selection phase, length of edges and diagonals are divided by $\lambda(p)$ for the purpose of identifying the next element to be collapsed.

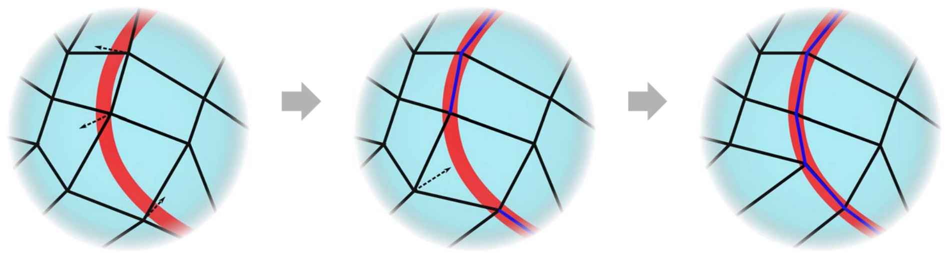


Figure 2.7: *Preservation of a feature line (red line). Vertices are moved on the feature line (left, middle). This causes some quads to fall across the line: moving a third vertex on the line solves the issue (right).*

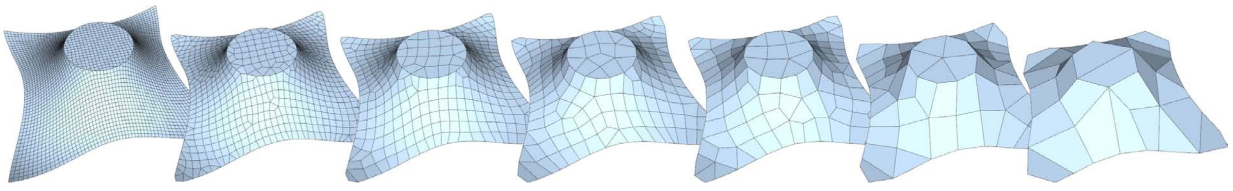


Figure 2.8: *Feature lines preserved though the simplification process (TableCloth dataset, simplified from 3.25K to 40 quads).*

Depending on the application, importance functions can be either defined by the user (for example, in order to devote more vertices to regions of interest of a model), or precomputed by geometry processing performed over M_0 and/or its attributes, so to achieve an adaptive simplification. Since the resulting mesh has no T-junctions, spatial changes in the resolution cannot be sudden, so λ is supposed to vary smoothly over the mesh.

2.3.3.2 Preservation of feature lines

In many contexts it can be useful to preserve feature lines in the quad mesh. Feature lines can be either prescribed by the user, or automatically identified, e.g., as: attribute discontinuity lines; creases extracted from the original geometry in CAD models; or high curvature lines extracted by analyzing the geometry of range scanned models. In open meshes, borders should also be preserved.

Our objective is to make feature lines emerge as collections of edges in most levels of detail produced by the simplification process. This is achieved by acting just in the tangent smoothing phase, adding a term in the cost function, which penalizes vertices that are close but not over such lines. In other words, feature lines are made to attract nearby vertices toward them. The radius R at which this happens is computed as a fixed fraction k of the average edge length in that zone. In our experiments we used $k = 3/4$.

For each vertex v_i we find the closest point f_i on a feature line with the help of a spatial index, and we compute an attraction factor a_i, which is equal to 1 if v_i is on f_i and decreases linearly to vanish at distance R.

Uncontrolled snapping of vertices to nearby features may cause some quads to fall across them, i.e., to have their diagonal aligned with the feature line. This happens when two opposite vertices v_a and v_c of the diagonal of a quad (v_a, v_b, v_c, v_d) are attracted to the same line, while vertices v_b and v_d are not. This situation is healed by making one of v_c and v_d also move toward the same line, regardless of their distance from it (see Fig. 2.7).

Use of a scaling factor $k < 1$ for the radius of influence R ensures that both endpoints of an edge are attracted to a feature line only if that edge is sufficiently close to the feature itself. In this way, we try to avoid compressing quads around features, which would be against our objective of homeometry. Even when that undesirable situation occurs, compressed quads soon disappear because their elements (either diagonals or edges) are selected for collapse.

Figure 2.8 shows a how this method performs on a simple example containing a circular feature: even in that challenging case (for quad meshing) both the feature and homeometry are maintained to some degree during the simplification.

An advantage of this approach is that feature lines are enforced, rather than just preserved, meaning that they need not be present as edges in the input mesh. For example, CAD models often contain explicit sharp crease lines that are prescribed during design, while scanned models rarely contain exact sharp creases.

2.3.4 Implementation details

Prioritizing operations. Finding the top priority operation can hinder performance in a naive implementation, because of the linear search it involves. As in standard iterative simplification approaches [LRC$^+$02], a good solution is to build a heap of potential operations sorted by prerecorded priorities. Every time a local operation is performed, only the potential operations related to neighboring elements are affected and must be updated in the heap.

This is possible because the priority criterion does not depend on the average edge length μ as defined in Eq. (2.2), i.e., priorities do not change when the global number of faces changes (otherwise, the entire heap would be invalidated at each simplification step). This is one reason to use a heuristic criterion (Sec. 2.3.2.2), rather than trying to explicitly find the operation that minimizes (2.1).

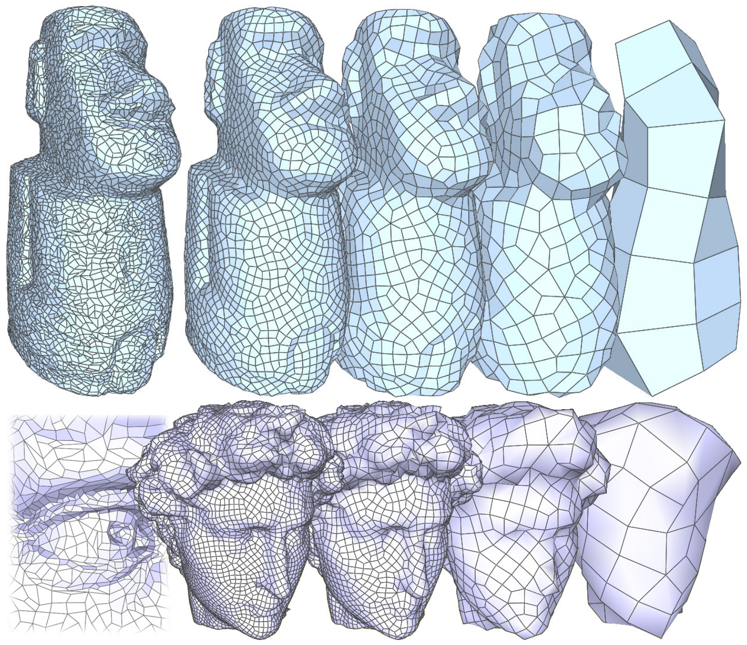


Figure 2.9: *Examples of models simplified at decreasing number of quads (including extremely low ones, useful for base mesh or as a base for a refinement). Above, Muai dataset: 8.2K (original), 3.3K, 1.4K, 600, and 40 quads. Below, David dataset: 100K (original, detail), 10K, 5K, 1.2K and 150 quads.*

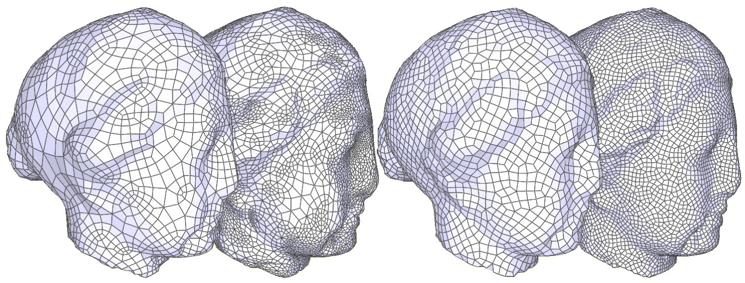


Figure 2.10: *Simplified mesh pairs (5K and 3K) obtained from Igea dataset with [DSC09a] (left) and our approach (right). For a quantitative assessment, refer to table 2.1.*

Tangent space smoothing. There are several possible ways to perform smoothing. One would be to minimize objective function (2.1) iteratively in 3D space and, after each iteration, re-project every vertex to the surface of M_0 via some spatial indexing structure [THM$^+$03]. Otherwise, if an almost isometric parametrization of M_0 is available, smoothing can be solved in parametric space. In this case, gradient descending vectors are computed in 3D, then converted to 2D vectors via the mapping defined by parametrization. In our experiments, we adopted the latter approach, making use of a parametrization produced by the method described in [PTC09], which provides a reasonably isometric, seamless, globally smooth parametrization of M_0. Any other technique for smoothing in tangent

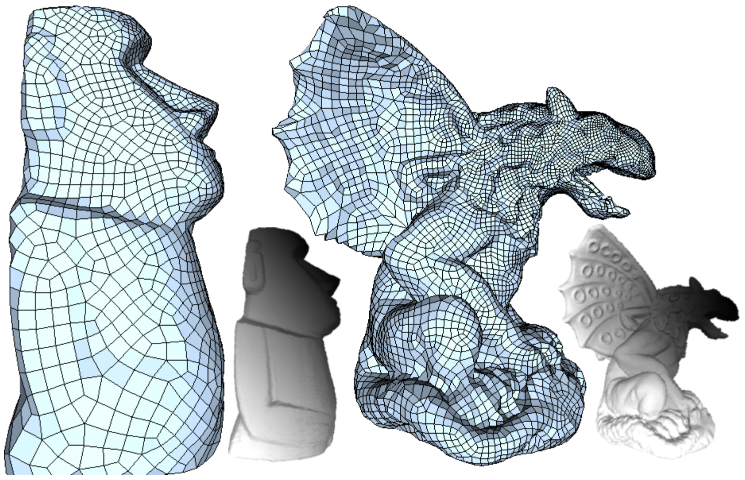


Figure 2.11: *Examples of models simplified following a user defined importance map (shown in small: darker regions correspond to higher density).*

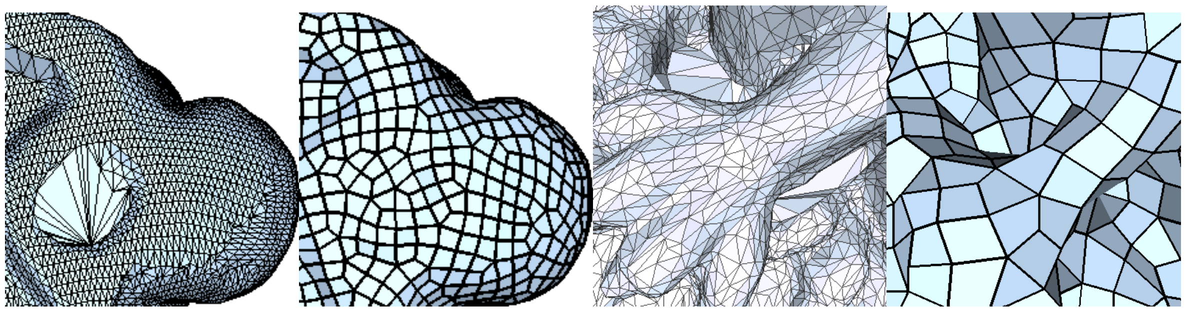


Figure 2.12: *Examples showing robustness to unevenly sized or poorly shaped initial triangles. Left: original. Right: output (close-ups of Bunny and Gargoyle datasets).*

space could be used instead.

We perform tangent space smoothing with an iterative, explicit solver that lets the mesh relax as in a mass and spring system [MHHR07], where the rest position of each spring coincides with the ideal length of its associated edge or diagonal, i.e., either μ or $\sqrt{2}\mu$, respectively.

In Step b of the algorithm, smoothing is stopped relatively early, i.e., after a fixed number of iterations (we used 20), because a coarse minimization of function (2.1) is sufficient in practice to guide the selection of the next operation. Smoothing is performed until convergence in Steps 1 and 3.

In Setp 1, a global smoothing would change the geometry of the entire mesh and, thus, it would invalidate the precomputed priority of all potential operations. To avoid that, smoothing is done with the incremental scheme described in Sec. 2.4.1.5.

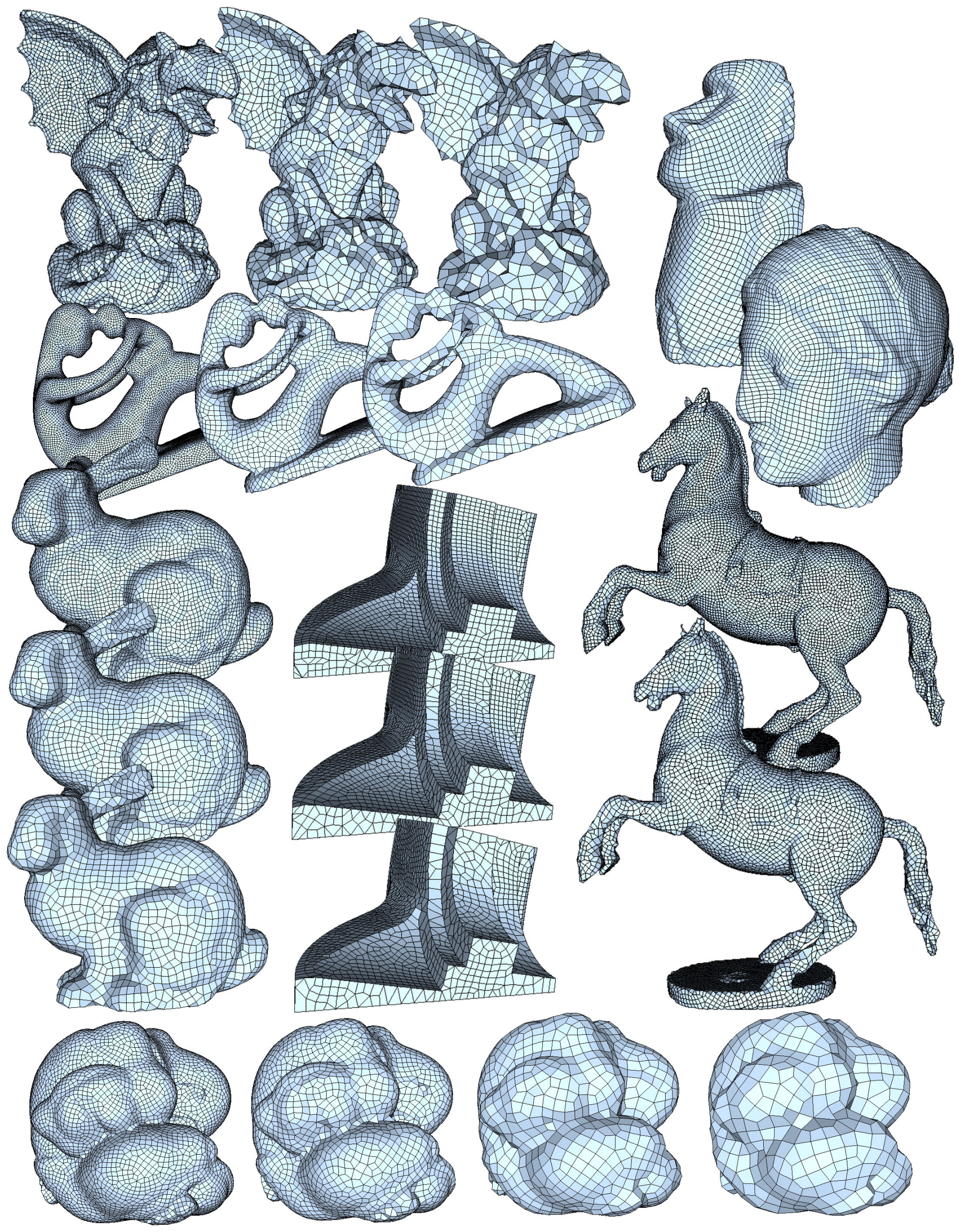


Figure 2.13: *Different datasets shown at various simplification steps. Refer to table 2.1 for numerical data. Original models are not shown (see also attached video). Top right, in blue: two semi-regular models obtained by regularly subdividing extremely simplified models (see text).*

Feature line extraction. Any feature line, whether it is present or not in the original mesh as a collection of edges, can be enforced during the simplification process. Thresh-

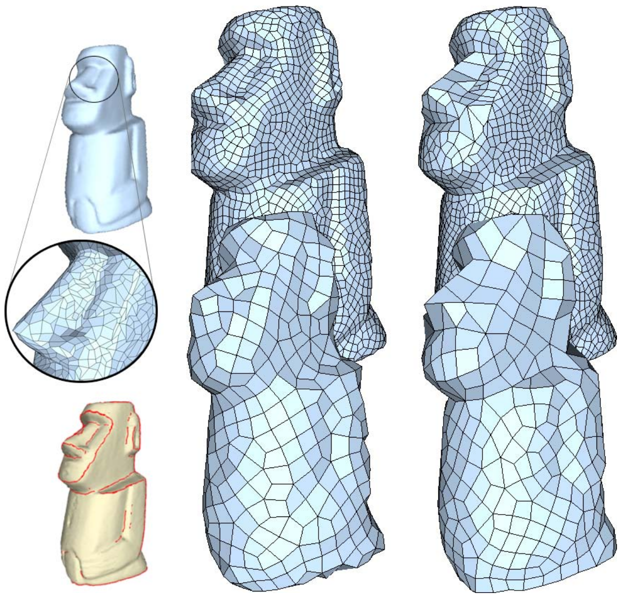


Figure 2.14: *Left: input of the simplification algorithm (top: input mesh M_0; bottom: feature lines extracted from M_0). Center: models simplified with basic algorithm. Right: models simplified with feature preservation. See also tab. 2.1.*

olding dihedral angles is sufficient to extract sharp creases from CAD models, such as the fandisk or the tableCloth datasets. For range scanned models, such as the Mohai dataset, creases are extracted with a modification of the method proposed in [HPW05] (see Fig. 2.14).

2.3.5 Results and discussion

We tested our quad simplification method on quad meshes converted from triangle meshes (Appendix A) coming from either range scanning, or CAD. Examples of results are illustrated in Fig. 2.8, 2.14, 2.9, 2.10, 2.11, 2.12, and 2.13. A numerical assessment is given in Table 2.1.

Discussion: In spite of using only local connectivity operators, the simplified models show a very good quality, throughout all steps of simplification. Regular results are obtained independently from regularity of the input. Tangent space smoothing has proven to be very effective not only to improve the final result, but also – and more importantly – to drive the selection of the operator to be performed at the next cycle. This concept makes the process robust and general, so quad mesh simplification can be easily addressed in spite of the intrinsically harder challenges it poses with respect to the case of triangle meshes. Our length based approach is also efficient, scalable, robust with the initial tessel-

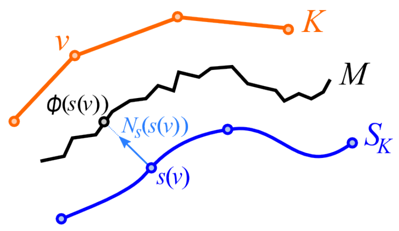


Figure 2.15: *The reference mesh M, the control mesh K and the subdivision surface S_K; a vertex v of K has its limit position at $s(v)$ and N_S is the normal at the limit point. Projection ϕ of S_K to M is a normal displacement.*

lation quality (see Fig. 2.12) and easily extended to allow for feature line preservation and varying controlled tessellation densities. The system is capable of reducing an initial quad mesh to extremely low number of faces (see Fig. 2.9, right). Semi-regular meshes can be obtained from such a simplified model, by a regular subdivision and re-projection onto the original surface. Examples of results obtained this way are shown in Fig. 2.13 (top-right).

We compared our approach, which strives to maximize homeometry, with [DSC09a], which strives to maximize regularity (and, to a lower extent, geometrical faithfulness). Models maximizing the sought objective are always obtained (Table 2.1). In terms of Hausdorff distance, however, the presented strategy is around twofold better, suggesting that our objective is more suited for that. This is probably also due to the better sampling distributions that is implied by homeometry. The same approach presented here can probably be applied to triangle mesh simplification in contexts where quality of meshing is a concern as much as geometric similarity. Our method naturally lends itself to the construction of progressive and CLOD models; however, further investigation is needed to efficiently incorporate the changes occurring through simplification in the data structure.

Our current method has some limitations. Geometric fidelity is not measured and it is addressed only indirectly; the general objective to place extraordinary vertices at points that concentrate strong Gaussian curvature is achieved, but only partially; vertex snapping, used to enforce features, can produce nearly triangular elements that would be harmful for, e.g., numerical analysis; alignment of edges to curvature directions has not been addressed. These issues might be addressed by suitable modifications of the objective function in the context of the same framework.

2.4 Automatic Construction of Catmull-Clark Subdivision Surfaces

In the previous Section, we started from an irregular triangle mesh and we produced a coarse quad mesh, where every quadrilateral approximates a part of the surface with a bilinear patch. In this section, we extend the algorithm to produce a quad mesh that is the control grid of a Catmull-Clark surface that approximates the input surface. The problem is more challenging and requires further analysis of the original mesh, but the core idea of the algorithm is similar.

We take in input a mesh M and we want to build a CC subdivision surface S_K, with control mesh K, that fits M according to requirements stated in Section 2.1.2.

We start with a subdivision surface interpolating M at all its vertices, and we progressively simplify K through local operations. Let v be a vertex of K: $s(v)$ denotes its limit position on S_K; $N_S(v)$ denotes the surface normal at $s(v)$; $\phi(s(v))$ denotes the *projection* of $s(v)$ to M along direction $N_S(v)$. Symbols are summarized in Figure 2.15.

2.4.1 Algorithm

The algorithm has the following outline:

1. Analyze input mesh M and compute its *Fitmap*, which is made of a pair of scalar fields that will be used to drive simplification during Step 3 (Sec. 2.4.1.1);
2. Compute initial control mesh K, having the same connectivity of M, and such that S_K interpolates the vertices of M (Sec. 2.4.1.2);
3. Iteratively process control mesh K. At each step:
   (a) Perform a diagonal collapse, followed by edge swaps and/or cleaning operations, if appropriate (Sec. 2.4.1.3);
   (b) Fit and smooth positions of vertices in the area affected by the previous operation(s) (Sec. 2.4.1.4);
4. Globally fit S_K to M (Sec. 2.4.1.5).

We describe our method to work on a watertight model, its extension to models with boundary being just straightforward. We assume M to be a discrete representation of a smooth manifold. Smoothness is intended in a relaxed sense here: meshes with sharp edges can also be taken in input, but they will be treated as discrete approximations of smooth manifolds containing zones with very high curvature. Since we model our output with smooth CC surfaces in output, some smoothing of sharp creases is unavoidable, and this represents a limit of our current method. See Section 2.5 for a possible extension to explicitly represent sharp creases.

Throughout the algorithm, we adopt the method proposed by Loop and Schaefer [LS08] to evaluate the limit point $s(p)$ and its surface normal $N_S(p)$ for an arbitrary point p sampled on control mesh K. A spatial index on M is used to support ray-casting to evaluate

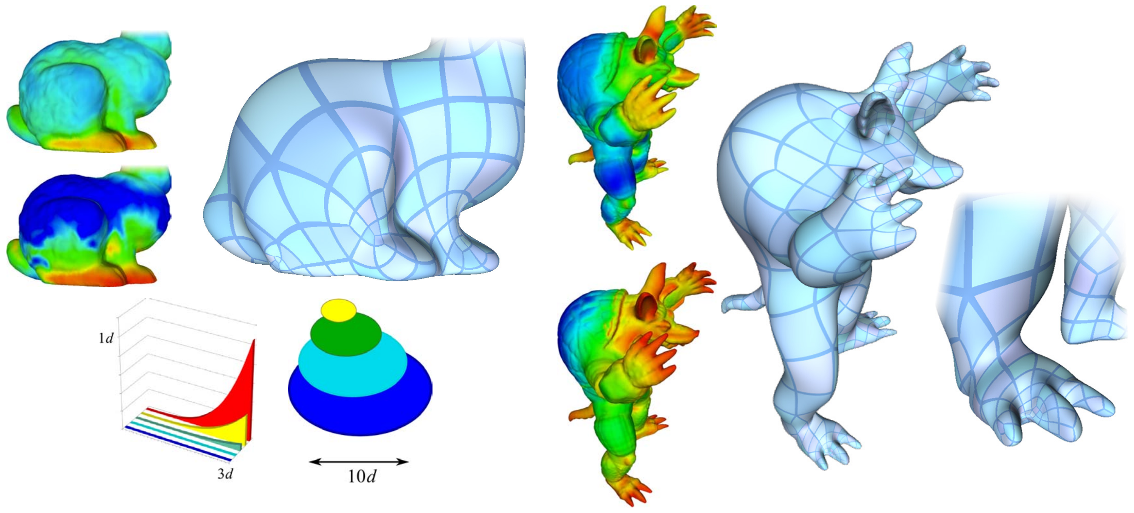


Figure 2.16: *The two channels of the Fitmap (S-fitmap upper; M-fitmap lower), together with subdivision surfaces built by our method, with patches outlined: adaptive distribution of patches follows the Fitmap. Color coding is depicted at the bottom left (d is* 1% *of the bounding box diagonal): color of the S-fitmap (left) represents the coefficient a of the model of RMS residual; color of the M-fitmap (right) corresponds to size limit for the patch.*

function ϕ.

2.4.1.1 Mesh analysis

During Step 3 of the algorithm, we need a mechanism to select local operations, as well as a halting criterion. The general idea is that conciseness is achieved through simplification, which should proceed as long as the surface maintains acceptable regularity, accuracy and projectability. Therefore, we wish to select, at each iteration, the local operation that best preserves such criteria. Regularity is somehow easy to test on a local basis, and it is addressed explicitly by the edge swap operations performed in Step 3a and by smoothing performed in Step 3b. On the contrary, variations in accuracy and projectability are very expensive to test, involving surface fitting and normal projection of the portion of mesh affected by an operation. For this reason, we rely on an analysis of the input mesh, both to drive the selection of collapse operations and to halt simplification.

The general idea is to estimate, for each point p of the input mesh, how well neighborhoods of p can be modeled, in terms of accuracy and projectability, by using a single patch. This analysis provides a rough estimate of how patches generated during simplification should behave.

Fitmaps A Fitmap consists of a pair of values for each point p of a surface M: the *S-fitmap* ("Scale" fitmap) estimates how the RMS error of fitting a patch P to a neighborhood B of p increases with radius of B; the *M-fitmap* ("Maximal radius" fitmap) estimates how much a patch P can extend around p before correct projection of P to M through normal displacement becomes impossible. The Fitmap is computed at vertices of M and extended by linear interpolation to its faces (See Figure 2.16).

The Fitmap of mesh M can be interpreted as a prescription on the patches of an ideal approximation S:

- The local radius of each patch of S should be inversely proportional to the value of the *S-fitmap* computed at its central point;
- No patch of S should have a radius larger than the value of the *M-fitmap* computed at its central point.

The first condition aims at distributing error evenly, while the second condition aims at preserving projectability.

Building the S-fitmap In order to estimate the S-fitmap $\mathcal{F}_S$, we proceed as follows. For each vertex p of M, we consider neighborhoods $B_{p,i}$ of p of increasing radii r_i, for $i = 0 \dots h$. For each i, we collect all vertices of the input mesh in $B_{p,i}$: vertices are gathered with a breadth-first traversal of M starting at p; for simplicity, we use Euclidean distance instead of geodesic distance to stop the search. In all our experiments, we set $h = 8$, r_0 equal to the average length of edges of M, r_h equal to 1/4 the length of the diagonal of the bounding box, and we distribute the other radii on an exponential scale.

Next, we express $B_{p,i}$ as the graph of a bivariate function, on a local frame defined with tangent plane at p, and we fit a cubic polynomial to its vertices. Cubic polynomials serve as an easy and conservative surrogate to the more general bicubic patches that constitute our output surface S, as they are easy to fit and they are independent of the orientation of the local frame. Roughly speaking, we are assuming that bicubic patches of S will do at least as well as we can do with cubic polynomials. Vertices of $B_{p,i}$ are expressed in the tangent frame $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ at p, $\mathbf{u}$ and $\mathbf{v}$ being mutually orthogonal tangent directions at p and $\mathbf{n}$ its normal. Surface $B_{p,i}$ is then expressed as the graph of function $g_{p,i} : \mathbb{R}^2 \to \mathbb{R}$ in the tangent frame, and a linear least squares problem is resolved to fit a cubic polynomial $\tilde{g}_{p,i}(u, v)$ to vertices of $B_{p,i}$. We measure the RMS residual as

$$E(r_i) = \frac{1}{n_{p,i}} \sqrt{\sum_{j=1}^{n_{p,i}} (\tilde{g}_{p,i}(u_j, v_j) - n_j)^2},$$

where summation is run over all vertices (u_j, v_j, n_j) of $B_{p,i}$, and $n_{p,i}$ is their number.

The sequence of $E(r_i)$ values for $i = 0 \ldots h$ provides a sampling of how the RMS error grows in the neighborhood of p. Now, we need to compress information into a single scalar value. To this aim, we model error increase as a function of radius r. Without loss of generality, assume function g_p (i.e., the surface M expressed in the tangent frame of p) admits a Taylor expansion. Let T_3 be the cubic Taylor polynomial of g_p centered at p. Then the error of approximating g_p with T_3 at a single point at distance r from p belongs to $O(r^4)$. Now the RMS error E_{T_3} of T_3 is computed by integrating the square error over the circle of radius r, computing its square root and normalizing this by the area of the circle. It readily follows from integration and normalization that also E_{T_3} belongs to $O(r^4)$. Let $E(r)$ be the RMS error of the best fitting cubic polynomial $\tilde{g}_{p,r}$, then we will have $E(r) \leq E_{T_3}(r) \in O(r^4)$, i.e., $E(r) \leq ar^4$ for some $a > 0$. Thus, we model error with a simple function $E(r) = ar^4$. Having collected h measurements of errors at different radii r_i, we fit (in the least squares sense) such function to these values to estimate parameter a. We have also validated this error model with empirical tests: we have fitted functions of the type ar^k for various values of k to sampled error measures; the best average fit to actual errors was consistently obtained for $k = 4$ on all datasets.

We set the value for the S-fitmap $\mathcal{F}_S(p)$ to $a^{1/4}$, so that we obtain a function that increases linearly with the radius. In this way, if two patches centered in p_0, p_1 have radii r_0 and r_1, respectively, they contribute approximatively the same error if the values of $r_0 \cdot \mathcal{F}_S(p_0)$ and $r_1 \cdot \mathcal{F}_S(p_1)$ are equal.

Building the M-fitmap The M-fitmap $\mathcal{F}_M$ is built together with the S-fitmap. For a given neighborhood $B_{p,i}$ of radius r_i, we sample $\tilde{g}_{p,i}$ and we use a spatial index to cast a ray from each sample point along its surface normal (computed analytically from $\tilde{g}_{p,i}$) to M. Samples are distributed randomly on the neighborhood and their number is $4n_M\frac{|B_{p,i}|}{|M|}$, where n_M is the number of vertices of M, $|M|$ is its total area and $|B_{p,i}|$ is the area of the neighborhood. In this way, the density of samples is roughly twice the density of vertices of M. We flag each triangle hit by a ray and we count the percentage of triangles in the spanned neighborhood that are not hit by any ray, which are (conservatively) classified as flipped faces. The value of $\mathcal{F}_M(p)$ is set to the largest tested radius at which the portion of neighborhood covered by flipping faces is smaller than a "tolerance" threshold τ. Parameter τ can be user-defined, depending on the amount of high frequency noise expected in the input mesh, or on the amount of 3D high frequency detail that could be ignored, to avoid an excessive fragmentation of patches. Parameter τ is the **only** parameter used by our simplification method, and it can be used to trade-off between projectability and size of the simplified surface.

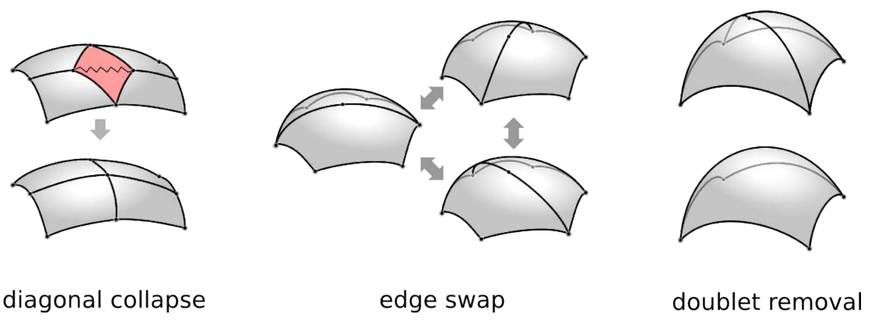


Figure 2.17: *The set of local operators. See details in Sec. 2.4.1.3.*

2.4.1.2 Building the initial control mesh

We first apply a tri-to-quad conversion algorithm [TPC$^+$10] to transform the reference mesh M into a quad mesh M_q having the same set of vertices. We set initial control mesh K to the same connectivity of M_q, while placing its vertices to have the limit surface S_K interpolate the vertices of M. This is done by resolving a sparse linear system

$$LV_K = V_M, \tag{2.3}$$

where V_M is the vector of positions of vertices of M, V_K is the vector of (unknown) positions of vertices of K, and L is the limit subdivision matrix, which is determined by the connectivity of K and by the subdivision masks. Because of its sparsity, this system can be solved easily and efficiently with any sparse solver, even for meshes of large size [eig]. We rather adopted a simple natural fixed point iterative method, which displaces each control point towards its limit position until convergence. This is perhaps not the fastest available method, but it is very simple as it does not require to manipulate L directly, and it is sufficiently fast for our purposes. In fact, this phase takes just about one second for the largest mesh we have processed. Besides, it has the advantage to be also applicable locally: we will exploit this feature during vertex optimization (see Section 2.4.1.4).

Next, we define a mapping ϕ from the subdivision surface S_K to M for every point $p \in S_K$ by taking the closest point of M intersected by a ray cast from p along its normal $N_S(p)$.

2.4.1.3 Local operations

We use a subset of local operators that have been introduced in Section 2.3.2.1 to modify control mesh K throughout simplification (see Fig. 2.17). We have found this subset to be sufficient and effective for our purposes, the quality of results being not different from those obtained with the complete set of operators, while implementation is simpler and more robust. The criteria for applying operators during simplification, which are different from those in 2.3.2.1, are discussed in the following.

Diagonal collapse This is the main operator used to reduce mesh complexity. It eliminates a quad q, two edges and a vertex, by collapsing one diagonal of q.

We maintain a heap of potential collapses, which is kept up-to-date throughout the simplification process. Cost of collapsing a diagonal d is given by $|d| \cdot \mathcal{F}_S(\phi(c))$, where $|d|$ denotes the length of d, c is the center of the patch containing d, and $\phi(c)$ is its projection to M. Collapses are prioritized according to least cost.

Collapsing an element causes its neighbors to expand. Systematically executing the least expensive collapse causes survivors to have a size proportional to the inverse of the S-fitmap, thus producing patches of variable sizes, yet yielding a roughly uniform distribution of the approximation error, as depicted in Figure 2.16. At the same time, where S-fitmap is nearly uniform, equality of diagonal lengths tend to favor rectangular patches. Note that the contrasting objectives of tessellation adaptivity and of patch regularity are sought together, with a natural trade-off, by operating a single criterion on lengths.

The M-fitmap is used to try avoiding collapses that hinder projectability. Given a potential collapse, we evaluate the M-fitmap at the center of surrounding patches that the collapse would extend. We perform the collapse only if, at each such patch, the M-fitmap is smaller than its diameter, measured as the maximal distance between its center and one of its corners. Simplification is halted when no feasible collapses remain.

The vertex generated from a collapsed diagonal is set to its midpoint, prior to optimization (see Section 2.4.1.4).

Edge swap This operator is used to improve the quality of the mesh. It substitutes an existing edge e with one of the other two diagonals of the hexagon formed by the two quads incident at e. Edge swaps have also the side effect of modifying lengths of diagonals, effectively driving the selection of local operations to be performed next.

After performing a diagonal collapse, we consider all faces in the 1-ring of the collapsed element and we test all their edges for potential swap. A swap operation is performed if it improves the valencies of vertices. Given an edge e, let $v_1, \ldots, v_6$ be the vertices bounding the pair of faces incident at e. We measure the valence $D(v_i)$ of each such vertex and we set an energy $\sum_{i=1}^{6} |D(v_i) - 4|$. We swap e if and only if such a swap decreases this energy. In this way, we tend to increase the number of regular vertices of K.

Doublet removal Collapse and swap operations may generate *doublets*, i.e., configurations where two adjacent quads share two consecutive edges, which join at a vertex with valence two. Doublet-removal is applied to eliminate a doublet as soon as it appears, by simply merging the two quads.

2.4.1.4 Local optimization of vertex positions

We wish to maintain surface S_K close to M throughout the simplification process. We do not need to warrant accurate fit, though, since this is done just on the final mesh, during Step 4 of the algorithm (see Section 2.4.1.5).

After each diagonal-collapse (and edge-swap and doublet-removal operations potentially triggered by it), positions of all affected vertices of K must be updated to re-fit S_K to M. Let $W \subseteq K$ be the portion of mesh directly affected by the last local operation(s), plus all the mesh spanned by its 1-ring. We resolve system (2.3) with only the vertices in W as unknowns, while freezing the remaining vertices of K. As W usually contains few dozens vertices, this operation is very fast. This is just an approximation: limit points of vertices in W interpolate M, but vertices in the 2-ring of W may be perturbed and leave M. However such perturbation is generally irrelevant to subsequent processing.

A better approximation to fit could be computed by using the least squares fitting algorithm described in the following subsection, by sampling just faces of W and using just vertices of W as unknowns. However, this solution is more time-consuming and we did not notice any relevant benefit in terms of final result.

In order to obtain more regularly shaped patches, we interleave local fit with Laplacian smoothing: each vertex $w \in W$ is moved midway between its current position and the average of centroids of its incident faces, before being displaced to interpolate M. We empirically found that alternating smoothing and fitting twice is sufficient.

The combined effect of Laplacian smoothing and local fit is equivalent to smooth the vertices of S_K in tangent space, without leaving the surface of M (similarly to [TPC$^+$10], but without the need for a parametrization). This greatly improves the shape of patches.

2.4.1.5 Final fitting

After the connectivity of control grid K has been obtained through simplification, a more accurate and global fitting process is run over the entire K. In this phase quality is crucial, therefore, following [HL93], we add extra equations to the system (2.3) to enforce that not only vertices of S_K but also points sampled inside its patches are close to M. Each patch of the limit surface is sampled with a number of points proportional to its area. Let $|P|$ be the area of patch P, $|M|$ be the total area of M and n be the number of its vertices. We sample P on a $k \times k$ regular grid in its parametric domain, where $k = \lfloor \sqrt{n \frac{|P|}{|M|}} \rfloor$. The limit surface corresponding of each face f of K is evaluated as a Bézier patch, whose coefficients are a linear combination of vertices in the 1-ring of f [LS08]. Let p be a point sampled on f, with parametric coordinates (u, v). The position of $s(p)$ on the limit surface is defined in terms of (u, v) and of the coefficient of the Bézier patch $s(f)$. This allows us filling the

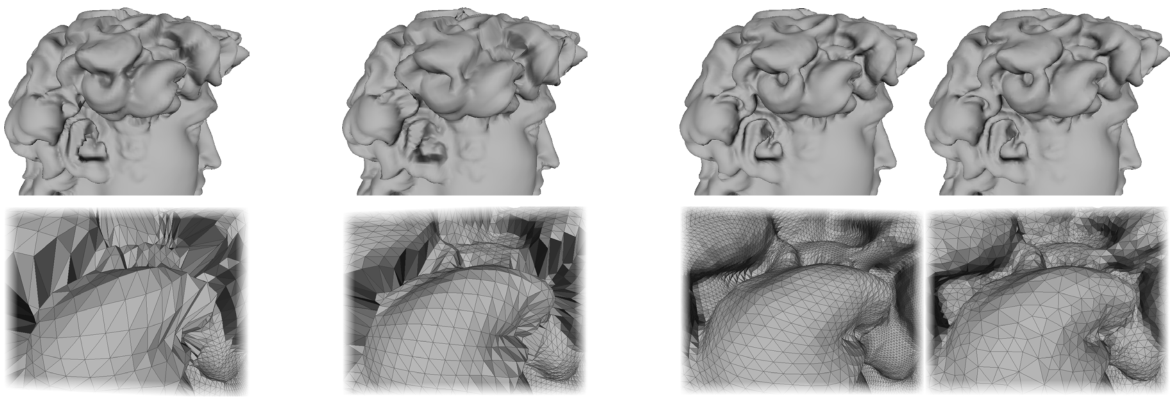


Figure 2.18: *Comparison of displaced subdivision surfaces of the David's hair. Because of higher projectability, we produce a more accurate approximation of the original mesh using a similar number of patches. In this example patches are sampled uniformly* 10×10.

corresponding row of matrix L in system (2.3). Projection $\phi(s(p))$ of $s(p)$ on M is used as a target position for $s(p)$, which is plugged into the corresponding position of column V_M in system (2.3). The resulting overdetermined sparse linear system is solved in the least squares sense using a sparse solver from the Eigen library [eig].

A better fit could be probably obtained by using a more advanced technique, e.g., based on the *square distance minimization (SDM)* introduced in [PLH02]. However, control mesh simplification is independent on the technique used for final fitting, thus we did not explore this possibility so far.

2.4.2 Experimental results

The proposed method was tested on several datasets coming from range scanning. Some results are shown in Figure 2.21. In Table 2.2, we provide statistics in terms of the four requirements outlined in Section 2.1.2:

1. *Conciseness* is reported as the number of patches of surface S_K (faces of K) with respect to the number of faces of M;
2. *Accuracy* is reported as the RMS error of approximating M with either S_K, or its normal displaced surface;
3. *Regularity* is reported as the number of irregular vertices in the control mesh K;
4. *Projectability* is reported as the percentage of surface M that is not reached correctly by normal projection from S_K (0.0% means perfect projectability).

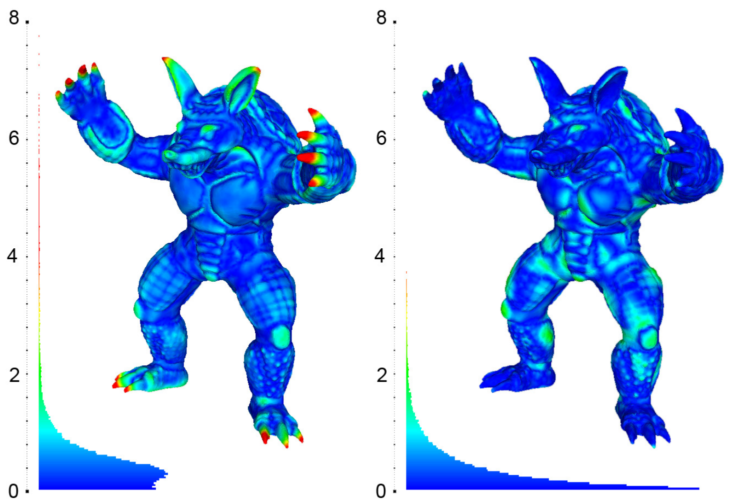


Figure 2.19: *Comparison of error distribution of the Limit surface for the meshes built with MI and our approach as reported in table 2.2 with a and b.*

Parameter τ of the M-fitmap, as defined in 2.4.1.1, is used to obtain more or less simplified meshes, depending of the tolerance on the loss of projectability: higher tolerances allow for more drastic simplifications. The set values turned out to be very conservative in practice. As shown in table 2.2, we used tolerances of 1% and 10%, but the projectability of results is always larger than 99%, except for the David dataset which shows 4% of non-projectable surface if the tolerance is set at 10%. Overall, our meshes achieve better projectability than those produced with other methods, for a comparable, and sometimes even much smaller, number of patches.

Subdivision surfaces with a relatively large number of patches, which can be obtained by setting a small tolerance for the M-fitmap, or by stopping simplification before its completion, approximate the input shape well even without displacement. In such cases, normal mapping is sufficient to achieve a reasonably good quality of rendering. Conversely, even drastically simplified surfaces made of few patches can achieve almost perfect quality through displacement mapping.

In order to test the effectiveness of the M-fitmap to prevent losing projectability, we have run some experiments by inhibiting tests on the M-fitmap, thus allowing all collapses to be performed. The process is stopped manually after very drastic simplification. As shown in Figure 2.20, an extremely simple subdivision surface made of just 20 patches is still able to give a reasonable reconstruction through displacement mapping, but relevant artifacts appear.

The proposed method works with real world objects of medium resolution. In case a subdivision surface, with a reasonably small number of patches, is to be extracted from a large high resolution mesh M, it is possible to compute the Fitmap (and the spatial

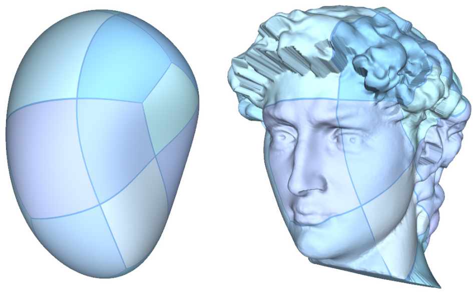


Figure 2.20: *A subdivision surface made of just 20 patches obtained from the original mesh of 50,446 triangles by deactivating tests on the M-fitmap. The overall shape is still preserved but relevant artifacts appear due do severe loss of projectability. Surface without (left) and with displacement mapping (right) - see artifacts on hair and ear.*

index necessary for ray casting) on the original mesh M, while starting simplification from a simplified mesh M' with a smaller size, which can be computed with any standard program for triangle mesh simplification. The control mesh K is initialized on the basis of M', but all computation, including fitting and mapping, is referred to the original mesh M. We followed this approach for running experiments on the Armadillo dataset, starting with a mesh M' with about 100K triangles.

The technique is somehow time consuming, partly because the code was not optimized for fast prototyping. However, timings reported in Table 2.3 are reasonable considering that this is a pre-processing computation. See Section 2.5 about possible optimizations.

2.4.2.1 Comparison with literature

We compare our results against the ones obtained by first computing a control mesh with alternative methods, then fitting a Catmull-Clark surface with the algorithm described in Section 2.4.1.5. We tested two remeshing algorithms, *Mixed Integer (MI)* [BZK09] and *Periodic Global Parametrization (PGP)* [RLL+06], and one clustering method, *Variational Shape Approximation (VSA)* [CSAD04].

PGP meshes have been computed with the publicly available *Graphite* software, extracting the coarsest possible mesh with standard parameters. Since produced meshes containts triangles, one step of Catmull-Clark subdivision was needed to remove them.

MI meshes (provided by the authors of [BZK09]) are available only for some datasets. In the David dataset MI places a large number of cone singularities in areas containing small features (hair curls), so that model cannot be coarsened below 39K quads. A much coarser version has been produced by *manually* placing just 8 cone singularities.

VSA meshes were produced with a software provided by the authors of [RLL+06]. The

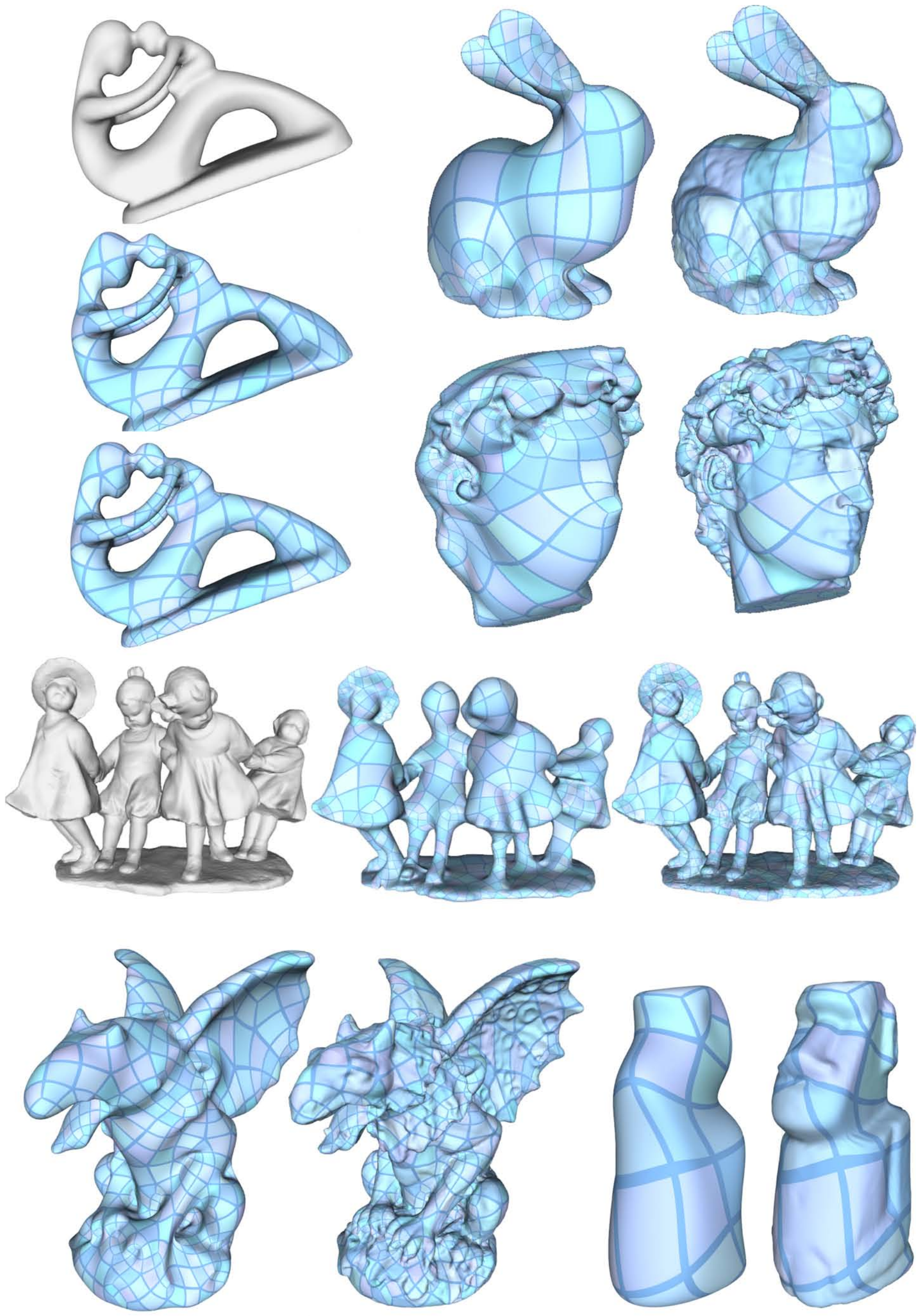


Figure 2.21: *Examples of meshes computed with our system. From the left: original mesh M in white; subdivision surface S_K; and displaced subdivision surface.*

process is semi-automatic: the user chooses the number of seeds and when to stop optimization. We have tried to roughly match the coarseness of meshes extracted with our method, or the coarsest which could be obtained before results differed too much from the input mesh for the fitting algorithm to work. Again, one step of Catmull-Clark was necessary to get rid of triangles in the in resulting models.

Results show that our method is consistently able to obtain much coarser meshes than competitors. In most cases the number of quads is lower by about one order of magnitude with respect to other methods, yet perfect or almost perfect projectability is maintained.

To assess accuracy, in Table 2.2, the RMS error of limit and displaced surface is measured in 1000th's of the bounding-box diagonal of the object (in each experiment, the displacement maps are obtained by sampling each patch with a fixed resolution chosen so that the total number of samples roughly matches the number of vertices of the original mesh). For sake of comparison, in a few experiments we forced our Fitmap based approach to produce base meshes with a face-count similar to the best competitors (rows marked with a $*$ in tab. 2.2). The results show a comparable, often better, RMS error obtained with our method. Moreover, visual comparisons (e.g. Fig. 2.1) and error distributions (Fig. 2.19) reveal that our method better preserves local small scale features, something that the RMS error fails to clearly detect, being averaged over the entire surface. Again, we stress that one strength of our method consists in the ability to reduce the face-count of base mesh drastically more than other methods can, while preserving good accuracy and projectability.

In terms of regularity, comparison yields mixed results. Note that, with our method, presence of a few extra irregular vertices is implied by the adaptiveness of the tessellation, as irregular vertices are unavoidable in zones of transition among different levels of detail (e.g. from the back to the arm to the fingers in the Armadillo, or from the cheek area to the hair area of the David). The MI gives fewest irregular vertices with some datasets (Gargoyle, Fertility and Armadillo), and a comparable number with the David (unless, predictably, singularities are placed manually). Our method performed comparably with PGP and VSA, resulting in more regular or less regular results depending on the dataset and the used parameters. *Highly* irregular vertices (valency > 5) are very rare with our method (unlike, for example, with VSA). VSA and MI in some cases also generated doublets (valency 2 vertices), whereas our method is bound to never output them.

2.5 Concluding remarks

We have presented a fully automatic method for building a coarse control mesh for approximating an input shape with a Catmull Clark surface. Our method produces surfaces with low complexity, good quality and accuracy, and almost perfect projectability. In contrast with standard quad-simplification and quad-remeshing approaches, our method explicitly works on a control mesh for subdivision, and the limit surface is targeted throughout the process. To our knowledge, this is the first method to address the problem in this integrated way, in the context of quad-based subdivision surfaces.

A key issue here is adaptiveness of patch density to geometrical complexity. To this aim, we have introduced Fitmaps, which provide an *a-priori* estimation for the ideal localized patch densities. Fitmaps demonstrate to be very effective at driving the local simplification process to build better control meshes that adapt locally and concisely to fine details. The concept of Fitmaps can probably be adapted to any other form of parametric surfaces, as well as to the simpler case of adaptive mesh simplification.

Surfaces produced with our method are suitable for GPU-assisted rendering via displacement mapping. The method may also support reverse engineering, by providing initial control grids that may be adjusted and refined with modeling tools.

Our current method has some limitations, though. Compared to global remeshing methods (like [BZK09]), it produces meshes containing more irregular vertices. Some such vertices are necessary to warrant transitions through different levels of resolution inherent to adaptive tessellation. However, we believe that irregular vertices could be further reduced by more careful tessellation. The definition of a good balance between adaptivity and regularity demands further investigation.

Another problem is the lack of alignment of patch boundaries to either curvature directions, or other line features. Alignment to features could be enforced with a snapping mechanism, similarly to [TPC$^+$10]. The same mechanism can be also used to model surfaces with sharp creases, by combining it with the extension of subdivision surface evaluation presented in [KMDZ09]. Also this issue requires further investigation.

The time performance of the method can be certainly improved, possibly of orders of magnitude. The main bottlenecks come from subdivision surface evaluation, which is performed many times at many samples during simplification, and by the extraction of large neighborhoods during the construction of fitmaps. Surface evaluation could be easily delegated to the GPU. The extraction of large neighborhoods could be made faster by using an ad-hoc data structure, such as a hierarchical spatial index supporting range queries according to geodesic distance [RGPP11].

	$\|M\|$	val max	reg (%)	Homeometry min	Homeometry max	Homeometry var	Dist 10^{-3}
ideal values:		—	—	1	1	0	0
Moai	*8.2K*	9	44	0.18	3.8	0.49	0
(11sec)	3.3K	6	64	0.60	1.82	0.16	0.5
	0.6K	6	62	0.70	1.73	0.17	1.7
Pensatore	*15K*	8	48	0.12	3.4	0.44	0
(28sec)	10K	7	61	0.58	2.01	0.19	0.3
	5K	7	57	0.68	2.17	0.19	0.7
	2K	6	68	0.59	1.76	0.18	1.4
	1K	7	64	0.6	2.01	0.19	2.0
Gargoyle	*25K*	8	46	0.15	12.22	0.53	0
(63sec)	11K	7	61	0.2	2.8	0.17	0.5
	4K	7	57	0.5	2.28	0.19	1.2
	2K	7	54	0.58	2.31	0.20	2.0
Igea	*25K*	8	48	0.12	4.09	0.48	0
(66sec)	12K	7	65	0.64	2.05	0.16	0.24
	3K	6	67	0.69	2.11	0.16	0.7
	3K[DSC09a]	6	80	0.22	3.74	0.41	1.7
Bunny	*35K*	12	88	0.1	14.53	0.34	0
(85sec)	11K	7	69	0.66	3.3	0.19	0.3
	5K	7	68	0.65	2.8	0.18	0.5
	5K[DSC09a]	6	93	0.24	4.87	0.41	1.1
	3K	6	61	0.47	2.45	0.17	0.7
Fertility	*40K*	8	47	0.09	9.56	0.48	0
(115sec)	22K	6	63	0.59	3.95	0.18	0.12
	5K	6	67	0.65	2.94	0.17	0.4
	5K[DSC09a]	7	65	0.23	6.4	0.48	0.8
	2K	6	67	0.61	2.22	0.19	1
	2K[DSC09a]	7	81	0.23	4.20	0.46	2.5
	3.3K	7	71	0.60	1.81	0.15	0.56
	3.3K[BZK09]	5	98	0.59	3.20	0.22	0.61
Rampart	*50K*	9	48	0.08	7.58	0.50	0
(174sec)	20K	7	75	0.29	3.19	0.21	0.2
	10K	7	62	0.34	2.66	0.21	0.35

Table 2.1: *Simplification results on various datasets. The computation times required for simplifying the initial dataset into the coarsest model (on a Intel Core2 2.4Ghz 2.00 GB). For each quad-mesh (input and simplified), we report: vertex valency (max and % of regular vertices); homeometry (min, max and variance of edge or diagonal length, all normalized with ideal length μ); and Hausdorff distance (computed with [CCR08]), with respect to bounding box diagonal. When possible, results from [DSC09a] and [BZK09] are reported too (the latter is a quad-remeshing approach).*

Mesh ($\triangle$)	Alg.	$\square K$	Irreg. vert.	Unp. (%)	RMS Error L	RMS Error D
Garg. (49k)	PGP	742	127	0.3	4.05	0.23
	MI	2904	83	**0.0**	1.00	0.02
	VSA	2946	816	0.1	1.46	0.06
	1%	1096	345	**0.0**	2.36	0.07
	10%	**493**	182	0.4	4.50	0.15
	*	2897	821	**0.0**	1.32	0.03
Moai (52k)	PGP	830	61	**0.0**	1.30	0.20
	VSA	240	61	**0.0**	2.62	0.13
	1%	92	30	**0.0**	5.00	0.12
	10%	**41**	16	**0.0**	8.22	0.18
Bunny (69k)	PGP	1042	114	**0.0**	1.98	0.13
	VSA	450	104	**0.0**	3.03	0.11
	1%	506	216	**0.0**	3.41	0.08
	10%	**218**	72	**0.0**	8.48	0.14
Fert. (80k)	PGP	3784	176	**0.0**	0.34	0.04
	MI	3357	48	**0.0**	0.28	0.10
	VSA	2772	739	0.2	2.16	0.03
	1%	489	194	**0.0**	2.35	0.04
	10%	**323**	130	0.6	5.10	0.40
	*	3352	956	**0.0**	**0.13**	**0.005**
David (98k)	PGP	616	108	4.9	5.70	1.08
	MI	39K	261	**0.0**	0.24	0.08
	(MI)	542	8	3.9	4.96	0.45
	VSA	1348	349	0.8	3.80	0.16
	1%	1603	496	0.2	3.29	0.07
	10%	**375**	145	3.0	9.09	0.43
Arm. (345k)	PGP	1600	231	13.2	11.03	5.86
	MI	2184	74	0.8	[a] 3.32	0.19
	VSA	534	143	2.3	8.24	1.29
	1%	1402	466	**0.1**	2.46	0.07
	10%	**476**	258	0.3	4.48	0.12
	*	2170	773	**0.1**	[b] **2.35**	**0.08**

Table 2.2: *From the left: name of input mesh (number of triangles); method used to produce the control mesh (percentages refer to our method with different values of parameter τ of the M-fitmap; (MI) refers to the MI method with manual placement of cone singularities); number of patches $\square K$ in the final control mesh; total number of irregular vertices; portion of the input mesh that is not reached correctly by normal projection from the limit surface; RMS error (in 1000th of the diagonal of the bounding box) by using either the limit (L), or the displaced (D) subdivision surface. The line denoted with the (*) refers to a model built with our approach and with a number of patches similar to the MI output.*

	Dataset name and size					
	Gargo	Moai	Bunny	Fert.	David	Arm.
Times (secs.):	40K	53K	70K	80K	99K	98K
Fitmap	69	149	293	332	454	442
simplif./fit	486	592	780	722	1320	1360

Table 2.3: *Running times in seconds for computing Fitmaps, and for simplification and subdivision surface fitting.*

Chapter 3

Mesh Parametrization

Mesh parameterization is a geometry processing tool with numerous computer graphics applications, including texture and normal mapping, detail transfer, remeshing, morphing and surface fitting.

In general, a parametrization is a map from a surface to another. If one of the two surfaces is a discrete representation of a continuous surface (i.e. a mesh) then the problem of computing such map is referred as mesh parametrization. Parametrizations are usually computed between a subset of $\mathbb{R}^2$ (the parametrization domain) to a 2D manifold embedded in $\mathbb{R}^3$. This is not possible for general surfaces, so they are cutted until they become homeomorphic to a disk. For a detailed description of the possible applications and of the classical parametrization algorithms, we refer the interested reader to the course notes of the Siggraph 2008 course on mesh parametrization by Hormann et all [HPS08].

In this chapter, we present two novel algorithms that compute a global parametrization of a triangle mesh. A global parametrization is a single, continuous map from an arbitrary base domain to the mesh, without any assumption on the genus of the target mesh. Furthermore, we are interested in computing parametrizations that are aligned to local features of the mesh.

The first algorithm produces a parametric domain made of few coarse axis-aligned rectangular patches, which form an abstract base complex without T-junctions. The method is based on the topological simplification of the cross field in input, followed by global smoothing. We provide different constraints on the relative sizes of quadrangular domains of parametrization, which allow us to trade-off between quality of parametrization and flexibility of use. Our method does not need to cut the original mesh in regions topologically equivalent to disks, thus providing a powerful parametrization that is much simpler to use than other similar methods. The core of the algorithm is a method that is able to perform a purely topological simplification of the separatrix graph, that is a graph that describes uniquely the topology of a cross-field. The simplification is performed using a greedy algorithm that executes chains of simple, local operations to produce a new simpler graph and the corresponding cross-field. We provide comparisons with state-of-the-art methods and show examples of coarse quad re-meshing computed using our algorithm.

The second parametrization algorithm computes a cross-field that adhere with the symmetry already present in the object itself. Before the description of the algorithm, we review the literature on symmetry detection and we propose two novels algorithms that are specifically designed for the task at hand and specialized to compute bilateral symmetries. To compute the symmetry map in the extrinsic case, we build an high dimensional embedding for the entire model, where two vertices are "close" only if they symmetric. We extend techniques based on invariants that have been developed for 2D images to work on meshes, obtaining a very simple, completely automatic algorithm. For the intrinsic case, we provide a new, purely topological definition of intrinsic symmetry and a customized algorithm to compute it. The new definition is very general and it is also able to represent symmetric maps that are not isometries. Our proposed algorithm starts from a small number of landmarks specified by the user, and we show that the quality of our map is higher than existing automatic and semi-automatic methods. Once the symmetry map has been computed, we use it to produce symmetric fields, that are then processed to produce symmetric meshes and non-photorealistic renderings. Our method is based on the Mixed-Integer framework [BZK09], and it is also able to properly handle models that are not perfectly symmetric. We compare with recent parametrization algorithms and we show results on a variety of models, showing that by exploiting symmetry we are able to greatly improve the final quality of the results. All our results are shown on cross-fields, but the algorithm can be trivially extended to work on vector, line, and tensor fields defined on a surface.

Finally, we focus on a very specific type of parametrization: the computation of bijective mappings between two axis-aligned rectangular regions of $\mathbb{R}^2$. This task is important, since it models the deformation of a 2D image into another with a different aspect ratio. Controlling the deformation map, we are able to produce a rescaled image, with no distortion in the important parts, that looks natural even after notable changes in its aspect ratio. Clearly, the iterative application of such a technique might allow to resize a sequence of images and thus to resize a video clip. To obtain robustness and efficiency, we propose the space of axis-aligned deformations as the meaningful space for this task. Such deformations exclude local rotations that may lead to harmful visual distortions, and they can be parameterized in 1D. We show that standard warping energies for image retargeting can be minimized in the space of axis-aligned deformations while guaranteeing that bijectivity constraints are satisfied, leading to high-quality, smooth and robust retargeting results. Our method only requires solving a *small* quadratic program, which can be done within few milliseconds on the CPU without precomputation overhead. The image size and the saliency map can be changed in real time, giving immediate feedback to the user. We

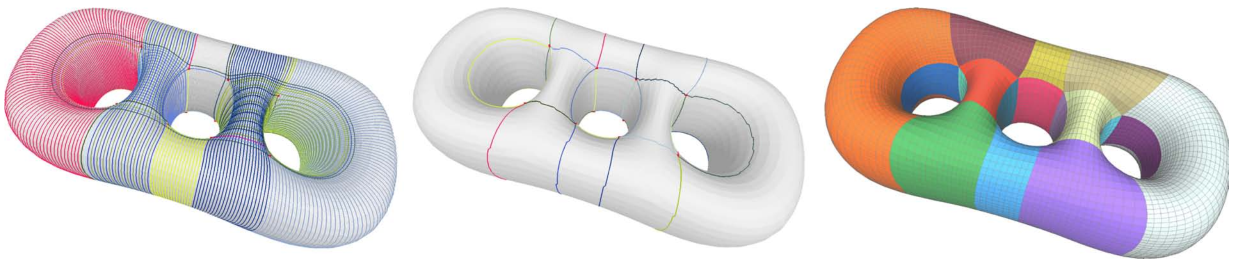


Figure 3.1: *(Left) An input mesh of quads induces a cross field with an entangled graph of separatrices; (center) the graph is disentangled with small distortion from the input field to obtain few parametrization domains; (right) parametrization is smoothed to make it conformal; an example of remeshing from the parametrization.*

present results on various input images, including the RetargetMe benchmark, and the summary of an user-study with 305 participants that confirms the quality of our results.

3.1 Related work

Mesh parametrization has been studied thoroughly in the literature [HPS08] . Here we consider just those works related to global parametrization and issues concerning domain simplicity.

Geometry Images [GGH02] map a whole triangular mesh onto a square parametric domain. Geometry images may be considered as the 2D flattening or development of an octahedron, which edges are mapped onto the borders of the parametric image. However, this mapping is possible just for surfaces of genus zero, their use is limited to the subclass of meshes that are homomorphic to a sphere. and mapping can be affected by a large amount of distortion. Distortion can be reduced with Multichart Geometry images [SWG$^+$03] where the domain is decomposed into a set of irregularly shaped flat patches. Because of their irregular borders, multi-chart geometry images have a complicate handling of discontinuities. Polycube-maps [THCM04] produce a seamless parametrization by projecting the geometry onto the faces of a polycube embedded in 3D space. Polycube mapping provides a very compact yet simple representation, , which is implicitly empty of discontinuities. As a consequence, and interpolating parametric positions involves just a simple 3D re-projection operation. Although techniques for the automatic generation of Polycube-maps have been proposed [LJFW08], generation of high quality embeddings still remains a manual task. but their quality still does not match hand designed Polycube-maps.

Methods based on mesh simplification [LSS$^+$98, KLS03, PTC10] automatically produce a parametrization domain composed of a set of equilateral triangles, This domain is obtained by a sequence of local simplification operations, keeping track of the mapping of the original shape onto its simplified version. producing very coarse domains. However they do not take into account alignment of parametrization to shape features. Moreover, they cannot be used to produce a quadrilateral remeshing. Other recent methods [DBG$^+$06, DSC09b] can produce simple parametric domains composed of a set of adjacent quads. The domains can be very simple, and they can be used to directly produce an isometric quadrilateral remeshing or for texture mapping, but However, these techniques do not allow control over the quad alignment. Spectral surface quadrangulation has been extended in [HZM$^+$08b] to take into account alignment to geometric features. However, methods based on Morse-Smale complexes force the size of quad patches to be uniformly determined by the underlying field. For this reason, according to the authors of [HZM$^+$08a], it is hard to align edges of a Morse-Smale complex with all feature lines if the distance between them is small.

Most recent approaches use a precomputed quadrilateral feature aligned quadrilateral mesh, either directly modeled by a human, or computed with methods such as Quadcover [KNP07], PGP [RLL$^+$06], Mixed Integer [BZK09], or Standing Wave [ZHLB10] as a base to gather a simplified parametric domain. Meshes generated automatically have a high quality, but they usually contain far too many quads to be used as parametrization domains for practical applications. Such meshes may in fact provide an input for our method.

Line fields and cross-fields have been used to compute mesh parametrization and surface quandrangulation. *Field-alignment techniques* adapt a parameterization to a shape by fitting the parametrization gradient to smoothed principal curvature directions, or more generally, to a smooth cross-field capturing surface features. The topological structure of the field (singularities and separating lines) indirectly determines how fine the domain mesh can be. A variety of methods were proposed for cross-field and more generally N-symmetry field (N-RoSy) construction. A number of methods rely on manually placed singularities and other user-specified information [RVLL08, CDS10, LJX$^+$10]. Manual placement of singularities is difficult even for shapes with moderate complexity, and only recently automatic methods able to produce high-quality, automatic results have been proposed [BZK09]. In this method, extending [RVLL08], fields are represented by per-triangle angles in fixed frames, and *matchings* on edges, indicating the additional $k\pi/N$ rotation of the field to transition between adjacent faces, which determine singularity placement. MI [BZK09] uses a greedy strategy to make the field as smooth as possible, with both angles and integers k as variables. Other methods include nonlinear optimization [HZ00, RLL$^+$06, RVAL09] also using angles to represent fields, and a linear optimization of a tensor field representation [PZ07]. As far as we know, no construction takes symmetry of the domain into account.

Motorcycle Graphs [EGKT08] partition a quadrilateral mesh into a set of quadrilateral patches by allowing T-junctions. A similar idea has been further exploited in [MPKZ10] to improve simplicity of the domain where T-junctions are reduced and optimized to produce a patch layout suitable for hi-order surface fitting. This method exhibits a great degree of adaptivity, i.e patches can vary noticeably their size over the surface to conform to details at different scale. However the presence of T-junctions complicates the structure of the parametric domain and may hinder its use for several applications.

Very recently, a method has been presented in [BLK11], to simplify the structure of a quadrilateral mesh while preserving its alignment. The optimization method is based on an extension of the polycord collapse operations [DSSC08] and consists of a greedy application of grid preserving operators directly on the quad mesh, directed at removing helical configurations. Similarly to what we propose here, this method has also the effect of simplifying the graph of separatrices induced by the quad mesh.

The quality of the feature-aligned quadrangulation depends on the quality of feature detection, a difficult problem for many classes of meshes. A number of techniques for defining and detecting feature lines were proposed: [OBS04, HPW05, WG09]. In our work, we use ridges and valleys computed from smoothed curvature values obtained using the robust estimation of [KSNS07].

Finally, it is worth mentioning that the topology of vector and tensor fields has been studied in the context of flow visualization techniques [DH94]. In order to avoid visualization cluttering, many approaches for simplifying the topology of fields have been proposed, like for example [TSH01, CMLZ08].

3.2 Simple Quad Domains for Field Aligned Mesh Parametrization

Finding a high-quality parameterization $f : D \to M$ for a given 3d polygonal mesh M is a prerequisite in a number of applications, such as quad-based semiregular remeshing, texture mapping, compression, fitting high order surfaces, physical simulations, tangent space geometry processing, and even tasks outside CG like physical modeling with metal sheets.

The definition of quality for f depends on the application, but usually encompasses criteria like injectivity, isometry (implying angle preservation and area preservation), smoothness (continuity of gradient vectors), and alignment of gradient vectors with geometric features of M. This problem has been addressed with a wide arsenal of tools [HPS08] and good automatic results are becoming increasingly common.

A recent trend is to first define a cross field $\mathcal{C}$ over M, and then to find f such that its gradient vectors match $\mathcal{C}$ as much as possible. Interestingly, each of the criteria above can be redefined in terms of desired properties of $\mathcal{C}$. Thus the task is shifted from the definition of a good parameterization to the definition of a good cross field $\mathcal{C}$ (for a given M). High quality parametrization can be obtained following this approach [RLL$^+$06, BZK09]. It is now apparent that the definition of a good cross field $\mathcal{C}$ implies, among other things, the good placing of a few irregular points (a.k.a. cone singularities). Irregular points tend to be needed, for example, in places where M exhibits high Gaussian curvature.

An important, *additional* criterion for the quality of f is what we term here the *simplicity* of domain D (see below for an informal definition). Simplicity determines how much a parametrization f will be effectively useful in most applications, just as much as the other criteria listed above. As we will show, a cross field $\mathcal{C}$ designed to satisfy all the above conventional criteria, but simplicity, will usually fail producing an acceptably simple domain. Still, it is often the case that a slightly modified cross field $\mathcal{C}'$ exists, which is able to generate a parametrization f with a dramatically simpler domain, while preserving to a large extent the other qualities of $\mathcal{C}$. This Section presents a way to obtain the cross field $\mathcal{C}'$, given $\mathcal{C}$.

Objective: Domain Simplicity For topologies of M other than the disk, the domain D must necessarily include discontinuities (a.k.a. cuts, or seams): two infinitesimally close points m_0 and m_1 of M lying of different sides of the cut are mapped by f^{-1} to arbitrarily distant positions d_0 and d_1 of D. The values d_0 and d_1 are often constrained to be reciprocally associated with a "transition function" associated to that cut.

Simplicity of domain D is a concept encapsulating: how many discontinuities are needed (the fewer, the simpler); how simple the discontinuity lines are in D (e.g., straight axis-aligned lines are simpler than jagged lines); and also how constrained and straightforward the transition functions are (the more constrained, the simpler). For example, a domain D consisting in a single flat unit square, with no seams, would exhibit the maximal possible domain simplicity (possible only for disk-like M). On the other extremum, a 2d packing of all single faces of M, each one laid separately on a plane, would give a domain so complex to the point of making parametrization useless for any practical purpose.

When, like in our case, D is defined as a collection of patches separated by cuts (also known as an atlas), domain simplicity means to have fewer and more regularly shaped patches, properly aligned and with simple transitions.

In our work, we consider the use of a domain D consisting of a collection of integer sized, axis-aligned rectangular patches $D_0, D_1, \cdots, D_n$ (see Sec. 3.2.3). Rectangles have a side-to-side adjacency relationship defined over them, mapping the *entire side* of a rectangle D_j with the *entire side* of another rectangle D_i, (i.e., there are no "T-junctions"), thus making the transition between them straightforward. For small n, this type of domains can be considered extremely simple, allowing, for example, straightforward applications to tasks like regular quad-remeshing at arbitrary resolutions, texture mapping, and so on.

It is easy to see that, For a parametrization f generated from a cross field $\mathcal{C}$, the simplest domain of this kind which can be adopted is determined by cutting M along the separatrices connecting the singularities of $\mathcal{C}$. Such separatrices define a graph G embedded on M, which may contain crossing nodes.

Even when $\mathcal{C}$ has relatively few singularities and separatrices connecting them, graph G can contain many crossing nodes and, thus, it can induce a very high number of patches.

It happens because separatrices can make long tours, possibly spiraling around the object many times, and crossing other separatrices (or even themselves) a very large number of times during their way. See the left side image of Fig. 3.29 for an example.

Main contributions The main contributions of this Section are:

1. A new general algorithm for simplification of the graph of separatrices G, which is generated by a cross field $\mathcal{C}$. A modified cross-field $\mathcal{C}'$ is trivially induced from the simplified graph of separatrices. Notwithstanding the dramatically simpler graph that it generates, $\mathcal{C}'$ is similar to $\mathcal{C}$ (see Sec 3.2.1).
2. A practical way to implement the above algorithm on a pure-quad semi-regular meshing Q, which is taken in input as a way to represent $\mathcal{C}$: edges of Q are aligned with directions of $\mathcal{C}$, and irregular points of Q correspond to singularities of $\mathcal{C}$. All the basic operations and the measurements of the algorithm can be easily stated in this scenario (see Sec 3.2.2).
3. A new kind of parametrization domain D, consisting of a collection of axis-aligned 2D rectangles with predefined side-to-side manifold connectivity. This type of domain arises naturally from a graph G induced by $\mathcal{C}$, and it exhibits an high degree of simplicity by construction. Global smoothing of an existing parametrization defined over D, and quad remeshing (see Sec. 3.2.3).

Contributions 1 and 3 could be adopted independently in different scenarios. In the following, we integrate all three contributions in a pipeline, ultimately aimed at parameterizing a semi-regular quad mesh over a simple domain.

Overview Our pipeline consists of the following phases:

1. Input: an initial semi-regular quad-based domain Q;
2. A graph of separatrices G is trivially extracted from Q (Fig. 3.1, left);
3. Graph simplification: a new "disentangled" graph of separatrices G', defined over Q, is constructed from G (Fig. 3.29, center);
4. A "simple" abstract domain D is constructed from G', as well as an initial parametriza-

tion $f : D \to Q$ (see Sec. 3.2.3.3);

5. Global smoothing: parameterization f is globally smoothed (see Sec. 3.2.3.3).

A new semi-regular remeshing (or embedded domain) Q' can be found by applying f to a regular sampling of D. Mesh Q' is naturally partitioned into few rectangular regions corresponding to the various faces of D. See the image on the right side of Fig. 3.1.

Final quad-mesh Q' looks at first similar to the original quad-mesh Q, in terms of density, alignment to geometric features, regularity and so on. In particular, Q' and Q always share the same number of irregular points. However, the merit of Q' over Q becomes evident if, for example, one tries to partition their quads into regular rectangular regions of $n \times m$ quads (see fig. 3.1). Thanks to a better alignment of irregular points, quads can be grouped in far fewer regions in Q' than in Q.

3.2.1 Cross-field topology simplification

In this Section, we describe an algorithm to "disentangle" the graph G of the separatrices induced by an input cross field $\mathcal{C}$, producing a simpler but still consistent graph G'. by reducing its number of crossing nodes, hence of arcs and faces, while maintaining the same set of singularities, each with its own index and corresponding number of incident separatrices.

This implicitly produces a modified cross field $\mathcal{C}'$ that has G' as separatrix graph. We strive to keep the differences between $\mathcal{C}'$ and $\mathcal{C}$ small, even if G' is dramatically simpler than G. Specifically, $\mathcal{C}'$ and $\mathcal{C}$ share identical irregular points (of the same order).

In terms of parametrization, as discussed, this means that we trade some alignment for a simpler domain topology. The algorithm aims at maximizing graph reduction, while minimizing deviation from the original cross field. A unique parameter sets the relative importance of topology simplification and faithfulness to the cross field.

3.2.1.1 Preliminaries

Let $\mathcal{C}$ be a smooth cross field defined on a 2-manifold M. Field $\mathcal{C}$ associates to a point $p \in M$ four orthogonal directions on the tangent plane $T(p)$ of M at p. One unary vector on $T(p)$ is sufficient to describe $\mathcal{C}$ at p, the other three directions being obtained by rotations of such vector by multiples of $\frac{\pi}{2}$. For all general definitions and properties about cross fields, we refer to [RVLL08]. We assume that $\mathcal{C}$ has only a finite set S of isolated *singularities.* For the sake of simplicity, we assume that such singularities may just have indices $+\frac{1}{4}$, $-\frac{1}{4}$, $+\frac{1}{2}$, or $-\frac{1}{2}$, corresponding to the most common cases. Our method can be easily extended to work also with singularities of higher order, though. Each point of M, which is not a singularity of S, will be said to be *regular.*

A *streamline* of $\mathcal{C}$ is a line on M that is tangent/orthogonal to the directions defined by $\mathcal{C}$ at each point.

A streamline with endpoints at singularities is called a *separatrix.* Field $\mathcal{C}$ is regular within each patch, i.e., patches are actually "gridded" by the streamlines of $\mathcal{C}$.

For a singularity of index $+\frac{1}{2}$, $+\frac{1}{4}$, $-\frac{1}{4}$, or $-\frac{1}{2}$, there are exactly 2, 3, 5, and 6 incident separatrices, respectively. The network of separatrices is a graph embedded on M, describing the topology of $\mathcal{C}$. Separatrices cross at a finite set X of regular points of M, called the *crossing nodes.* Only two (possibly not distinct) separatrices can cross at each node. Crossing nodes subdivide separatrices into *arcs* of a set E. The planar graph $G = (V, E)$, where $V = S \cup X$ subdivides M into quadrangular patches.

For each singularity $v \in S$, let $s_1, \ldots, s_k$ be the separatrices incident at v, and let $\mathbf{t}_{v,s_i}$ be the unit tangent vector of separatrix s_i at v, pointing outwards v in the direction of s. Each vector $\mathbf{t}_{v,s_i}$ is called a *port* of v.

By design, the simplified graph $G' = (S \cup X', E')$ will have the same set of irregular points, ports, and separatrices of the original graph $G = (S \cup X, E)$. The objective of the simplification is the reduction of the number of crossings $|X'|$ (and thus of edges $|E'|$).

3.2.1.2 Graph energy

Given the graph G of separatrices, we aim at obtaining another graph $G' = (V', E')$ such that $V' = S \cup X'$, with $|X'| < |X|$ and having the same features of G: nodes of S maintain the same ports; nodes of X' are regular; patches induced on M are quadrangular; separatrices are smooth lines defined by chains of arcs.

We introduce a measure of *"drift"* and *"extension"* for a line l on M. Drift $\delta(l)$ measures the misalignment of l with respect to $\mathcal{C}$. Extension $\eta(l)$ measures the length covered of l following $\mathcal{C}$.

Assume l is parametrized by arc-length: $l : (0; \lambda(l)) \to M$, where $\lambda(l)$ is its total length. The $\delta(l)$ and $\eta(l)$ are defined as

$$\delta(l) = \textstyle\int_l |\sin(\theta(t))| dt, \quad \eta(l) = \int_l \cos(\theta(t)) dt,$$

where $\theta(t)$ is the angle between $\nabla l(t)$ and the closest unit vector (one out of four) at $\mathcal{C}(l(t))$.

The energy of a graph is the sum of the energies associated to all its separatrices. The energy of a separatrix s is given by the weighted sum of its extension and its drift:

$$k\delta(s) + \eta(s)$$

with a parameter k setting the relative importance of the two terms. The first term forces the graph to follow the original field $\mathcal{C}$. The second term is minimized when shorter separatrices are used, which implies a simpler graph with less crossings. We could penalize directly the (discrete) number of crossings, instead of the separatrix estensions. However, the two terms of energy as defined above use the same unit of measure (metric length), thus making k independent of rescaling. Parameter k is the *only* one in our framework, and balances the need to preserve the initial field $\mathcal{C}$ with the amount of simplification.

Empirically, we found a good value of k to be 5, which was used in all our experiments.

3.2.1.3 Graph reduction algorithm

In the initial graph G, which follows $\mathcal{C}$ exactly, there is no drift, and the total energy is given by the sum of the extensions of the separatrices, which for each separatrix amounts to its (geodesic) length.

Our algorithm follows a greedy strategy, trying to reduce the energy of the graph by substituting some of its separatrices with different lines, while maintaining its topological structure consistent.

Hard constraints may be set, e.g., by "freezing" those lines lying on sharp creases. On the other hand, soft constraints given by the drift component of energy are aimed at maintaining the output close enough to cross field $\mathcal{C}$, wherever small modifications are allowed, like in smooth areas or at smooth creases.

The algorithm performs a sequence of simplification *cycles*, each consisting in a sequence of *moves*: an *opening* move; a sequence of (zero or more) *continuation* moves; and a *closing* move.

Each move is composed of at most two sub-operations: an opening move consists of a deletion sub-operation; a continuation move consists of a deletion sub-operation, immediately followed by a creation sub-operation; a closing move consists of a creation sub-operation.

The **deletion sub-operation** is simply the removal of a separatrix. One *open port* is created at each end, i.e., two ports remain with no associated separatrix. A graph is consistent only if it has no open ports.

The **creation sub-operation** connects two open ports with a new separatrix, thus closing them. The new separatrix is plotted over the surface: it starts from from an open port, it may cross other separatrices, and it ends at the targeted open port (never crossing any other singularity). At both ends, the new separatrix matches the tangent directions of the open ports it connects to.

Some constraints must be fulfilled when the new separatrix l is traced. We impose the

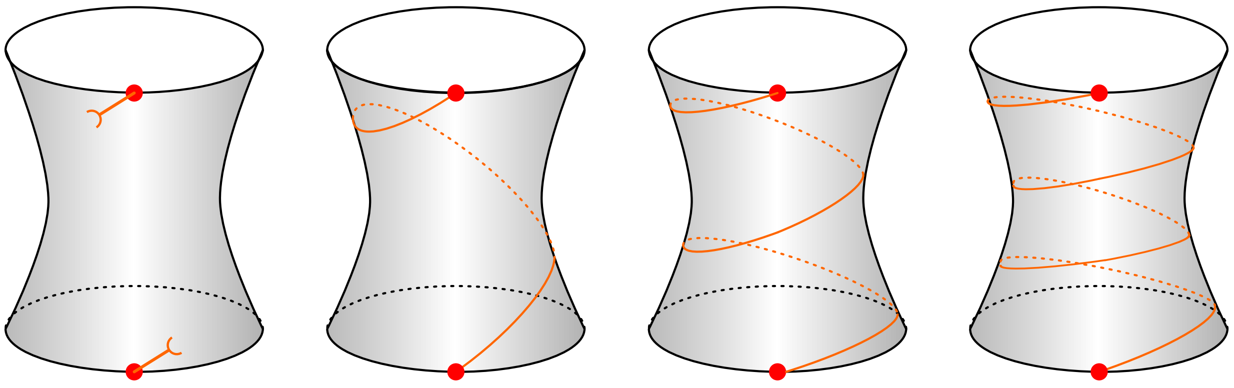


Figure 3.2: *Two open ports (left) can be connected with several different separatrices traveling in the same corridor: the separatrix minimizing the energy is selected (depending on the field, not necessarily the shortest one).*

drift to be monotonic, i.e., such that $\theta(t)$ is either positive (right drift) or negative (left drift) along the whole line. We also impose $|\theta(t)| < \pi/4$, posing a limit on the accepted discrepancy. This means that the cross field direction (one out of four) more closely matching ∇l can only change with continuity over l. In other words, l is not allowed to switch the direction of the field it is aligned to.

There can be several ways at which the new separatrix can be drawn, even very different from each other: the one with the least associated energy is selected (see an example in Fig. 3.2).

3.2.1.4 Selecting moves

A cycle starts with a consistent graph without open ports. The opening move creates two open ports; a continuation move creates two more open ports (separatrix deletion), and immediately closes other two (separatrix creation), so that a total of two ports remain open during the entire cycle; the closing move closes the two open ports, bringing the graph back to a consistent state.

Note that a deletion sub-operation necessarily decreases the total energy of the graph (removing the contribution of the deleted separatrix), whereas a creation sub-operation necessarily increases it (adding the contribution of the new separatrix). Therefore an opening move always decreases energy, and a closing move always increases energy (but it is necessary to close the cycle). A continuation move changes the energy of the difference between the deleted and the created separatrix energies, which can be positive or negative.

We employ a greedy strategy to pick moves. When the cycle is started, there is one possible

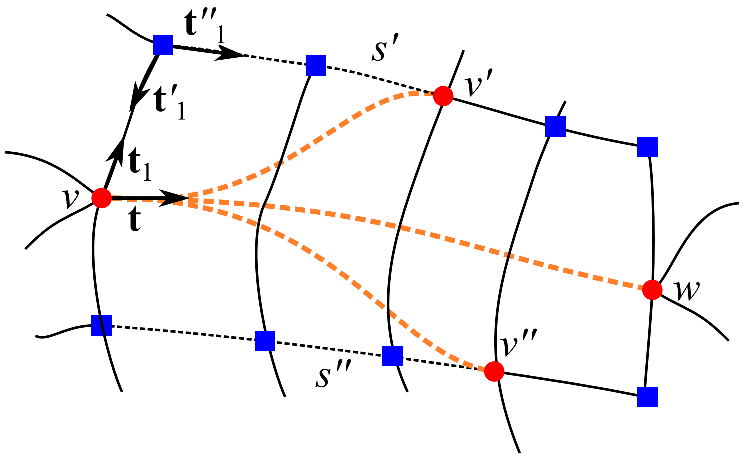


Figure 3.3: *The corridor of an open port* $\mathbf{t}$ *at* v*: singular nodes* v' *and* v'' *on the walls can be connected to* $\mathbf{t}$ *with a continuation move, which would respectively first delete separatrix* s' *or* s''*, then connect* $\mathbf{t}$ *with* v' *or* v''*. In case the other open port at* w *is reached along the corridor, then a closing move is possible, which connects* $\mathbf{t}$ *with* w*.*

opening move for each separatrix (which deletes it); we simply pick the one that deletes the most energetic separatrix.

When two open ports are present, all possible continuations and/or closing moves are found, as explained in the following subsection. The effect of each move on energy is evaluated (see sec. 3.2.1.5). A potential move is considered valid only if, after performing it, the total change of energy of the current cycle remains strictly negative. This guarantees that overall energy is decreased by the cycle.

If a valid closing move is available, it is always preferred over any continuation moves. This move closes the cycle, and a new cycle can be started by selecting the opening move again. If no valid closing move is available, we choose the move, among valid continuation moves, that decreases the energy the most (which sometimes can be an energy increasing move). If no valid move is available, a backtracking mechanism is triggered (see Sec. 3.2.1.6).

3.2.1.5 Enumerating continuation and closing moves

Let $\mathbf{t}$ be an open port (see Fig. 3.3). In order to maintain a consistent structure of graph G', any new separatix starting at $\mathbf{t}$ must necessarily be contained in a *corridor* bounded by two chains of edges, called the *walls*. The left wall of the corridor is defined as follows (refer to Fig. 3.3). Let $\mathbf{t}_l$ be the port next to $\mathbf{t}$ by rotating counterclockwise about its node, let $\mathbf{t}'_l$ be the port opposite to $\mathbf{t}_l$ on the same arc, and let $\mathbf{t}''_l$ be the port next to $\mathbf{t}'_l$ by rotating counterclockwise about its node. The left wall starts at $\mathbf{t}''_l$; it continues at each next node by skipping one port in counterclockwise order, and taking the next port; and it

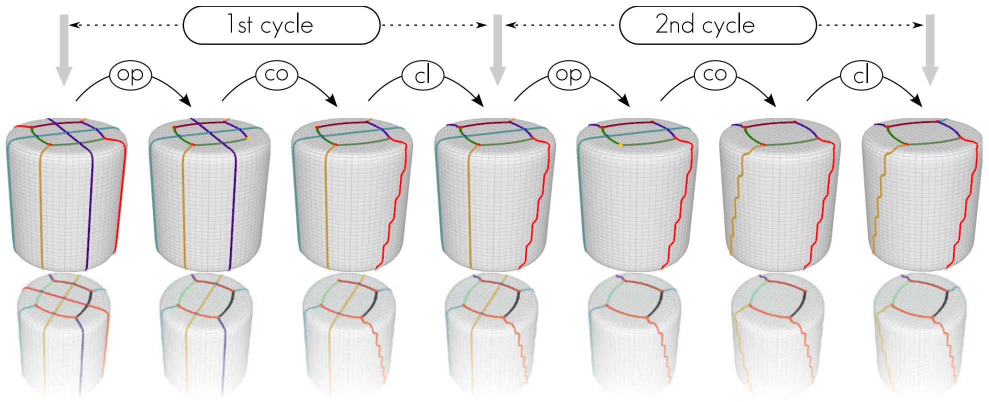


Figure 3.4: *A simple example of the graph simplification algorithm in action. The surface is shown reflected in the floor to make its lower face visible. The algorithm starts from a cross-field with eight* 1/4 *valencies, for a total of 24 ports, connected by 12 separatrices, which cross 10 times, dividing the surface in 16 rectangular patches. Valid configurations are pointed by gray arrows. The algorithm performs two cycles: each cycle starts with a opening move ("op"), a single (in this case) continuation move ("co"), and finishes with a closing move ("cl"). In this specific case, the algorithm removes all the crossings, producing a graph where only 6 rectangular patches are needed to cover the object. A third cycle is then attempted (not shown), in which a total of 54 moves are performed and eventually rolled back. The algorithm then returns the rightmost shown configuration. See attached movies (and result images) for more complex examples.*

stops when reaching either the node of $\mathbf{t}$ or the node of $\mathbf{t}''_{\mathbf{l}}$. The right wall of the corridor is defined analogously. Ports skipped at intermediate nodes along the walls connect opposite walls through transversal arcs.

If the other open port $\mathbf{p}$ lies along one of these arcs, and it is directed towards the beginning of the corridor, then a closing move which connects $\mathbf{t}$ to $\mathbf{p}$ is reported as possible. In this case, the end wall of the corridor is made of the two transverse arcs emanating from $\mathbf{p}$. Otherwise, the corridor is circular and it ends at one of the two transverse arcs emanating from $\mathbf{t}$ itself. In the latter case, the corridor bounds the search space for the new separatrix in a continuation move. For each port $\mathbf{p}$ stemming from singular nodes that lie on walls and point towards the beginning of the corridor, there is a possible continuation moves that connects $\mathbf{t}$ to $\mathbf{p}$. This move consists in the deletion of the separatrix currently starting from $\mathbf{p}$, followed by the creation of a separatrix connecting port $\mathbf{t}$ to $\mathbf{p}$.

3.2.1.6 Exploring the space of solutions

If no valid move is available, i.e. when even the best one would result in a negative overall score for the current cycle, then the last performed move is rolled back. The cycle continues by picking the *next* valid move at that configuration, if it is available. Else, another rollback

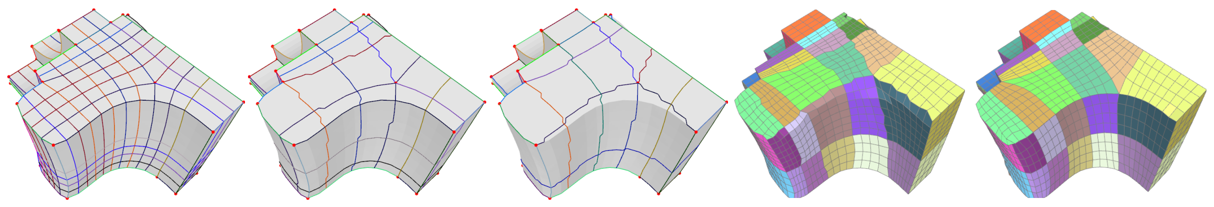


Figure 3.5: *Preservation of creases. From the left: initial graph; graph simplified by hard constraints on creases; graph can be simplified more without constraints, but some creases are lost; related domains just after simplification; final domain of parametrization: creases lost during unconstrained simplification can be recovered through snapping during the smoothing phase.*

is performed, and so on.

Iterating this simple strategy is equivalent to a depth-first visit of the tree of possible states reachable by valid moves: the root is the consistent graph at the beginning of the cycle, intermediate nodes are inconsistent graphs, and links are valid moves.

This search for a closing move can be either successful or not. If a valid closing move is found, then it is performed and the current cycle is over. The reached valid graph is necessarily different and it has a strictly minor energy than before. After that, a new cycle can be started.

Conversely, if the search fails (this happens when there are no opening moves left at the root of the tree) the algorithm is over, and the current graph (which is consistent) is returned as the final result.

The non-negative energy reduction constraint serves as a pruning of the tree during the search. In theory, the number of reachable nodes of the tree is still gigantic, but the number of possible states is not, so a dynamic programming approach makes an exhaustive search feasible. A graph with n ports can be connect them (pairwise) in only $\frac{n!}{(n/2)!2^{n/2}}$ different ways, the vast majority of which is not reachable by allowable moves. We hash each such configuration, and reject any move that would produce a configuration that has already been seen, during the simplification process, with a lower or equal associated energy. The algorithm is greedy and it does not give any guarantee of always returning the best configuration, but in practice it dramatically improves any input configuration.

3.2.1.7 Crease preservation

For surfaces with crease angles, like mechanical objects, it can be important that separatrices pass though creases.

In this algorithm, it is easy to prevent losing separatrices which are aligned to creases, simply by tagging them and disallowing any opening or continuation move which would remove them (see for example Fig. 3.5). This is a very safe option, but reduces the degree of freedom of the algorithm, and produces a less simplified graph.

Another viable strategy which is sometime available is to let crease separatrices be first deleted and replaced in this phase, and let other close separatrices snap into their place in the smoothing phase, which is described in Sec. 3.2.3.

3.2.2 Implementation on semi-regular quad meshes

We show a practical implementation of the graph reduction algorithm which takes as input a semi-regular quad mesh M, which implicitly provides a discretized cross field $\mathcal{C}$. Edges of M represent directions of $\mathcal{C}$. Irregular vertices of M represent singularities of $\mathcal{C}$, and edges stemming from them are their ports. We further assume that, as it commonly results from remeshing algorithms, all edges of M have approximately the same length. We approximate this edge length as the unit length. In this setting, the algorithm described above can be implemented in a simple and efficient manner.

Each separatrix s is composed of a sequence of edges of M. The initial graph of separatrices G can be extracted trivially from M by tracing all chains of edges stemming from irregular vertices. Any drifting separatrix generated during graph reduction is stored as a jagged sequence of edges (see Fig. 3.6).

This strategy can be seen as problematic, since the separatrices that are drawn are not smooth lines. However, exploiting the merits of the global parametrization which will result from the simplified graph, it will be easy to remove these local defect in a subsequent phase (see Sec. 3.2.3).

Its length and drift can be easily computed by a simple count of how many edges agree/disagree with the reference direction of the starting port.

A corridor is traversed by following the chain of edges stemming from the open port $\mathbf{t}$. Whenever a transverse separatrix s of G' is met along the corridor, then the two walls of the corridor at the intersections with s are tested for potential candidates to connect $\mathbf{t}$; the possible presence of the other open port along the segment of s within the corridor is also tested. If a wall is hit during traversal of the corridor, the direction of traversal is just drifted opposite to such wall.

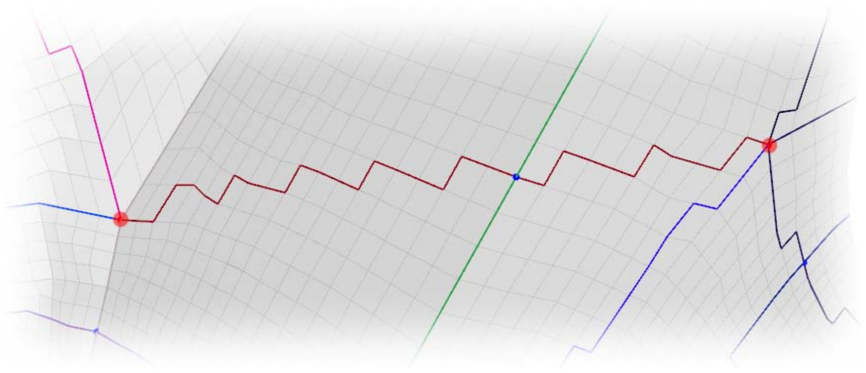


Figure 3.6: *On a quad mesh, a drifting separatrix s' of G' consists of a chain of either field-aligned or trasverse edges of M. The one depicted here has a value of extension of 24 units, and a value of drift of 8 units.*

3.2.3 Parametrizations over abstract quad-mesh domain

A graph of separatrices G over mesh Q partitions the surface of Q into rectangular patches. Each patch can be easily parameterized over a flat, axis aligned rectangle D_i. Let D be the collection of all $D_1, D_2, \cdots D_n$, let $f : D \to Q$ be the global parameterization, and $\phi = f^{-1}$ be its inverse.

As we will discuss, D is very appealing in terms of simplicity; moreover, f is aligned by construction to the cross field $\mathcal{C}'$ associated with G. These two facts constitute our main motivation for the graph-simplification algorithm of the previous section.

This Section discusses the properties of a parametrization domain of this kind, and shows the operations that can be performed over it, including global smoothing. The operations discussed in this section have the following aim: remove jagged artifacts introduced by the algorithm in 3.2.2; compute a smooth cross field over M (whereas tracing separatrices only defines it along these lines); optimize the placement of singularities and cross points (to comply with the new graph); optionally, recover of lost feature lines.

Even if Q is a quad mesh, in this section it is easier to consider the triangle mesh M obtained from Q by means of diagonal splits. This is no limitation however, because the connectivity of Q is unaffected: edges of M which are quad diagonals of Q can be tagged as such and the quad connectivity of Q can be recovered at any time.

3.2.3.1 Construction of initial parameterization

As commonplace, a parametrization over M is defined by discretely sampling its inverse, e.g., by assigning an explicit parametric position $p_i = \phi(v_i)$, $p_i \in D$ to each vertex v_i of M. This per-vertex assignment can be propagated, by linear interpolation, over the entire

M, because D allows for interpolations, even for triangles whose vertices are scattered in different patches. The only necessary assumption is that, for a triangle t in M, $\phi(t)$ is not so large to span a set of several different domains of D not sharing a vertex (see Sec. 3.2.3.2).

Vertices of M are separated in disjoint groups surrounded by four arcs of the graph G. Within each group, all vertices are assigned to a different rectangular domain D_i, and each arc surrounding the group is assigned to a corresponding edge of D_i.

A vertex v_i is considered a "border" vertex if it shares an edge with any vertex v_j, such that p_i and p_j belong to different domains. When this happens, the edge separating p_i and p_j is identified, as well as a parametric position within that edge. Positions p_i and p_j are determined inside their respective domain D_k and D_h, at the corresponding position of the appropriate border of D_k and D_h, respectively.

When parametric positions of all border vertices are set, they are fixed and the positions of non-border vertices are found by applying a single-patch energy-minimization parametrization technique inside every patch. We use [JSW05], but many other methods could be used in its place. This method tends to produce conformal mapping but, due to the shape and size similarities between the D_i and $\phi(D_i)$, it also delivers a certain degree of isometry.

The dimensions of D_i in parametric space are determined using a separate procedure, described in Sec. 3.2.3.5.

3.2.3.2 Transition functions and interpolation domains

A transition function is associated to each edge j of each domain D_i, consisting of a rotation by a multiple of $\pi/2$ and a translation. We call the domain D abstract in the sense that, even if it is endowed with a manifold connectivity (including face-face connectivity and shared vertices), it is not embedded in Euclidean space. Similarly to what is done in [PTC10] for triangular domains, interpolation-domains can be easily defined over D, and they can be employed to allow interpolation between points in D lying in different patches.

An interpolation domain E_i is a 2D region where a set of k contiguous patches $D_{a_1}, D_{a_2}, \cdots D_{a_k}$ are mapped by into one bigger patch, and it is associated with an invertible function $g_{E_i} : \cup_{j\in(1..k)} D_{a_j} \to E_a$. Functions g are expressed in closed forms and are easy to evaluate in both ways, mapping each domain D_{a_i} into E_a.

"Edge" interpolation domains E_i^e unify two adjacent patches of D over the shared border; *"Vertex"* interpolation domains E_i^v, unify n patches around a vertex (see Fig. 3.8 left). For domains around edges and regular vertices, the associated function g_i is simply an appropriate rigid roto-translation. For irregular vertices of valency k, g includes an exponential map (with exponent $k/4$), which is conformal (see Fig. 3.8 right). For completeness, we

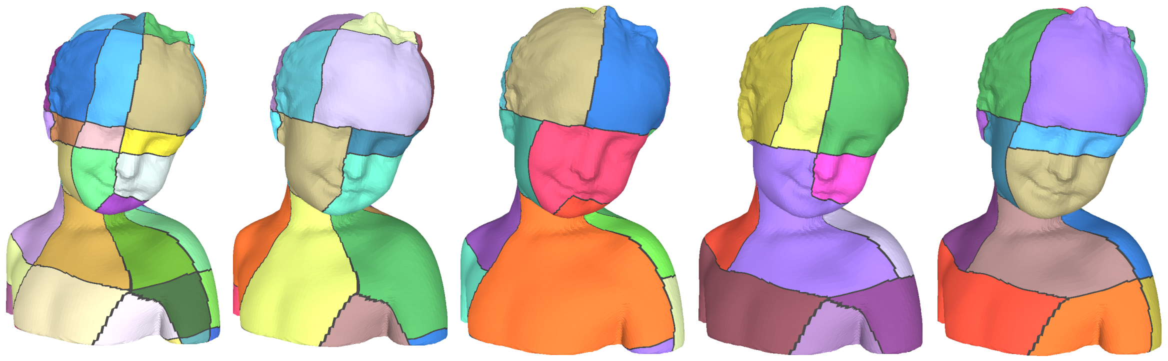


Figure 3.7: *Leftmost image: mesh M is partitioned into $f(D_1), (D_2), \cdots, (D_n)$ (each vertex v is coded according to domain $\phi(v)$, but triangles connecting fixed vertices are darkened). Other images: the same is repeated using four different set of domains $E_1, E_2 \cdots, (E_m)$, partitioning of D. Note that each vertex of M is not a fixed vertices in at least one of the four partitions.*

also define a trivial *"Face"* interpolation domains E_i^f which is identical to a single patch D_i; the associated function g_i is the identity. There is exactly one Edge domain for each shared edge of D, one Face domain for each patch of D, and one Vertex domain for each shared vertex of D.

An interpolation domain allows to interpolate, in parameter space, between a pair of points $p_0, p_1 \in D$, as long as they belong to two domains D_0 and D_1 which share at least one vertex. The interpolation can be computed as $g_a^{-1}(I(g_a(p_0), g_a(p_1)))$, where I is the common interpolation operator (see Fig. 3.8 top).

3.2.3.3 Global smoothing of the parametrization

Another natural use of interpolation domains is the global smoothing of a given parameterization. This is done in a sequence of passes. The idea is that, at each pass, some vertices are kept fixed and their positions will not be optimized, but these will be necessarily optimized in subsequent passes.

At each pass, we adopt a different set $S = \{E_0, E_1..E_i\}$ of interpolation domains such that each patch D_i belongs to exactly one interpolation domain. A simple heuristic is adopted to determine set S. Starting from an empty set S, vertex domains E_0^V are inserted into S, only if they are composed by patches not already included in any $E_k \in S$. The process is repeated for Edge domains, and finally the isolated domains D_i still not included in an $E_k \in S$ are inserted as Face domains. Before each pass, we keep a count, for each edge and each vertex of D, of the number of consecutive passes that element lies on the boundary of the interpolation domain embedding it. Such count is used as a priority to select the vertex and face domains.

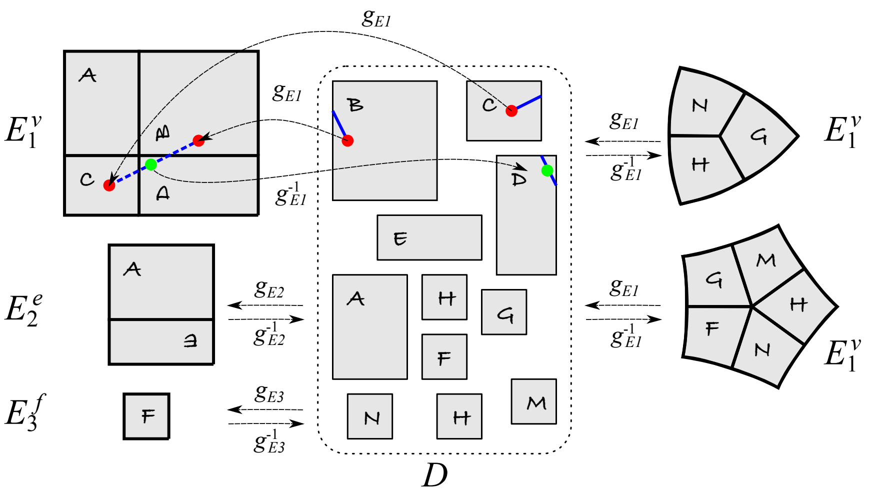


Figure 3.8: *Middle: a domains D composed of patches $D_1 \cdots D_n$, marked by calligraphic uppercase letters. Left, from top: an example of a Vertex, Edge and Face interpolation domain, and associated functions g. Right, from top: other examples of Vertex domain for vertices of D with valency 3 and 5 respectively. Top right: an interpolation domain used to define the interpolation between the two red dots $\in D$. The result of the interpolation is the green dot in D.*

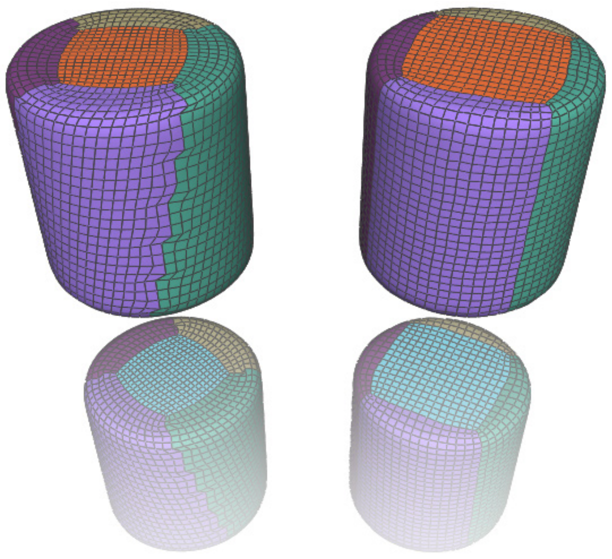


Figure 3.9: *Two semi-regular meshes obtained by resampling the parametric domain Left: parametric domain resulting from the simplification of Fig. 3.4: patches are separated by visibly jagged lines. Right, after multiple session of global smoothing (see Sec. 3.2.3.3), the jagged lines are gone, and irregular points moved to aligned, optimized positions.*

For a vertex v_i with associated parametric position p_i inside D_j, a position p'_i into one E_k is found by $p'_i = g_k(p_i)$. Again, vertices of M connected by edges of M to vertices in different interpolation domain are fixed, and the other are smoothed locally inside the respective interpolation domain. After the smoothing, vertices are remapped into D by the functions g_{E_i} of respective domain, and a new pass is started.

A formal demonstration is omitted for space reasons. A trace is that, at each pass, functions g and their inverse preserve conformal energy, and the smoothing operation monotonically decreases it. Results of this smoothing process is depicted in Fig. 3.10 and 3.9.

3.2.3.4 Recovering lost feature lines

As mentioned in Section 3.2.1, we can choose *not* to preserve creases in Q (and therefore in M) during the graph simplification. When this is the case, we can demand to the smoothing phase the task of realigning patch borders to the geometric feature of M.

Specifically, this can be done when the parametrization is being optimized over a interpolation domain. Edges of M tagged as feature edges can be snapped, and then constrained to lie on a 2d straight internal line $l = g(e)$, e being the set of points in D which lie on the border of the appropriate patch D_i.

3.2.3.5 Isometry

Each 2d rectangle in the domain D is assigned to an extension in each dimension. The dimensions can be chosen to maximize the isometry of mapping f, by solving a system with just two variables. For better results, this process is interleaved to the smoothing passes (Sec. 3.2.3.3), because, during the smoothing, the portion of M mapped inside each patch of D can vary.

3.2.4 Results

In this section, we show a gallery of results obtained on several datasets commonly used as benchmarks. Our input fields come from quad meshes either kindly provided by the authors of [BZK09] (drill-hole, fandisk, fertility, joint, rockerarm), or produced with an independent implementation of the same method (bimba, bunny, cube-blob, fertility-sym, holes3, kitten) and cover a spectrum of mechanical and natural objects, simple and complicated shapes. Table 3.1 provides statistics on the datasets and results. For each input mesh we provide: its number of facets; the number of irregular and crossing nodes of the graph of separatrices and the number of domains induced by such a graph; the number of crossing nodes and domains for the simplified graph.

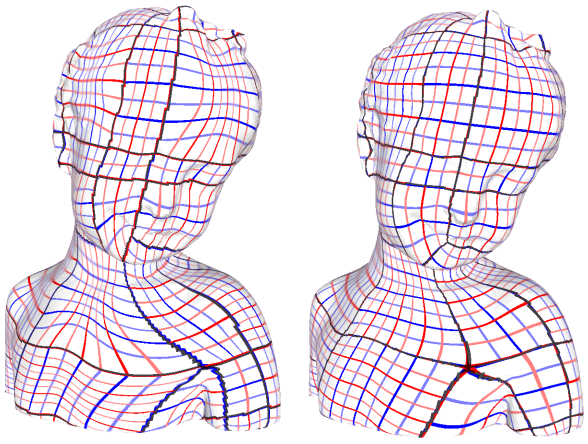


Figure 3.10: *Smoothing: an initial parametrization is computed by assigning vertices of the input mesh to domains induced from the simplified graph (left); parametrization is smoothed to make it conforming (right).*

Models	Facets	Singular nodes	*Original Graph* Crossing nodes	Domain patches	*Simplified Graph* Crossing nodes	Domain patches
holes3 (Fig.1)	36487	16	7729	7749	0	20
cubeblob (Fig.14 top left)	9146	56	5546	5600	24	78
fertility (Fig.13 left)	3357	48	2217	2271	199	253
fertility symm. (Fig.13 right)	5785	46	4546	4598	79	131
kitten (Fig.14 mid left)	31198	30	31168	31198	49	79
bimba (Fig.7,11)	31618	26	31594	31618	58	82
bunny (Fig.14 bottom left)	35862	43	33938	33979	83	124
drill_hole (Fig.14 middle)	3077	26	1344	1368	58	82
joint (Fig.12)	8804	23	473	498	87	112
fandisk (Fig.5)	764	30	380	408	60	88
rockerarm (Fig.14 top right)	9413	36	4488	4524	62	98

Table 3.1: *Statistics on datasets and graph simplification.*

In Fig. 3.14 we show results of the simplification phase from two datasets also used in [BLK11], for comparison. In the drill-hole dataset, exactly the same input mesh has been used, and our result provides a simpler domain. In the rockerarm dataset, a finer mesh has been used, which provides a more entangled input field. In spite of that, our result also provides a simpler domain. In both cases, alignment is preserved. Note that we just depict the graph of separatrices: some lines are jagged because they are traced in a discrete way on the underlying mesh. Jags are eliminated with the subsequent phase of smoothing, as shown in results from the same datasets in Fig. 3.13.

In Fig. 3.10 we show the effect of smoothing on the bimba dataset. Fig. 3.13 shows results from the two phases of the algorithm on several other datasets. In Fig. 3.11 we show how sharp creases can be preserved by freezing them during simplification. Note that also sharp creases that were *not* captured by separatrices, like the border of the big hole in the left image, can be recovered during the smoothing phase thanks to the snapping. Finally, in Fig. 3.12 we show how results can be improved by providing a better placement

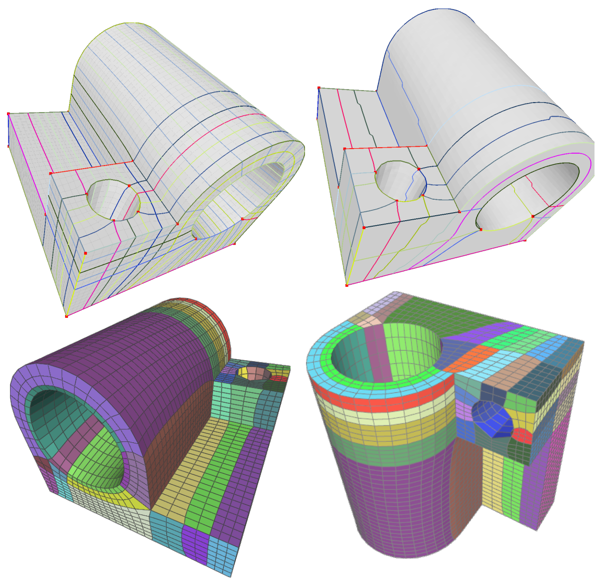


Figure 3.11: *Sharp features are preserved either by hard constraints during simplification, or by snapping during smoothing.*

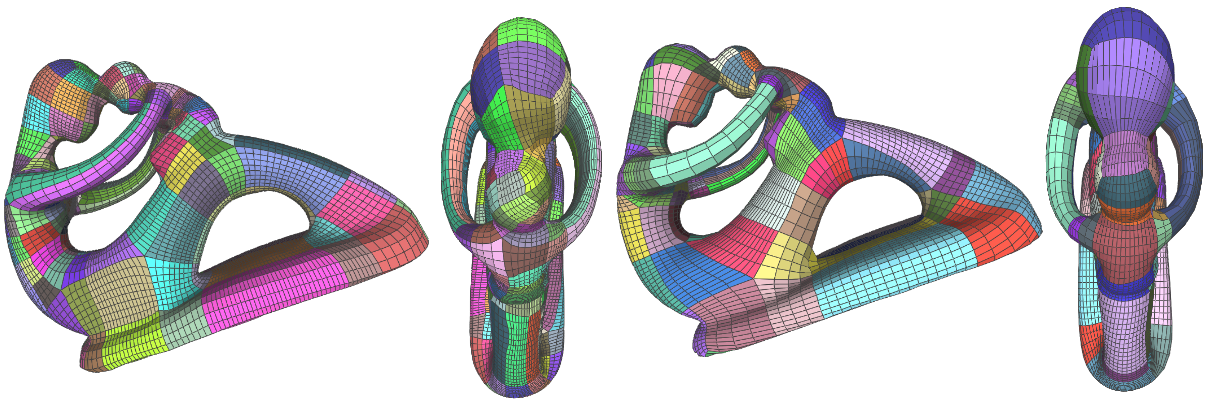


Figure 3.12: *A better layout of singularities helps obtaining a simpler domain and a better remeshing.*

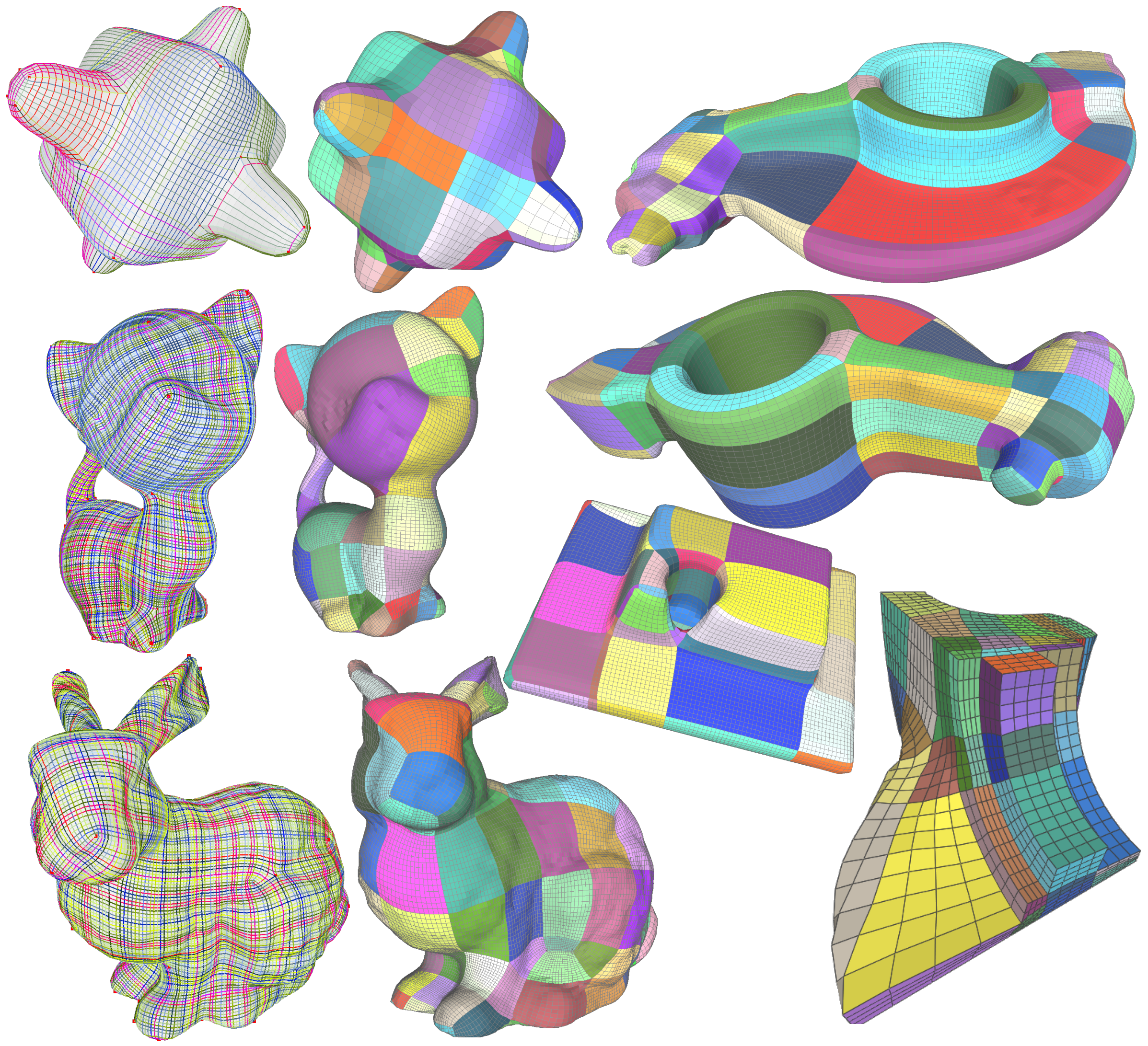


Figure 3.13: *Results of our algorithm on several datasets. For each dataset, the input graph of separatrices and a remeshing colored with faces of the resulting base domain are shown (input graphs are depicted in previous figures for some datasets).*

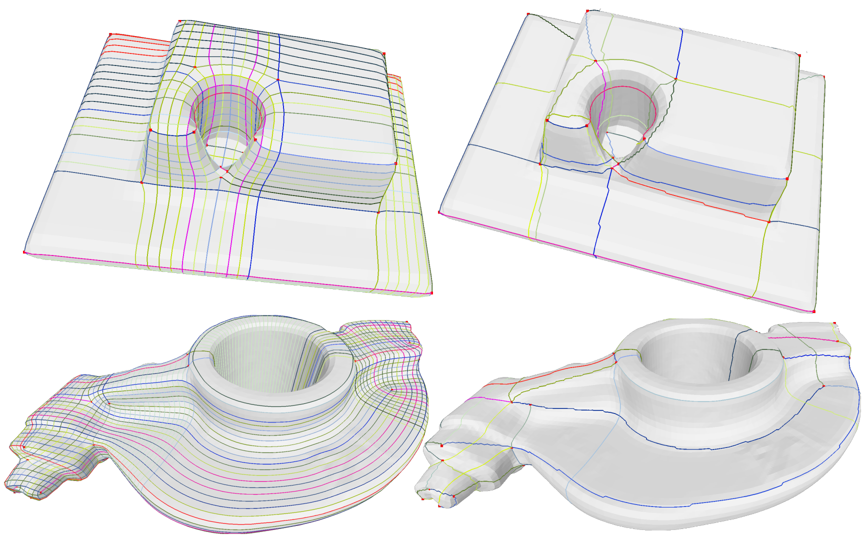


Figure 3.14: *Graph simplification: input the graph of separatrices may be more or less entangled, in all cases our graph reduction algorithm manages to dramatically reduce the complexity of the domain while preserving alignment to the input field.*

of singularities of the cross field. Results on the left side are taken from an input mesh obtained with the original Mixed Integer algorithm (dataset fertMI); results on the left side are obtained from another dataset which has been computed by placing singularities of the cross field more symmetrically with respect to the shape (dataset fert-sym). In both cases, the initial graphs are extremely entangled (most edges belong to separatrices). It is evident, especially from the top view, how a better placement of singularities allow us to obtain a smaller number of simpler domains, and a better remeshing (see also statistics in Table 3.1).

3.3 Symmetric N-symmetry Fields

Many geometry processing applications require the construction of N-symmetry fields on surfaces, i.e., fields that associate to every point a set of N unary vectors forming equal angles between radially consecutive directions. For example, a direction (2-symmetry) field can be used to guide texture placement or texture synthesis, as well as for anisotropic smoothing or text placement; a cross (4-symmetry, 4-Rosy or 4-tensor field) field is useful for constructing quadrangulations and for anisotropic remeshing, as well as for supporting non-photorealistic rendering.

In most cases, it is desirable for the fields to respect the symmetries of surfaces: meshes respecting surface symmetries are visually preferable and reduce deformation/animation artifacts related to asymmetric meshing; patterns and small-scale geometry (for example, fish scales or fur) are often required to follow the shape's symmetries. Figure 3.15 shows field-aligned parametrization using cross-field constructed with our method, side by side with the Mixed-Integer quadrangulation (MI) field of Bommes *et. al.*[BZK09]. We highlight two integral lines to emphasize the symmetry aspect of the fields.

The goal of this section is to provide an algorithm for the construction of quasi-symmetric cross-fields on surfaces, which strikes a balance between three important properties of fields: symmetry, smoothness, and alignment with local geometry; we found this combination to be an essential requirement for successful field construction: while coarse global symmetry is highly desirable, alignment with local, possibly non-symmetric features is essential as-well, and smoothness must be always guaranteed.

To construct symmetric fields, we need to choose a class of surface symmetries, that is, mappings of the surface to itself identifying symmetric points. Most previous works consider isometric maps (intrinsic or extrinsic, possibly with additional uniform scale) as the central model for symmetries, and make the assumption that the relevant symmetries are close to isometric. We formulate our algorithm for a broader class of *generalized symmetries* without making explicit assumptions about isometry.

There are three-fold advantages to this approach: (1) our method for symmetric field construction is less dependent on specific assumptions about the symmetry maps; (2) we can handle significant local deviations from isometry gracefully; and (3) in the case of genus zero surfaces, the concept of generalized symmetries and their properties lead to a robust and efficient algorithm for computing symmetry maps. While our symmetric field construction algorithm can use symmetry maps produced by different algorithms, we demonstrate that the new algorithm yields substantially better quality.

To summarize, the main contributions of this Section are:

1. the introduction of generalized symmetries, with focus on generalized reflections, and invariant N-symmetry fields;

2. an algorithm for the construction of quasi-symmetric N-symmetry fields, which maintains alignment to local (possibly asymmetric) features - the single tunable parameter is used to adjust the relative importance of symmetry vs smoothness in the output (see Figure 3.16);

3. an algorithm for computing generalized intrinsic reflection maps of surfaces of genus zero, providing robust input for the symmetric field computation algorithm.

3.3.1 Related work on symmetry detection

Several techniques based on voting have been proposed in the literature to detect either global or partial extrinsic symmetries. In [PSG$^+$06, CDPB08], symmetry planes are detected, while symmetric patches are found in [MGP06]. PIRS [XZT$^+$09] extends the voting approach to find stationary lines of partial intrinsic symmetries. However, PIRS does not provide a dense map of correspondences, and the detection of the stationary line for approximate (non-isometric) symmetries is not stable enough to support our algorithm.

Other techniques reduce intrinsic symmetry to extrinsic by "straightening" objects through deformation. [Mit07] proposed a fully automatic method, which works for symmetric

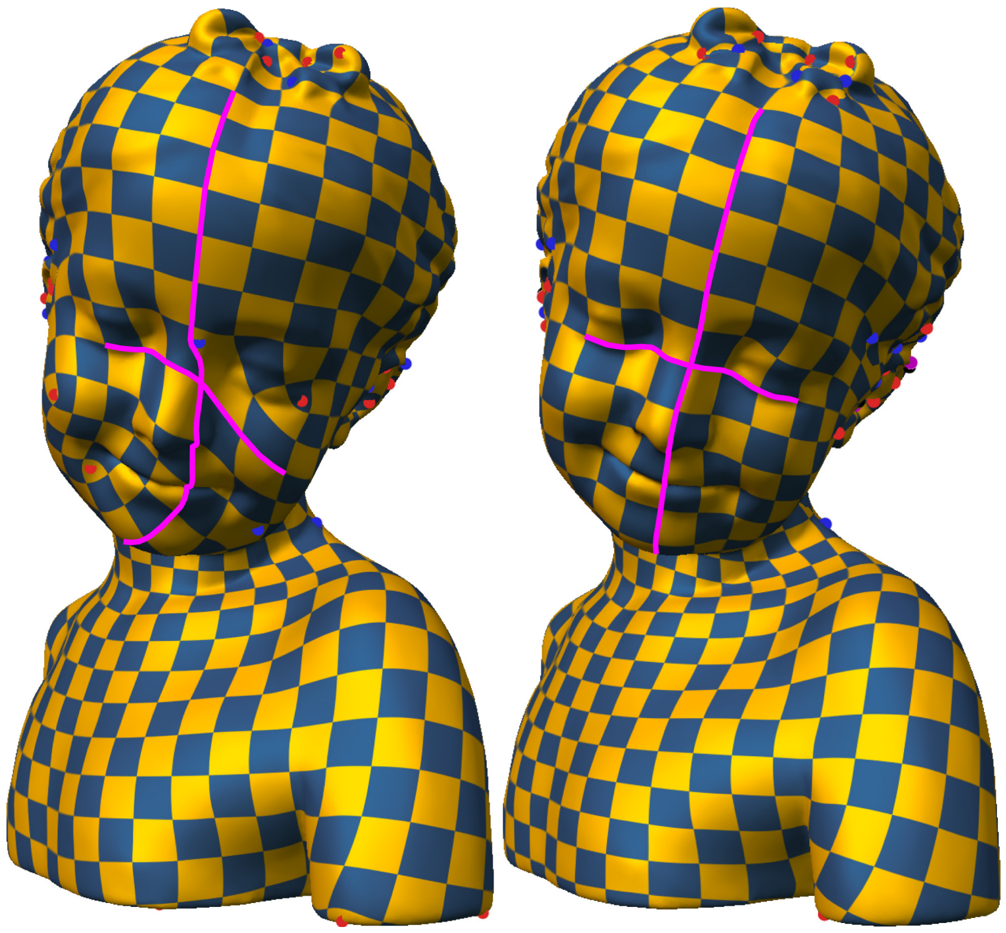


Figure 3.15: *Field-aligned parametrization of the Bimba model using the symmetry field construction method developed in this section, and using the MI technique of Bommes* et. al.*[BZK09]*

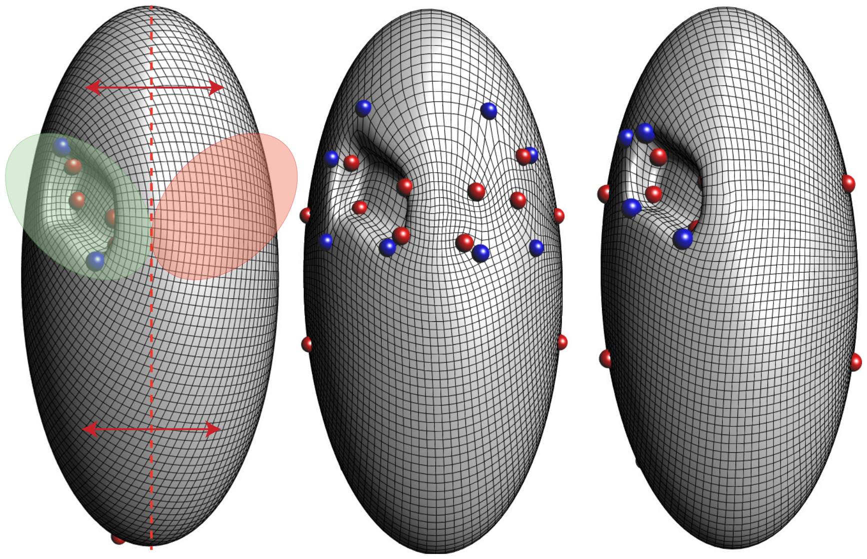


Figure 3.16: *Global symmetry vs local alignment on a synthetic model: (left) non-symmetric field obtained with plain MI [BZK09]; (center) global symmetric field transfers singularities also to the smooth side; (right) local alignment allows to better trade symmetry for smoothness.*

objects with multiple extrinsic symmetries, based on [MGP06]. A similar, approach is proposed in [KAG$^+$09, GAK10], requiring manual input for the starting phase. These approaches do not provide a dense map of correspondences.

In [GPF09, PGR07] some of the above methods are combined with mesh simplification techniques to produce symmetric triangle meshes.

A few other works address intrinsic symmetry through embedding methods that either reduce intrinsic to extrinsic symmetry [OSG08], or factor out symmetry by mapping sets of symmetric points to a single point in an embedding space [OMMG10, LCDF10]. These techniques are fully automatic; they may provide a dense (in a vertex-to-vertex sense) mapping of symmetries; and they may be easily generalized to find the stationary line. However, they are not robust to surface deformations that break isometry.

Embedding techniques, as well as voting techniques, do not exploit spatial coherence: point-to-point symmetry is estimated without taking into account what happens at nearby points. For this reason, they may be prone to errors such as false matchings and discontinuous mapping, which may severely hinder the application of extracted maps for our purposes.

Most recent approaches exploit spatial coherence by using surface parametrization: [KLCF10, KLF11] present fully automatic methods that first find sparse matchings of symmetric points, and then apply a Möbius transform that realizes extrinsic symmetry in parameter space, assuming a topologically restricted, conforming parametrization. These methods can provide continuous mappings of symmetries, but they are prone to severe errors in the presence of false matchings during the coarse phase.

To the best of our knowledge, approximate intrinsic symmetries in the non-isometric case have been addressed only in [RBBK07, RBBK10] by using an alternative definition to the one we present in Section 3.3.2. However, their definition (even in the continuous case) does not imply a smooth diffeomorphism, and the derived algorithms are combinatorial in nature and are meant more to assess the degree of non-isometry than to detect an explicit map of symmetry.

Given a set of few symmetric landmark pairs, one can think of using inter-surface mapping methods for building the symmetry map. A common approach is to find a common parametrization domain for the two surfaces [PSS01]. Later work [KS04, SAPH04] developed automatic algorithms to find a suitable base parameter meshes. However, these methods are general and do not exploit the fact that the final map should be a *symmetry.* For example, in case of bilateral reflective symmetry these methods will not force the existence of a stationary closed curve, as we require for our field construction.

3.3.2 Symmetric fields

Symmetries on a surface are usually defined as isometric automorphisms: *extrinsic* symmetries preserve Euclidean distance, while *intrinsic* symmetries preserve geodesic distance. The class of intrinsic symmetries trivially includes the class of extrinsic ones.

In practice, few surfaces have perfectly isometric symmetries, and deviations, sometimes quite large, need to be allowed. A symmetry map may not be isometric but it may be reduced to an isometry by a smooth deformation of the surface.

Since the symmetry map may not be isometric, looking for a map that is as isometric as possible will certainly lead to errors spread over the entire surface. These errors may be small but for our purposes even a minor error like mapping the tip of a finger to some part in the middle of the symmetric finger cannot be tolerated.

To be able to handle such maps, we regard any smooth automorphism of a surface as a symmetry, and focus on topological properties, as it is common in mathematical study of symmetries of surfaces (cf. [FM11]).

We consider N-symmetry fields defined on 2-manifolds, as defined in [RVLL08], and we study the properties that one such field must have to comply with a given symmetry on its domain. In Section 3.3.4.3, we discuss the generalization to *sets of symmetries.*

3.3.2.1 Generalized reflections

We focus on *reflections*, which account for most global symmetries observed in real objects. A reflection g and identity form a finite group of transformations, and each point has an orbit with respect to this group consisting of two points. In the extrinsic isometric case, all global symmetries for compact objects can be composed of reflections, as any 3D rotation can be decomposed into three reflections. While objects with rotational but no reflectional symmetries do exist, these are relatively rare.

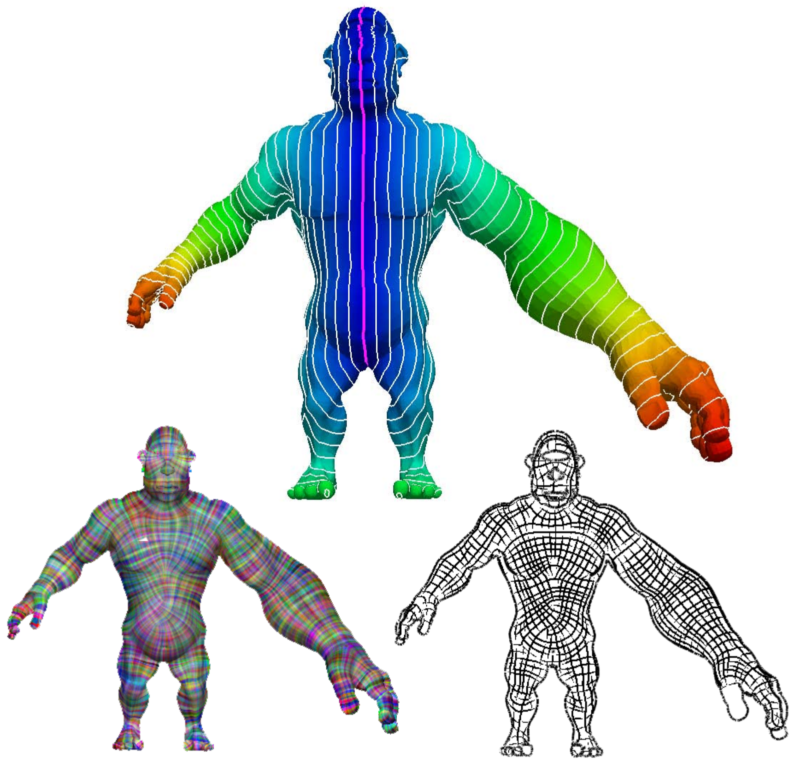


Figure 3.17: *A mode with non-isometric symmetry: color map represent symmetry and stationary line is depicted in magenta; our algorithm computes a field that respects this generalized symmetry.*

A diffeomorphism $g : M \to M$ is a smooth and invertible map (which is also smooth) from the surface to itself. $M(g)$ denotes the *stationary set* of g, i.e., the set of points p of M for which $g(p) = p$. For a point p on M, T_pM is the tangent plane at p. The differential of g at p, $Dg : T_pM \to T_{g(p)}M$ maps tangent vectors at p to tangent vectors at $g(p)$.

Generalized reflections. We adopt the following definition of a *generalized reflection* (cf. Koszul [Kos65]):

Definition 1 *A* reflection *on M is a diffeomorphism g, such that $g(g(p)) = p$ for all $p \in M$, and the set of non-stationary points of g is nonempty and disconnected.*

Stationary points of symmetry mappings g play a particularly important role in our construction. For a generalized reflection g, it turns out that the local behavior near stationary points is similar (although not identical) to the behavior of isometric reflections.

Lemma 1 *Let g be a reflection. If p is a stationary point of g, then:*

1. *the differential Dg_p has a stationary direction v;*
2. *for a choice of an orthonormal coordinate system on T_p the differential Dg_p has the form*
$$\begin{bmatrix} 1 & c \\ 0 & -1 \end{bmatrix}.$$
3. *[MZ55] There is a neighborhood $U(p)$, and a choice of smooth coordinates $h : U \to \mathbb{R}^2$ system on U such that g in these coordinates is a linear transformation A_g, i.e.*
$$g = h^{-1} \circ A_g \circ h \tag{3.1}$$

Proof. By Proposition 2, the differential Dg_p at a stationary point p has two eigenvalues -1 and 1 (see proof above). Let e_1 be the eigenvector corresponding to eigenvalue 1: e_1 is a stationary direction of Dg_p. Now let us assume a change of coordinate system on T_p that aligns the first coordinate axis to e_1. If we express Dg_p with respect to the new frame, it must necessarily have the form:
$$\begin{bmatrix} 1 & c \\ 0 & d \end{bmatrix}.$$
Since p is stationary, we must have $Dg_p^2 = \text{Id}$, hence $c + cd = 0$ and $d^2 = 1$. And since $\det Dg_p = -1$ we necessarily have $d = -1$.

In the proximity of the stationary line, Dg behaves as a linear reflection combined with a shear, and the value of factor c determines the amount of shear. If a map is conformal it follows that $c = 0$ and g is isometric at stationary points.

Global properties of generalized reflections are also similar to the familiar reflections about a symmetry plane of an object [Kos65]. More specifically, the following proposition holds.

Proposition 2 *If g is a reflection on M, then:*

1. *g is orientation-reversing;*
2. *the stationary set of g is a set of closed smooth curves on M (generalizes intersection with the symmetry plane);*
3. *$M' = M \setminus M(g)$ consists of two connected components M_1 and M_2*

4. g maps M_1 and M_2 to each other.

An important consequence of 2 is the following.

Corollary 3 *For surfaces of genus zero* (3.1) *holds globally, i.e. for a generalized reflection g there is a diffeomorphism onto the plane $h : M \to R^2$, such that $g = h^{-1} \circ A \circ h$, where A is a reflection.*

Proof. First, consider a stationary point p of g. As shown [MZ55], there is a neighborhood U of p and a choice of smooth coordinates $h : U \to \mathbb{R}^2$ system on U such that g in these coordinates is a linear transformation A_g. It follows that Dg has the form VA_gV^{-1} where V is the differential of the transformation h. As $Dg(p)^2 = I$ at stationary point, it follows that $A_g^2 = I$. All such matrices have two eigenvalues, and both its eigenvalues satisfy $\lambda^2 = 1$. If g is orientation preserving, then both eigenvalues are either 1 or -1. In the former case, g is identity on U, i.e. any stationary point has an open neighborhood of stationary points. On the other hand, the set of stationary points is clearly closed, as the limit of any sequence of stationary points is stationary by continuity of g. We conclude that an orientation preserving g is identity, which contradicts existence of non-stationary set. If g is orientation-reversing, at every stationary point, its differential Dg and linear form A has eigenvalues 1 and -1, and in $h(U)$ the stationary set of A is a line ℓ, corresponding to the stationary curve $h^{-1}(\ell)$ of g. As this holds for any stationary point, the stationary curve can be extended indefinitely to an embedding of the real line or a circle in M, forming a connected component of the stationary set. As the stationary set is closed, its connected components are also closed. But an embedding of a real line in a compact manifold cannot be closed; we conclude that the stationary set consists of embeddings of circles.

Consider a point p in one of the connected components M_1 of the non-stationary set M' of M, mapped to a component M_2. Consider the set of all points in M_1 mapped to M_2, i.e. $M_1 \cap g^{-1}(M_2)$. As M_2 is both open and closed in M', so is $g^{-1}(M_2)$ by continuity of g. Thus, $M_1 \cap g^{-1}(M_2)$ is also open and closed, so it has to coincide with all of M_1 as M_1 is connected, i.e. $g(M_1) \in M_2$. As $g(g(p))$ is p, by a similar argument, $g(M_2) \in M_1$, so M_2 and M_1 are mapped to each other, and $g(M_1) = M_2$. Consider a point p on the boundary of M_1. As locally g acts as a linear reflection, mapping one part of the neighborhood U of p to the other, U has to consist of two disconnected parts from M_1 and M_2, i.e., any point on the boundary of M_1 separates it from M_2. Then the union of M_1, M_2 and their boundary is closed in M and has no boundary, i.e., it has to coincide with M.

We construct this type of global parametrization for genus zero surfaces in Section 3.3.4.

3.3.2.2 N-symmetry fields

A *N-symmetry field* v on the surface is an assignment to every point $p \in M$ (excluding a set isolated singularities) of N unit vectors $v_1, \ldots, v_N$ lying on the tangent plane T_pM and forming equal angles of $2\pi/N$ between adjacent vectors. The most common and useful examples are direction fields, line fields and cross-fields (i.e., 1-symmetry, 2-symmetry and 4-symmetry fields, respectively), with 3-fields and 6-fields occasionally considered. We primarily focus on cross-fields, but the algorithm described in Section 3.3.3 applies to any value of N, and would be easily adapted to non-unit fields.

Transport of N-symmetry fields. To define N-symmetry fields that respect a symmetry map g, we need a way to compare the values of the field at different points. In order to do this, we must be able to *transport* the symmetry field at a given point p to the tangent plane at $g(p)$. The differential Dg defines a natural map $T_pM \to T_{g(p)}M$ for vector fields. If Dg is orthogonal, then it can be trivially extended to transport any N-symmetry field: the N vectors $Dg\ v_i(p)$ form a N-symmetry value (a set of N unit-length vectors separated by equal angles). In a more general case of N-symmetry fields, Dg applied to the component vectors of the field does not yield a new N-symmetry value unless Dg is an isometry. Broadening the class of fields we consider is possible, but undesirable, as many target algorithms expect orthogonal directions. Instead, we define an orthogonal transport $T_pM \to T_{g(p)}M$ associated with g, as the closest orthogonal transform to Dg. As g is orientation-reversing, this transform is a refelection.

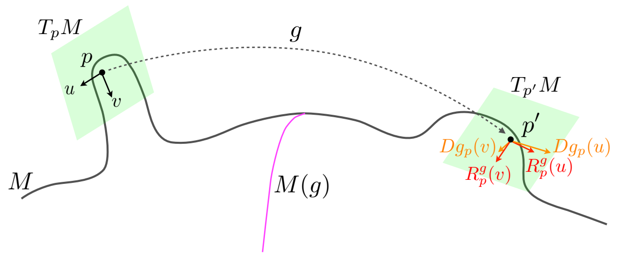


Figure 3.18: *Summary of notation for surface M with non-isometric reflection g. Orthonormal vectors u and v are transported to non-orthonormal orange vectors by differential Dg and to orthonormal red vectors by its closest orthogonal transform R^g.*

We map v using the closest orthogonal transform R^g to Dg, which can be obtained from the unique decomposition

$$Dg = R^g S^g, \tag{3.2}$$

where R^g is orthonormal and S^g is symmetric positive definite[1].Note that, if g is an isometry, we trivially have $Dg = R^g$.

[1]These decompositions can be computed, e.g., either via SVD: $A = U\Sigma V^T$, hence $R = UV^T$ and

Lemma 4 *$R^g(p) : T_pM \to T_{g(p)}M$ continuously depends on Dg, and if g is a reflection, then for every $p \in M$, and for any N-symmetry field v, we have $v(p) = R^g(g(p))R^g(p)v(p)$.*

Proof. Using the expression for R^g, we observe that it defines an analytic dependence of R^g on Dg, unless $det(Dg+Dg^T - Tr(Dg)I) = 0$, which, as can be seen by direct calculation, only happens if Dg is a similarity transformation. However, as Dg is orientation-reversing, this is not possible. Since $g^2 = Id$ then $Dg_{g(p)}Dg_p = I$. Since at a point p, $Dg = R^gS^g$, then $Dg^{-1} = S^{g^{-1}}(R^g)^T = (R^g)^TS'$ with $S' = R^gS^g(R^g)^T$ symmetric positive definite, so the closest orthogonal transform to $Dg_{g(p)}$ is $R^g(p)^T$, which implies the second statement of the lemma.

We are now at a position to define the symmetric N-symmetry fields:

Definition 2 *A N-symmetry field v is* symmetric with respect to a symmetry map *g if either p and $g(p)$ are both singularities of v, or*

$$R_p^g(p)v(p) = v(g(p)), \tag{3.3}$$

where R_p^g is the closest orthonormal transform to the differential Dg_p at a point p, as defined above.

N-symmetry fields on stationary lines. Symmetric N-symmetry fields have a very particular behavior at stationary points as we describe next. The next corollary follows from Lemma 1 and definition of R^g by observing that Dg at stationary points is a combination of linear reflection and shear and so R^g at that point is just a linear reflection:

Corollary 5 *If p is a stationary point of g, then $R^g : T_pM \to T_pM$ is a linear reflection.*

By definition, if v is a symmetric N-symmetry field, and p is a stationary point, then we must have $R^gv(p) = v(p)$. The above corollary also indicates that R^g is a linear reflection. This imposes stringent requirements on field v.

Proposition 6 *If v is symmetric with respect to a symmetry transform g and p is a stationary point of g, then one of the following holds: (1) v has a singularity at p, (2) one of the directions of v is the stationary direction t of R^g; (3) one of the bisectors of angles formed by consecutive vectors of v is aligned with t.*

Proof. Let us assume v is not singular at p, and let w be one of the N vectors of $v(p)$. Since v is stationary (as a N-symmetry field) for R^g, then R^gw must also be one

$S = V\Sigma V^T$; or through an explicit formula, $R^g = B/sqrt|det(B)|$, $B = (1/2)(Dg + Dg^T - Tr(Dg)I)$, with Dg expressed as a 2×2 matrix in an orthonormal basis.

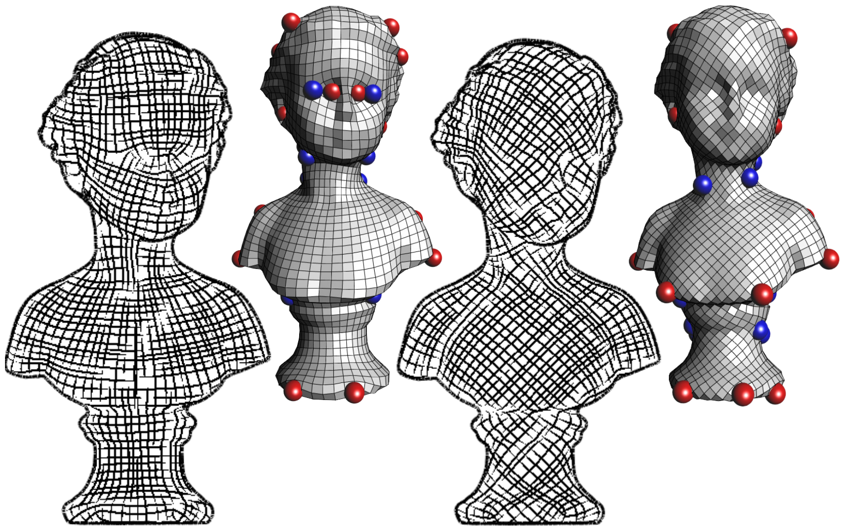


Figure 3.19: *A field can have just two possible orientations at the stationary line: results for a cross field either aligned with the stationary line, or rotated by $\pi/4$. Field singularities are depicted by blue/red bullets.*

of the vectors of $v(p)$, i.e., w and $R^g w$ must form an angle of $2k\pi/N$ for some integer $k = 0, \ldots, N-1$. Since R^g is a pure reflection about an axis t, this may happen only if w and t form an angle of $k\pi/N$.

The proposition implies that only two possible orientations at each point of the stationary line for a symmetric field v. Moreover, by continuity of v, the same choice must hold at all points along a connected component of the stationary line, unless it contains a singularity. We will use this fact as a basis of our algorithm for computing a symmetric field. Figure 3.19 shows the two possible choices for cross-field at stationary curve.

3.3.3 Field symmetrization

Given a mesh M, a generalized reflection g, and a set of orientation constraints, our algorithm computes an N-symmetry field v on M that is smooth, symmetric with respect to g, and aligned with the orientation constraints. The objectives of symmetry and smooth alignment to local shape features may be in conflict. Our algorithm provides a trade-off between the two, with a smoothness parameter chosen by the user. In fact, symmetry depends just on map g, while smoothness is highly influenced by the shape (more precisely, by Gaussian curvature) of M, which may be arbitrary, also at symmetric points, if g is non-isometric. The foundation of our algorithm is the algorithm [BZK09] for field optimization.

Algorithm outline. We assume that for the symmetry map g it is possible to: evaluate the stationary set $M(g)$ (realized as a set of segments on a subset of triangles); evaluate the field transport R^g (as rigid linear transform between triangles t, t'); and evaluate the orbit $\mathcal{O}(p)$ at any point p (for $p \in t$, we calculate $g(p) \in t'$, for some possibly other triangle t'). The evaluation of such data in specific cases will be addressed in Section 3.3.4. Our algorithm is not specific to a single reflection; we make no assumptions about the size of the orbit, or the specific origin of the transport map. This allows us to apply it in the case of sets of symmetries (Section 3.3.4.3).

Our method consists of the following steps:

1. Set hard orientation constraints at selected feature points.
2. Set additional hard constraints at the stationary curve of g, $M(g)$ (excluding conflicts with constraints from 1), by fixing one of the two possible orientations of v at stationary points (see Prop. 6), then extend field v to the rest of M by running the MI smoothing algorithm. We found that constraining the field on stationary lines is crucial for high-quality results.
3. Use field transport R^g (Equation (3.2)) to symmetrize field v, by averaging over orbits (excluding orientation feature points and stationary points). Denote the output field of this stage $\bar{v}$.
4. Repeat Steps 2, to obtain the final field v using $\bar{v}$ as soft constraint (this is where the parameter controlling smoothness vs. symmetry comes in).

Next, we describe the discrete representations of the field, symmetry map orbits and transport maps R^g.

Steps 1,2 - Constrained field optimization. Similarly to [RVLL08, BZK09], a discrete field v is represented at triangle t_i by an angle θ_i with respect to a local frame. With each e_{ij} separating triangles t_i and t_j, we associate a constant angle κ_{ij} and an integer variable p_{ij} (*period jump* or *matching*). The angle κ_{ij} is the rotation of the reference frame from t_i to t_j. The period jump p_{ij}, determines the additional rotation $2\pi p_{ij}/N$ of the N-symmetry field between values on triangles t_i and t_j. This rotation maps an N-symmetry value to itself, so it needs to be encoded separately (cf. [RVLL08])[2].

The smoothness energy of the MI algorithm is

$$E_{smooth} = \sum_{e_{ij} \in E} (\theta_i + \kappa_{ij} + \frac{2\pi}{N} p_{ij} - \theta_j)^2. \tag{3.4}$$

[2]We note that one can consider discrete principal connections instead of fields [CDS10], replacing matchings with a more geometrically natural notion of holonomy angles; we prefer [RVLL08] formulation as it allows for more natural handling of constraints.

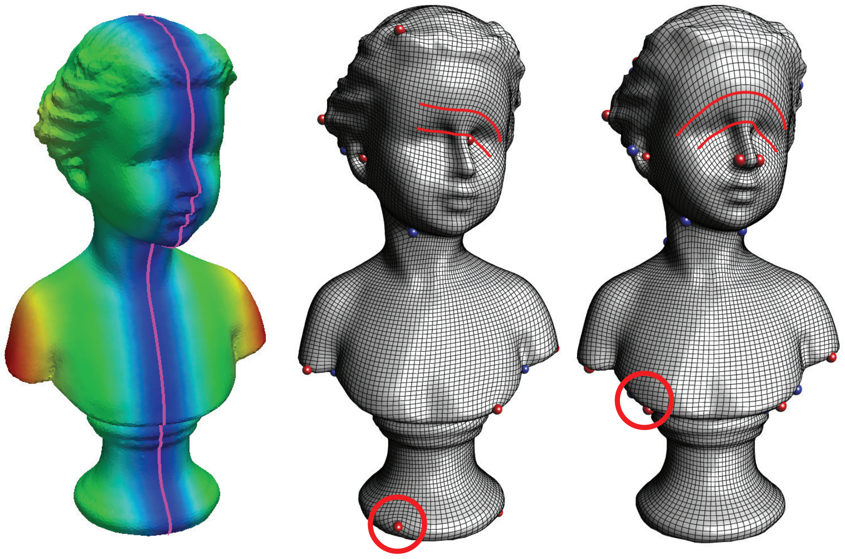


Figure 3.20: *Algorithm steps. From the left: input mesh with symmetry; after constrained field optimization algorithm follows stationary line but it is not symmetric everywhere; after symmetrization and final optimization field is fully symmetric. Red lines sow field flow; circles highlight non-symmetric singularities that become symmetric after optimization.*

Since variables θ_i and θ_j are real, while the p_{ij}'s are integer, optimization of the field v with respect to Energy (3.4) is a mixed-integer problem. This problem has a space of equivalent solutions: to make the minimizer unique, the value of p_{ij} is fixed at a subset of edges. Feature constraints are given as a set of angles $\{\hat{\theta}_{i_1}, \ldots, \hat{\theta}_{i_k}\}$ on a subset of triangles which remain fixed during optimization; we add stationary lines constraints described below to this set. A greedy mixed-integer optimzation algorithm is used for optimization. (see [BZK09] for details).

Stationary set constraints. For all $g \in G$, the stationary set $M(g)$ is given by a set of line segments l_i on a subset of triangles of M. The direction of l_i is the stationary direction of R^g at t_i. We add a hard constraint by expressing the direction of l_i as an angle $\hat{\theta}_i$ with respect to local coordinate frame at t_i. Alternatively, the angle $\hat{\theta}_i + \pi/N$ can be used, according to Prop. 6. This choice is left to the user and it must be consistent for all triangles intersecting a given connected component of $M(g)$. In our experiments, we always constrained v to be aligned with the stationary line. Figure 3.19 shows results obtained by using the alternative direction. Theoretically, it is also possible to have singularities on the stationary line. In this case, the field on the stationary line might have both orientations and the change between the two will happen on singularities only. It is not easy to automatically decide where singularities should be placed on the stationary line and in our experiments we always used a single orientation over the entire line.

Energy (3.4) is minimized by freezing all variables corresponding to hard constraints (see [BZK09] for details). The result of this phase already improves over the standard MI in terms of field symmetry in the proximity of the stationary line, while it does not warrant symmetry far from it (see Figure 3.20 center).

Step 3 - Symmetrization by field transport. Field v is symmetrized by averaging it over orbits of symmetry. We set the convention that field values are attached to triangles' centroids $c_i \in t_i$.

Fuzzy orbits. Since discrete symmetry maps do not map triangles exactly to triangles, we use the notion of fuzzy orbits (similarly to [LCDF10]) in order to define a symmetry averaging operator. In particular, given a non-stationary triangle t, we define its fuzzy orbit $\mathcal{O}(t)$ as the union of all triangles in its 1-ring and and the 1-ring of triangle containing $g(c)$. We assign a weight $s_i(t)$ to every $t_i \in \mathcal{O}(t)$ inversely proportional to its distance from $\mathcal{O}(c) = \{c, g(c)\}$, namely we set

$$\widetilde{s}_i(t) = \Phi(\min_{c' \in \mathcal{O}(c)} \|c_i - c'\|),$$

where we picked $\Phi(r)$ to be a Gaussian with standard deviation equals the maximum of the triangles' 1-ring diameter. To define averaging, $\widetilde{s}_i(t)$ is normalized to have a unit sum:

$$s_i(t) = \widetilde{s}_i(t) / \sum_i \widetilde{s}_i(t).$$

Averaging over orbits. For every triangle t, we average the field values over the orbit $\mathcal{O}(t)$ using the weights $s_i(t)$ and the field transport R^g as follows.

For every $t' \in \mathcal{O}(t)$ we transport the field value at t' to t using $R_{t',t}$. $R_{t',t}$ is defined as $R_{t',t} = (R^g_t)^{-1} R_{loc}$, where R^g_t is the field transport from t to the triangle $\widetilde{t}$ containing $g(c)$ (c centroid of t), and R_{loc} is the closest rotation in 3D taking t' to $\widetilde{t}$. Following [PZ07], we observe that the N-symmetry field can be represented in a given frame by a vector $w = [\cos(N\theta), \sin(N\theta)]$, eliminating the $2\pi/N$ ambiguity. Then, instead of transporting the N-symmetry field vectors by $(R^g_t)^{-1}$ and then converting to the N-th power vectors w can be transported directly by $R^{-N}_{t',t}$. The symmetrized value of the vector field is defined explicitly by normalizing

$$w^{sym}(t) = \sum_i s_i(t) R^{-N}_{t_i,t} w(t_i).$$

Step 4 - Optimization with soft symmetry constrains. Field $\bar{v}$ obtained from the previous step will be mostly smooth, but smoothness can be broken in some parts of M, namely:

- in the proximity of hard constraints, since the field is fixed at such triangles, while it is possibly deviated by symmetrization at neighboring triangles;
- in the proximity of singularities and of their symmetric mates, since directions of the field are arbitrary at singularities, thus symmetrization may produce invalid results.

Discontinuities may also introduce many undesirable singularities in the field. Therefore, we smooth $\bar{v}$ further by minimizing a modified energy, using values $\bar{\theta}_i$ as soft constraints. Following [BZK09], we measure the *local roughness* of $\bar{v}$ at each triangle t_i as the constant-weight discrete Laplacian of the field at t_i:

$$\delta_i = \min_{\bar{p}_{ij}} \sum_{t_j \in \mathcal{N}_i} (\bar{\theta}_i + \kappa_{ij} + \frac{2\pi}{N}\bar{p}_{ij} - \bar{\theta}_j)/3,$$

where $\mathcal{N}_i$ contains the three neighbors of t_i and the $\bar{p}_{ij}$ can take values in $0, \ldots, N-1$. We note that this method is justified only for uniform triangulations, but we did not see a significant difference in practice, and using dual-mesh discrete Laplacian results in less stable behavior for badly shaped triangles.

Since there are just N^3 possible combinations and N is a small number, the minimum is computed in a brute force (combinatorial) way.

We define the symmetry constraint weight:

$$w(\delta_i) = max(0, \delta_0 - \delta_i)/max_i(\delta_0 - \delta),$$

which is zero if roughness exceeds a threshold δ_0. Again, assuming a uniform triangulation, δ_0 is the maximal rate of field rotation for the average triangle size.

Finally, we define the modified energy as follows:

$$E_{symm} = (1-\alpha)E_{smooth} + \alpha \sum_{t_i \in M} w_i(\delta_i)(\theta_i - \bar{\theta}_i)^2, \tag{3.5}$$

with the second term constraining the field values to be close to symmetric values. The final result is thus obtained by minimizing Energy (3.5) with the same hard constraints and the same mixed integer solver used in Step 2.

Parameter α is the main parameter of our method, and it is used explicitly to allow the user selecting either a smoother or a more symmetric result. In Figure 3.26, we report results obtained with different values of α. Note that $\alpha = 0$ necessarily yields the same result that we obtain after Step 1; while, as α approaches value 1, field smoothing is progressively restricted just to areas where roughness exceeds threshold δ_0.

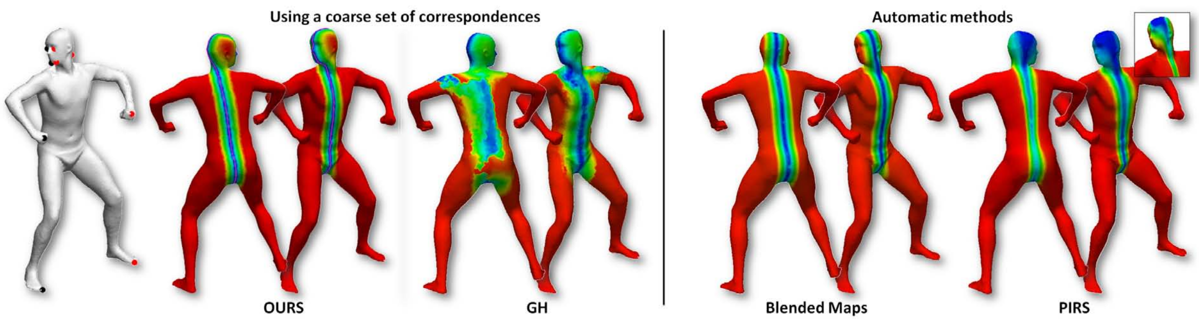


Figure 3.21: *Symmetry detection methods on a human with non-isometric deformation of the neck. On right we show automatic methods: Blended intrinsic maps [KLF11], and Partial Intrinsic Reflectional Symmetry (PIRS) [XZT⁺09]; note that these erroneously extrapolate the dominant body symmetry to the head. On left, we use a set of coarse correspondences (shown as black and red dots on the leftmost figure); we show GH-type interpolation [MS04, BBK06], side by side with our generalized reflection map. Our map is smooth and have analytic representation of the symmetry stationary curve.*

3.3.4 Symmetry detection algorithms

Our construction of symmetric N-symmetry fields requires a generalized reflection mapping g, as well as ways to evaluate its stationary set $M(g)$, and its transport R^g, as described in Section 3.3.2. The quality and precision of this transform, its stationary curve, and its transport are crucial, because they guide the field construction directly.

In this section, we first show that existing symmetry detection methods do not meet the needs of our algorithm. Then, we present a simple novel semi-automatic method for detection of generalized bilateral intrinsic symmetries in genus zero surfaces; and a simple embedding method to detect extrinsic symmetries on surfaces of arbitrary genus.

3.3.4.1 Intrinsic symmetry by fixed-point circle

In this section, we assume our surface M is of genus zero.

We have tested several existing automatic algorithms for intrinsic symmetry detection. While on a number of models sufficiently close to extrinsically symmetric or perfect isometry these provide good quality symmetry maps, we found that for many shapes with clear intrinsic symmetry, we could not get orbits and/or stationary lines and and/or transport maps with sufficiently high quality suitable for our algorithm. For this reason, we decided to require a small number of user defined correspondences. Even then, testings with different manifold interpolation methods was unsuccessful, and therefore we have developed a novel method for intrinsic symmetry interpolation from a coarse set of given symmetric points. The maps we compute are smooth and provide a stable set of inputs for our algorithm: reliable orbits, smooth stationary lines and robust transport maps. At the end of this subsection we describe in more detail relation to previous work.

Representing generalized reflection as linear reflection Motivated by Corollary 3, we next describe a simple novel method to compute a generalized reflection $g : \mathbf{M} \to \mathbf{M}$, given as input several pairs of symmetric landmarks (correspondences) $(p_i, q_i) \subset M \times M$, $i = 1, .., n$. Since in addition to be able to evaluate g at every surface point of M we would need an easy way extract the stationary set $M(g)$ and the transport R^g, we will follow Corollary 3 and compute a map $\phi : M \to \widehat{\mathbb{C}}$ ($\widehat{\mathbb{C}}$ is the extended complex plane[3]) that transforms g to a simple global linear reflection in the extended complex plane $\widehat{\mathbb{C}}$. That is, if we denote $\sigma(x + \mathrm{i}y) = x - \mathrm{i}y$ (the reflection w.r.t. the real axis) then, $\sigma = \phi \circ g \circ \phi^{-1}$, and equivalently,

$$g = \phi^{-1} \circ \sigma \circ \phi. \tag{3.6}$$

This point of view has several advantages: 1) it provides a (global) linear representation of the generalized reflection as a simple linear reflection σ; 2) it provides an analytic representation of the stationary curve $\mathrm{Im}(z) = 0$; and 3) it allows defining the operator Dg (from which we extract the transport R^g) in the discrete case in a natural way (as we explain below).

Note that using a per-vertex definition of a map $g : M \to M$, mapping vertices to points on triangles, it is not trivial to define Dg or extract the stationary curve $M(g) = \{g(p) = p\}$ in a robust way numerically.

From the purely topological point of view, there is much freedom in choosing g, ϕ that satisfies eq. (3.6), even when given a set of landmarks $\{p_i, q_i\}$. To restrict the space of possible maps, we add a natural constraint that if there is an *isometric* reflection that interpolates the given landmarks, our map will reproduce it. [KLCF10] observe that if the surface is mapped conformally to the extended plane $\varphi : M \to \widehat{\mathbb{C}}$, then perfect intrinsic symmetries $g : M \to M$ has to be an anti-Möbius map over $\widehat{\mathbb{C}}$, i.e., $\varphi \circ g \circ \varphi^{-1} = \widetilde{m}$, where $\widetilde{m}(z) = a\overline{z} + b/c\overline{z} + d$, for some $a, b, c, d \in \mathbb{C}$ with $ad - bc \neq 0$. To obtain a symmetry map for a set of landmarks mapped to $\widehat{\mathbb{C}}$, [KLCF10] fits an anti-Möbius transform and uses GH-type (Gromov-Hausdorff) interpolation. However, as the surface deviates significantly from being perfectly intrinsically symmetric this approach will suffer from three main drawbacks in our context: 1) it will result in non-continuous (and definitely not smooth) symmetry maps; 2) the least-squared fitted anti-Möbius $\widetilde{m}$ will not necessarily satisfy $\widetilde{m}^2 = Id$, nor

[3]The extended complex plane is the complex plane $\mathbb{C} = \{x + \mathrm{i}y \mid x, y \in \mathbb{R}\}$ with infinity added to it, that is $\widehat{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$.

will it generically have a curve stationary set; and 3) it will not get us to the desired representation where we have a linear reflection describing the symmetry g, as we required in (3.6).

We propose a new approach with two key ingredients. First, we use anti-involutions, rather than general anti-Möbius transformations, a subset of the anti-Möbius transformations that satisfy $\widehat{m}^2 = Id$ [Sch79] which is consistent with our definition of generalized reflection (Definition 1). An important property of anti-involutions is that they represent an inversion w.r.t. a circle which is their stationary set $\widehat{m}(z) = z$ (consistent with Proposition 2 and Corollary 3). This stationary circle is our initial guess of the stationary curve. Second, we will define our diffeomorphism ϕ using a *smooth* perturbation of the anti-involution. In particular our algorithm consists of the following steps:

1. Calculate the conformal map $\varphi : M \to \widehat{\mathbb{C}}$ of the genus zero mesh to the plane;
2. Transform the landmarks to the extended complex plane $z_i = \varphi(p_i)$, $w_i = \varphi(q_i)$, $i = 1, .., n$;
3. Fit (in the least-squares sense) an anti-involution $\widehat{m}(z_i) \approx w_i$ to the input landmarks;
4. Extract the stationary circle C of the anti-involution $\widehat{m}(z) = z$;
5. Map via a Möbius transformation $m(z)$ the circle C to the real axis;
6. Extract optimal (in the least-squares sense) symmetric configuration of the landmarks $m(z_i), m(w_i)$ w.r.t. the real axis;
7. Use smooth deformation (thin-plate splines) ψ to move the landmarks to their optimal symmetric configurations $\psi(m(z_i)) = \overline{\psi(m(w_i))}$;
8. The final map is $\phi = \psi \circ m \circ \varphi$.

The first and second stages are performed similarly to the algorithm of [KLCF10].

Step 3–4. To find the anti-involution Möbius $\widehat{m}(z)$ so to satisfy as much as possible $w_i = \widehat{m}(z_i)$, $i = 1..n$ we use the following lemma:

Lemma 7 *An anti-Möbius transformation $\widetilde{m}(z) = a\overline{z} + b/c\overline{z} + d$ is an anti-involution iff the matrix of coefficients $\begin{pmatrix} c & d \\ -a & -b \end{pmatrix}$ is hermitian, that is $d = -\overline{a}$, and $c, b \in \mathbb{R}$*

For a proof see [Sch79] (page 79). Therefore an anti-involution can be written as $\widehat{m}(z) = a\overline{z} + b/c\overline{z} - \overline{a}$, where $b, c \in \mathbb{R}$. We fit an anti-involution by multiplying both sides of the equations $\widehat{m}(z_i) = w_i$ by $c\overline{z_i} - \overline{a}$, and solve in the least-squares sense the resulting linear equations:

$$cw_i\overline{z}_i - \overline{a}w_i - a\overline{z}_i - b = 0 \ , \ i = 1, .., n.$$

One delicate point is that the stationary circle of an anti-involution can be imaginary and therefore will not form a "real" circle in $\mathbb{C}$. This would happen if in solving the above equations our solution gives $|a/c|^2 + b/c < 0$. However, this means that the deviation from isometry is extreme and we did not find this a problem in practice.

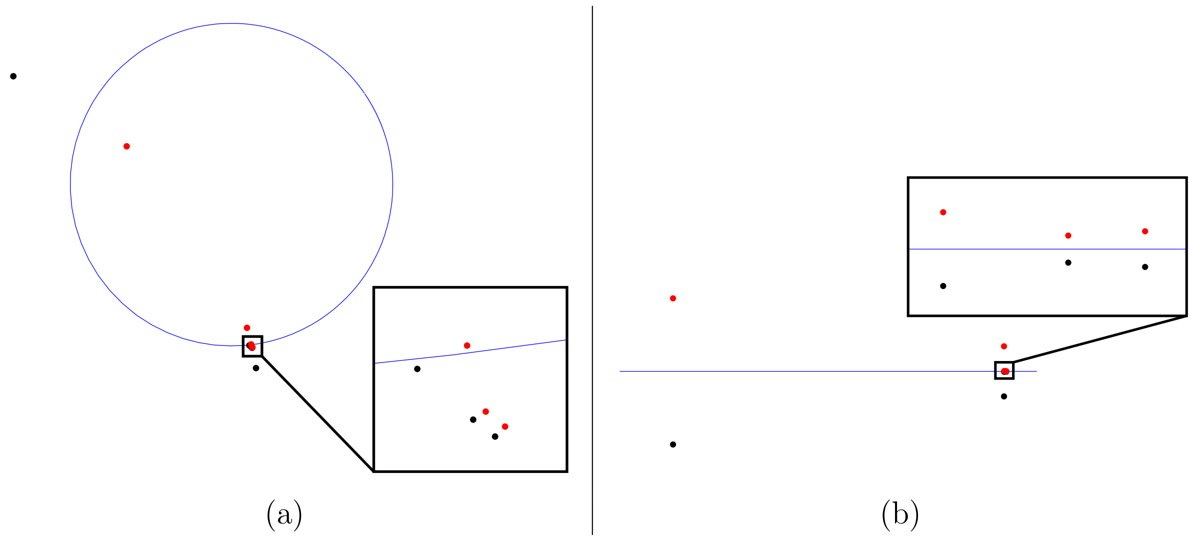


Figure 3.22: *Two stages in the intrinsic symmetry fit of the model from Figure 3.21: (a) shows fitting the best circle to a set of symmetric landmarks (red and black). Note how the landmarks in the head area (blow-up figure) are not conforming with the global symmetry. (b) shows the points after a smooth map is applied to bring them to be perfect reflectional symmetric w.r.t. the real axis; the head points are also symmetrized, as the blow-up figure shows.*

Step 5. Our next step is to find a Möbius transformation taking the circle stationary set of the anti-involution $\widehat{m}$ to the real axis. The stationary circle of $\widehat{m}$ has the explicit equation $|z - z_0| = r^2$, where $z_0 = a/c$, and $r^2 = |a/c| + b/c$. Figure 3.22 (a) shows the circle C for the example in Figure 3.21 (the symmetric landmarks are shown in red and black, consistently with the left image in Figure 3.21). Note how the landmarks on the head (shown in blow-up figure) do not agree with the global intrinsic symmetry as revealed by the circle.

For the Möbius transform mapping it to the real axis there are three degrees of freedom to set. We simply take three equal distant points t_1, t_2, t_3 on the circle C and find the Möbius transform m by solving a system of equations of the type $at_i + b = (ct_i + d)x_i$, $i = 1, 2, 3$, where $x_i \in \mathbb{R}$ are three points on the real axis.

Step 6–7. Next, we map the landmarks z_i, w_i with m and extract L^2 symmetric optimal points $s_i \in \mathbb{C}$, $i = 1, .., n$ such that $s_i = \operatorname{argmin}_z [|s_i - m(z_i)|^2 + |\overline{s_i} - m(w_i)|^2]$. It is not hard to see that the minimizer is $s_i = (m(z_i) + m(w_i))/2$. Finally, we find a Thin-Plate Spline $\psi : \mathbb{R}^2 \to \mathbb{R}^2$ such that $\psi(m(z_i)) = s_i$, and $\psi(m(w_i)) = \overline{s}_i$. Our final map is

$\phi = \psi \circ m \circ \varphi$, the symmetry map in these new coordinates is $z \mapsto \overline{z}$ and the stationary curve is $\mathrm{Im}(z) = 0$. Figure 3.22(b) shows the landmarks z_i, w_i (in red and black, resp.) after applying $\psi \circ m$ to it; note that they are now perfectly symmetric w.r.t. the real axis. Figure 3.21 shows (in purple) the stationary curve $\mathrm{Im}(z) = 0$ back-projected to the original model.

Computational complexity. The algorithm for finding ϕ, and $g = \phi^{-1} \circ \sigma \circ \phi$ requires solving one sparse linear system in the size of the mesh for the discrete harmonic part (standard cotangent Laplacian) of the uniformization, followed by linear time computation of its conjugate. The next steps of the algorithm are negligible computationally and require solving two linear systems in the order of number of landmarks (we typically used 5 to 10 landmarks).

Evaluation of orbits, stationary lines and transport. Function ϕ provides a parametrization for M, which can be evaluated in both directions. Therefore, given a point $p \in M$, its symmetric mate $g(p)$ is evaluated through Equation 3.6. The stationary line $M(g)$ is simply the image of the real line $\mathrm{Im}(z) = 0$ through function ϕ. More in detail, given a triangle t_i of M, let $\bar{t}_i$ be its image through ϕ in $\widehat{\mathbb{C}}$. Triangle t_i intersects the stationary line if and only if $\bar{t}_i$ intersects the real axis; and the linear segment of stationary line corresponding to t_i has endpoints at $\phi^{-1}(a)$ and $\phi^{-1}(b)$, where a and b are the intersections of edges of $\bar{t}_i$ with the real axis.

Evaluation of transport R^g. In principle, the differential Dg and, as a consequence R^g can be obtained by computing the gradient of g numerically; while the maps we construct in the intrinsic case are quite smooth, we found that following method to provide the most robust results. (We use this method also in the extrinsic case, as described in the next subsection.) Let $\zeta, \xi : M \longrightarrow \mathbb{R}$ be two scalar functions such that ζ is symmetric and ξ is antisymmetric, i.e., $\zeta(p) = \zeta(g(p))$ and $\xi(p) = -\xi(g(p)$. By chain rule,

$$\begin{array}{ll} \nabla\zeta(p) = \nabla\zeta(g(p))Dg_p & \text{and} \\ \nabla\xi(p) = -\nabla\xi(g(p))Dg_p. & \end{array} \tag{3.7}$$

where Dg_p is the differential of g computed at p. System (3.7) allows us to solve for Dg at any point, if the surface gradients of ζ and ξ are computed everywhere. R^g is then obtained from Dg as described in Section 3.3.2. In our case, we choose $\zeta(p) = \mathrm{Re}(\phi(p))$ and $\xi(p) = \mathrm{Im}(\phi(p))$.

Comparison to previous methods for intrinsic symmetry detection. A simple example is shown in Figure 3.21. This model of a human (SCAPE dataset [ASK+05]) has a dominant extrinsic reflection symmetry (neck down) while the head is rotated to the right. The deformation at the neck area involves a considerable metric distortion with respect to the rest pose, which makes recognizing head symmetry difficult for a global method. This problem is reinforced since the head has pretty good continuous rotational symmetry excluding negligible (in terms of area) features like ears and nose. Figure 3.21 shows the result of two automatic state-of-the-art-methods: PIRS [XZT$^+$09], and Blended Intrinsic Maps [KLF11]. PIRS is a particularly suitable algorithm for extracting stationary lines. However, as it is based on aggregating votes, the dominant extrinsic symmetry of the surface in this case overrides the symmetry of the head. Blended Intrinsic Maps also use a global deformation energy and therefore "extrapolate" the dominant extrinsic symmetry to the head.

We also compare to a method in the spirit of Gromov-Hausdorff (GH) [BBK06, MS04], where geodesic distances to the known landmark points are used to define feature vectors and closest point in the corresponding feature space defines the correspondences. Using five pairs of symmetric points (shown at the left of Figure 3.21) with GH-type interpolation produces a reasonable result in certain areas (like the head and the lower torso) where there are relatively many landmarks and/or geodesic distances reliably detect the symmetry transform. However, in vicinity of intrinsically distorted areas, where geodesics are not reliable, this method tends to produce worse results (e.g., the neck and left shoulder area). Another drawback of this type of methods in our context is that the correspondence mapping is not necessarily smooth or even continuous, and it hard to reliably extract the field transport from it. Lastly, given the set of correspondences it is also not trivial to extract the stationary line to be a closed curve, see for example the bottom and neck area again where any such curve is likely to be cut.

3.3.4.2 Extrinsic symmetry through embedding.

In the extrinsic case, a reflection mapping g is a simple orthogonal reflection about a plane Π in 3D space, its stationary set $M(g)$ coincides with $M \cap \Pi$, and its differential $Dg = R^g$ is g itself.

For this simpler case, we have combined two existing algorithms to find the reflection g and the stationary set $M \cap \Pi$ in a robust way. Inspired by [OFCD02] we have built extrinsic symmetry invariants at every point and used it to define a symmetry correspondence matrix, similar to [LCDF10]. The stationary plane was found by averaging the points' 3D coordinates over fuzzy orbits, and the final reflection g was set according to that plane. Let us briefly elaborate on each step.

Symmetry extrinsic invariant histograms. For each triangle t of M, we compute the Euclidean distances from its centroid c_t to the centroids of all other triangles of M, and we build a histogram of these distances using b bins. (We have used $b = 50$ for all our results.) We denote by $\Psi(t) \in \mathbb{R}^b$ the histogram vector for triangle t. Next, we fill a symmetry correspondence matrix S, by setting $S_{i,j} = \max_d - d(\Psi(c_{t_i}), \Psi(c_{t_j}))$, where d is the Euclidean 2-norm, and $\max_d$ is the diameter of the set $\{\Psi(t)\}_t$. We make sure S is sparse by keeping only 50 biggest values at each row and zero everywhere else. Finally we normalize each row of S to make it row-stochastic. The i-th row of S provides the fuzzy orbit of triangle t_i.

Calculating the symmetry plane, and stationary curve $M(g)$. Let $C_t \in \mathbb{R}^{n\times 3}$, where n is the number of triangles, be the matrix of all centroids' 3D coordinates. The product SC_t averages the 3D coordinates over the the orbit and therefore projects the points onto the stationary set (in 3D) (see [LCDF10]). We fit a plane Π using Principal Component Analysis (PCA). The stationary line is extracted simply by collecting the triangles of M that intersect Π.

Evaluation of field transport R^g. We perform a change of coordinates that places the origin in Π and aligns the z axis with the least significant direction yield by the PCA. Then we estimate $Dg = R^g$ by means of the same procedure explained in the previous section, by setting $\zeta(p) = p_x$ and $\xi(p) = p_z$, where $p = (p_x, p_y, p_z)$ is a generic point of M.

3.3.4.3 Fields symmetric with respect to sets of transforms

Next we explain how the theory and algorithms of the previous sections apply in the case when a shape has multiple symmetries. It is natural to define a field to be symmetric with respect to a set G of transforms, if it is symmetric with respect to all elements of G. For example, G may consists of reflections about two or more planes of symmetries. The input to our algorithm requires orbits, stationary lines, and transport maps.

The orbit for a point p with respect to a set G are simply the union of the orbits for individual transforms in G (we want to symmetrize our field at a point p with field values at all points $g(p)$, $g \in G$).

Similarly, the stationary set of interest is the union of the stationary set of individual elements of G. Note that if a point p is in two stationary sets $M(g_1)$ and $M(g_2)$, there are two distinct alignment requirements for the field. If the stationary directions of $R^{g_1}(p)$ an $R^{g_2}(p)$ are orthogonal, then both have to be satisfied at once, which many not always be possible, in which case the best we can do is to minimize the deviation from symmetry.

Defining the transport map requires a consistency condition between symmetry maps: there may be two maps f and h mapping a point p_1 to the same point $p_2 = f(p_1) = h(p_2)$.

We want the transports defined by f and g to be the same, so we require

$$R^f v = R^h v, \text{ at } p_1, \text{ for any } f \text{ and } h \text{ with } f(p_1) = h(p_1). \tag{3.8}$$

Observe that in this case, $h^{-1} \circ f$ is a non-identity transform with p_1 as a stationary point. so the field at p_1 and the map $R^{h^{-1}} R^f$ has to satisfy Proposition 6. In the cases of primary interest to us (sets of generalized reflections) the compositions of distinct reflections have, in general, only isolated stationary points which have little effect on the overall field behavior.

For all other points p, for each q in the orbit of $\mathcal{O}(p)$, there is a unique g mapping p to q, which defines the transport of the field from p to q unambiguously.

Rotations. So far, we have considered generalized reflections only. A rigid rotation can be computed as a composition of two reflections, and in general it has only isolated stationary points on a surface. It is natural to generalize the rotations as compositions of two reflections with isolated stationary points. This allows direct extension of all constructions we need to the case of sets including such generalized rotations.

3.3.5 Results

We have tested our methods on a range of geometric objects with approximate intrinsic and extrinsic symmetries (see Table 3.2). In this section, we describe our findings, discuss alternative approaches and design choices we made.

Symmetry detection. The method for intrinsic symmetry detection required specifying between 5 and 10 landmarks (as specified in Table 3.2, and typical running times were between one and five minutes. Comparison with other methods have been discussed in Section 3.3.4 and in particular Figure 3.21. The method for extrinsic symmetry detection is about five times slower (as it requires creating histograms of all pair distances), but it is fully automatic and it works for objects of any genus. On objects where both methods apply, namely extrinsically symmetric, results are overall similar, possibly with a slightly difference in cone placement, see Figure 3.23.

Reflecting cone singularities does not work. Once the symmetry map is available, a naive approach would be to first compute a non-symmetric field (with MI); then reflecting the cone singularities found on one side of the stationary line to their symmetric mates; and finally computing a field constrained to such singularities. As shown in Figure 3.24, while this may work in some cases (bunny), in general the field fails to follow the stationary line (e.g. Bimba model) and it may also introduce very large distortions (e.g. the right leg of the human model).

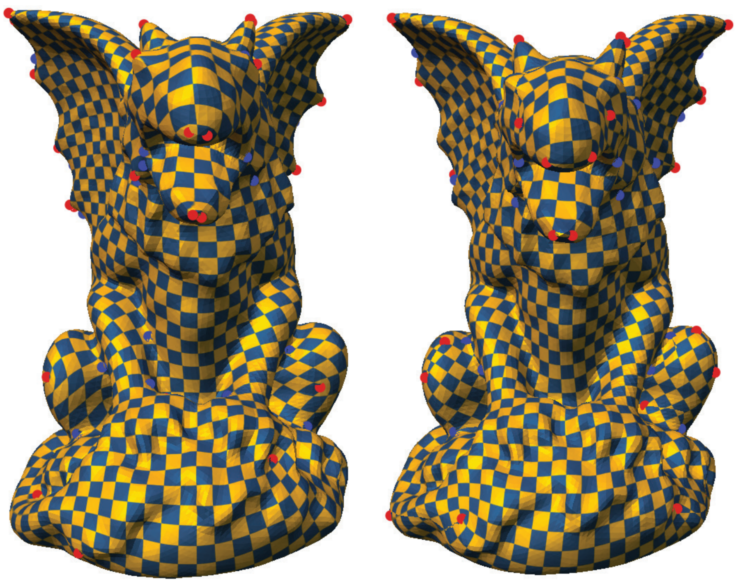


Figure 3.23: *Comparison of extrinsic (left) and intrinsic (right) symmetry detection on an extrinsically symmetric model. There are small differences in cone placements but the global the field alignment is similar.*

Stationary line constraints. We found that stationary line constraints plays an essential role in the quality of the results; Figure 3.25 compares the result of our algorithm with the result we obtain with no stationary line constraints. Note that without this constraint, although generally symmetric the field is not "parallel" along the stationary curve.

Multiple symmetries. Figure 3.28 shows experiments with multiple symmetry objects. For multiple stationary lines, the field is fixed on their union. Note how cones tend to respect symmetry relations in this case as well.

Effects of smoothness parameter. Our algorithm have three parameters: two parameters defining the features used in the mixed-integer field optimization, and the smoothness parameter α. The feature-detection parameters are the same as in [BZK09], and their effect is limited to changing the number of constrained faces.

Figure 3.26 shows dependence of the results on the choice of α: the higher α is, the sharper the transition between features and symmetrized areas; as a result the field's smoothness decreases and additional cones may appear.

Comparison to state-of-the-art. As there are no direct analogs of our method, we compare to the closest method, namely [BZK09]. Our claim that is that symmetry should

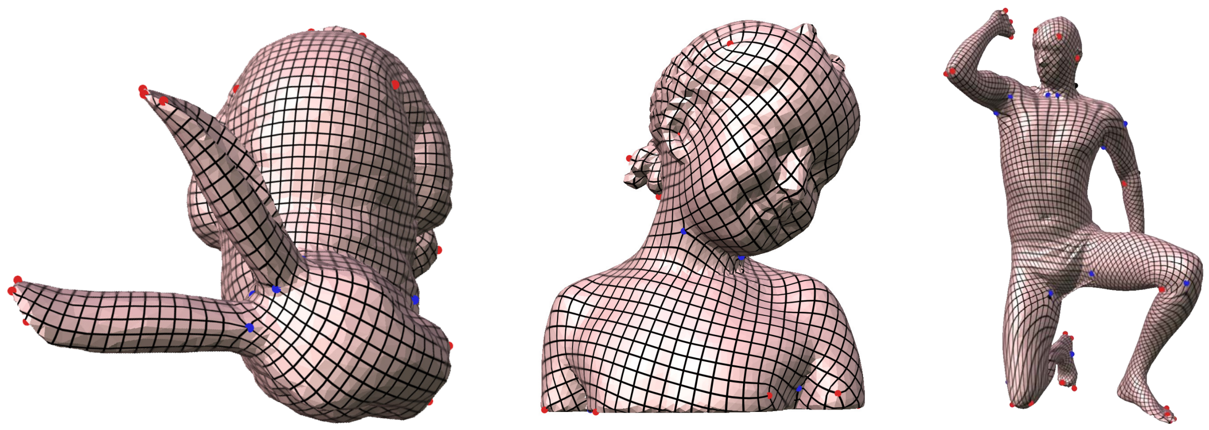


Figure 3.24: *Imposing symmetric cones is not sufficient, generally, to produce symmetric fields. It works in few cases (bunny), but it fails following the stationary line and it produces large distortions for most other models.*

be directly enforced, and "symmetry-unconstrained" mixed integer algorithm would not produce symmetric results. One exception is when the feature constraints of the MI are well distributed and symmetric, this drives the MI to construct a relatively symmetric field.

Figures 3.15 and 3.27 show our results compared with [BZK09]. In most cases, we observe a substantial improvement in visual quality: in the standing human model, note how our result manage to follow the line corresponding to sagittal plane even under a strong non-isometric deformation of the neck and head, as well as to preserve a horizontal field on the chest. The head symmetry is preserved equally well in the kneeling human model, as-well as the sagittal line on the torso. Similar behavior can be seen in the Busto and Igea models, where MI placed singularities asymmetrically at small details of the face, thus pushing the field away from respecting symmetry. On objects with an extrinsic global symmetry, like the lion and fertility models, plain MI yields satisfactory results, yet our method still improves the symmetry awareness of the resulting field. Lastly, in the Igea model, our algorithm preserves global symmetry while adapting to local non-symmetric features like the asymmetric broken piece from one cheek.

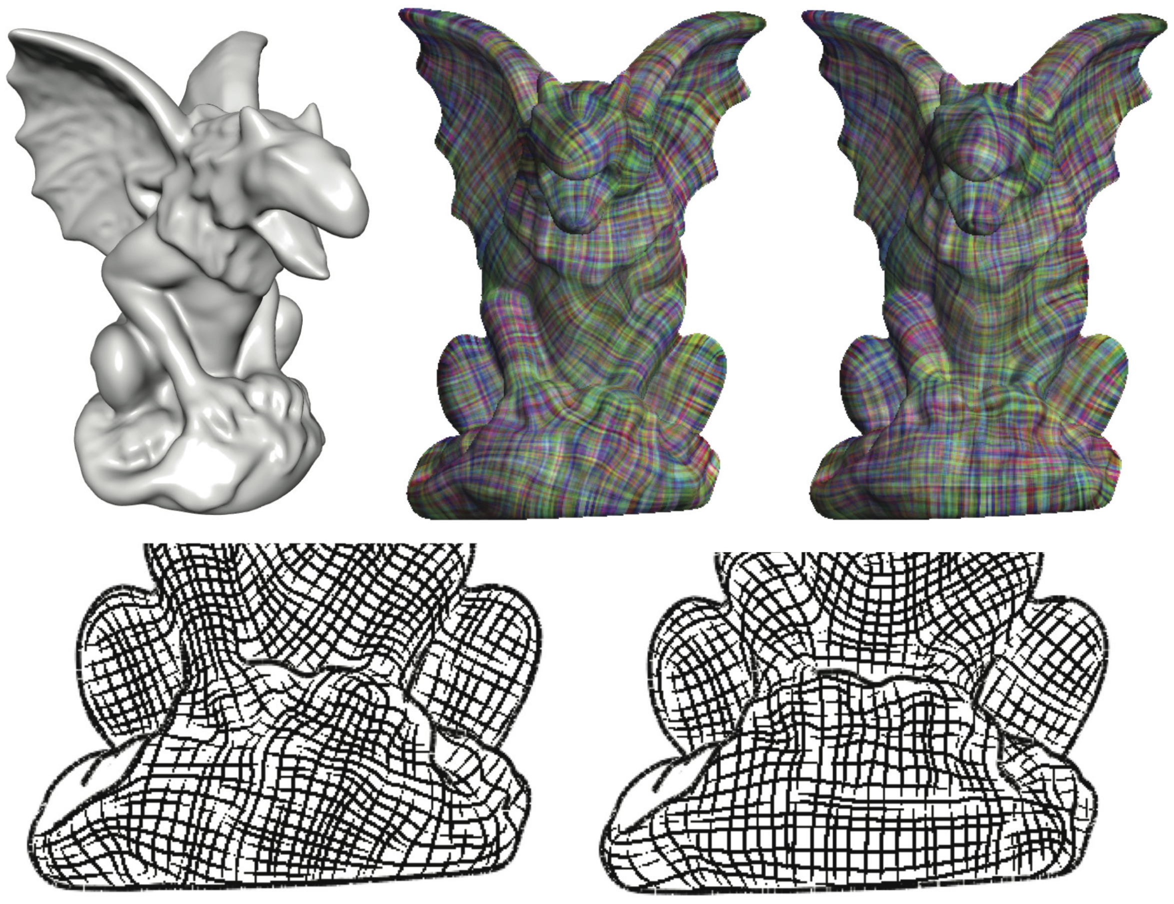


Figure 3.25: *A field symmetrized without (center) and with (right) stationary line constraints. The field in the middle/left image is overall symmetric but it fails following the stationary line.*

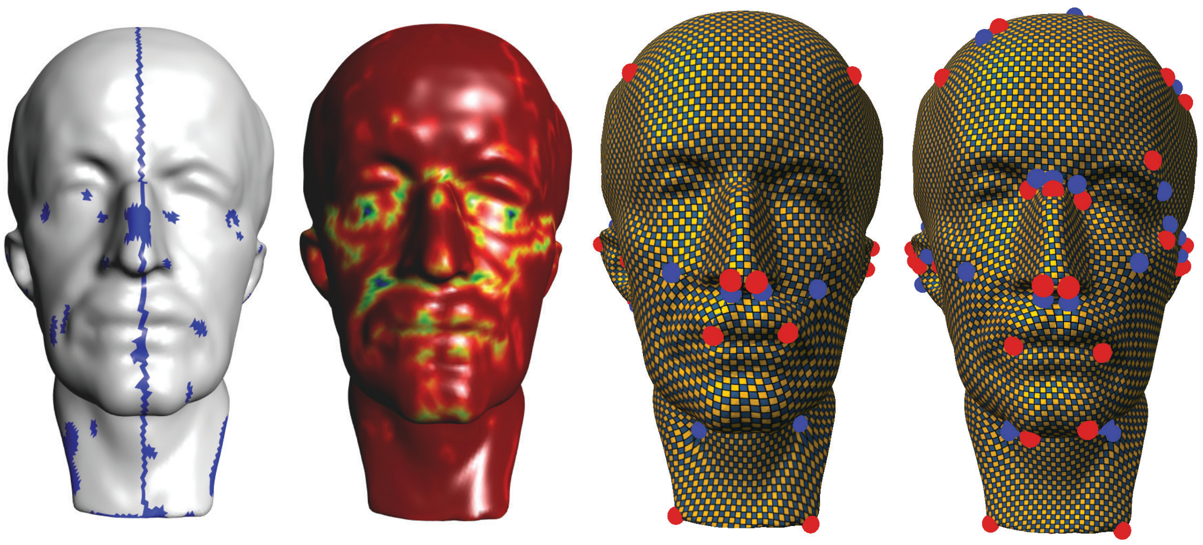


Figure 3.26: *Effect of the smoothness parameter. From left to right: curvature and symmetry constraints; smoothness of the symmetrized field and final results with* $\alpha = 0.1$ *and* $\alpha = 0.9$.

Model:	# faces	α	int/ext	# lmks	# cones
bimba	100,000	0.15	I	9	97
bumpy-cube	34,754	0.25	E	-	56
busto	50,930	0.30	I	8	28
egea	30,000	0.35	E	-	26
fertility	46,388	0.10	E	-	38
fish	48,592	0.10	I	5	24
gargoyle	54,000	0.25	E	-	94
holes3	28,800	0.15	E	-	32
knelt human	40,000	0.10	I	5	56
lion	66,696	0.25	E	-	66
Max Planck	54,000	0.20	E	.	20
ogre	52,306	0.30	I	10	238
rolling	20,000	0.12	E	-	27
stand. human	24,978	0.10	I	8	49
teddy	25,290	0.10	I	4	42

Table 3.2: *Statistics of datasets and results. We report: number of faces in the input; value of parameter* α *used; whether it has intrinsic or extrinsic symmetry; number of landmarks used for detecting intrinsic symmetry; number of cone singularities in the output.*

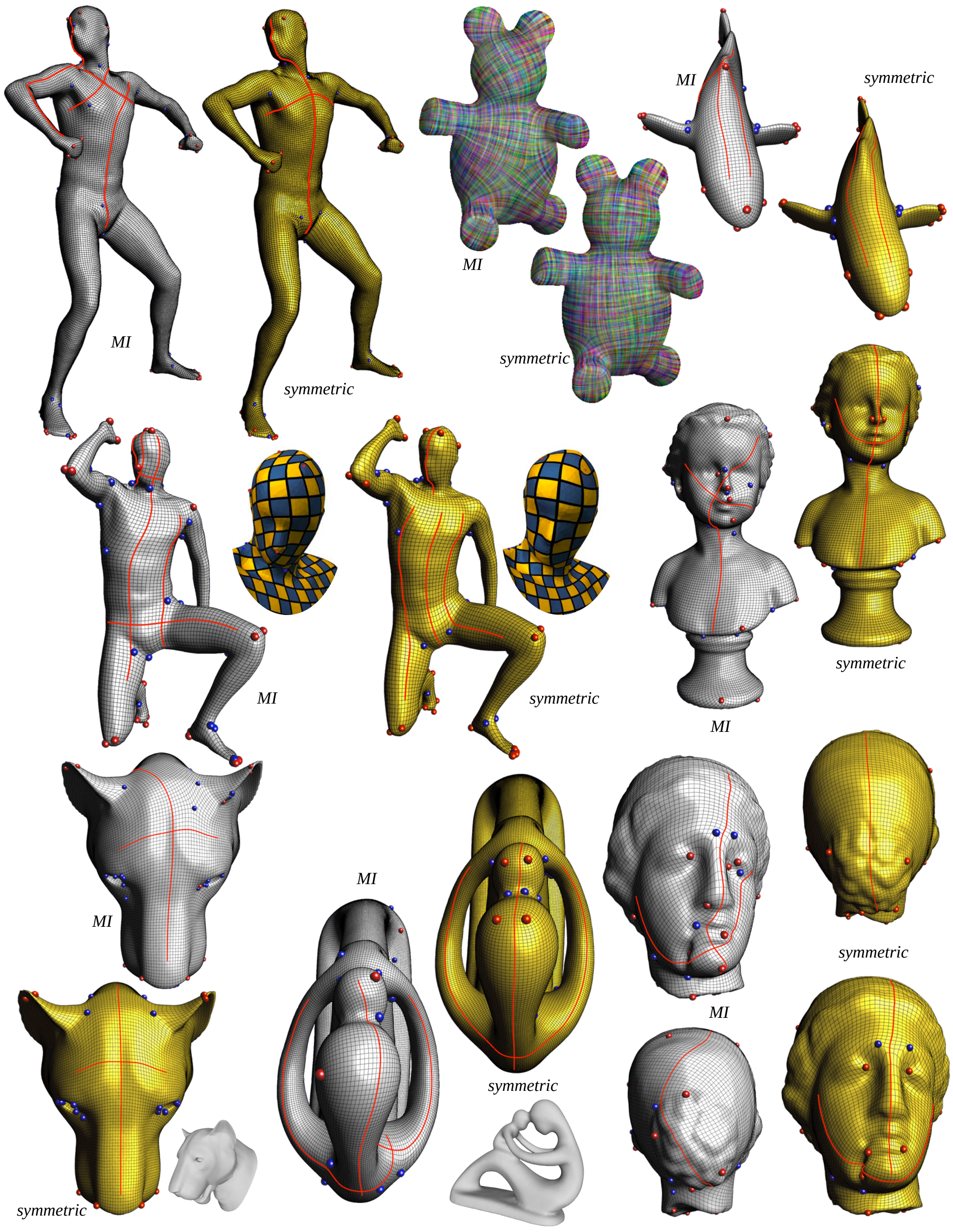


Figure 3.27: *Comparisons of our results with respect to plain MI.*

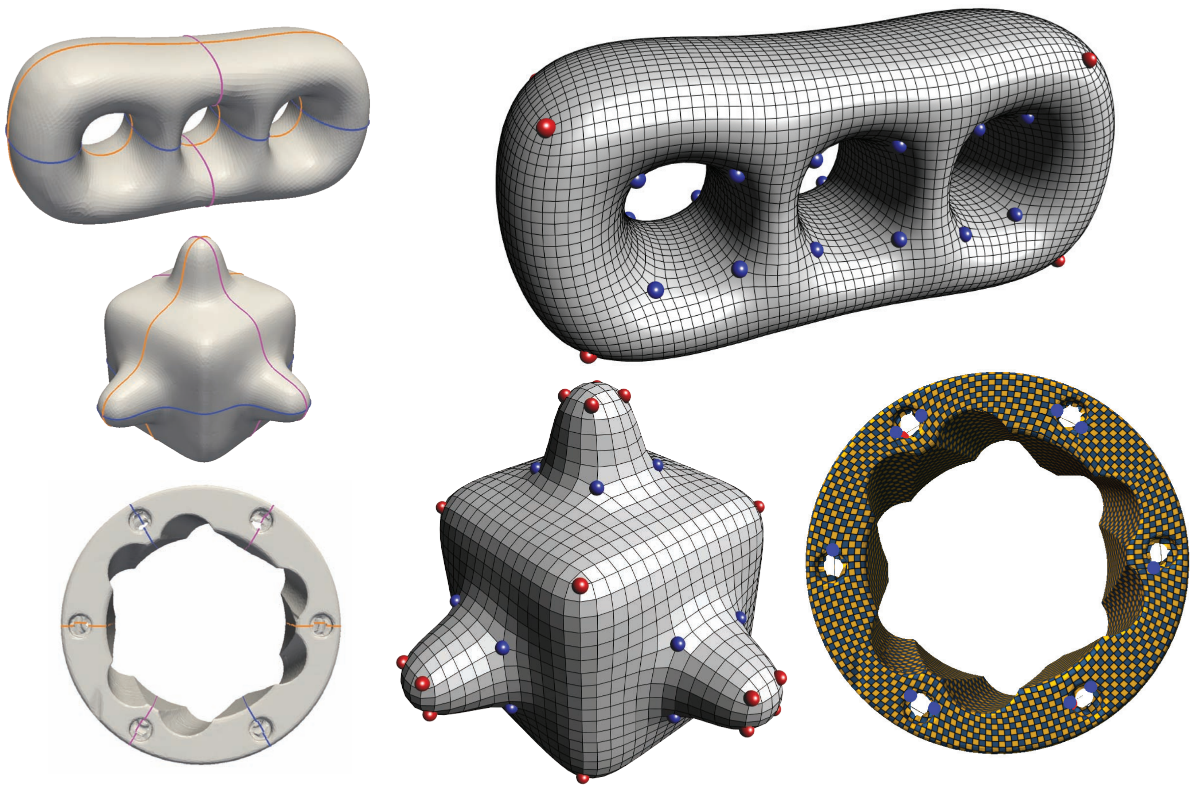


Figure 3.28: *Stationary lines and quadrangulations for a few objects with several bilateral symmetries.*

3.4 Axis-aligned, planar parametrization for content-aware image retargeting

Image retargeting resizes an input image to a given target resolution, where the aspect ratio changes. In order to avoid distorting the entire image by homogeneous scaling, or discarding important image parts by cropping, content-aware retargeting techniques were developed. These methods selectively deform the input image into the target dimensions according to a saliency map, preserving the shape of important image components while distorting unimportant background content. A few general methodologies for retargeting were proposed in the recent years, such as discrete carving/shifting, continuous warping and hybrid approaches [SS09]; some algorithms are even available in modern commercial image editing software [Ado10]. To help assess and further improve content-aware retargeting, a number of representative techniques were recently benchmarked and compared in a large-scale user study [RGSS10].

When analyzing the plethora of recent image retargeting approaches and their results, three prominent facts and challenges become apparent: (i) The quality of the resizing results depends immensely on the saliency map. Nearly every approach proposes its own variant of importance map computation, but it is disjoint from the content-aware resizing operator itself and can be easily exchanged. It is evident that a “silver-bullet” automatic saliency detection method does not exist, i.e., each saliency computation technique will fail on some input images, and user-guided importance specification is desired in such cases [SS09]. (ii) Fast or even realtime image resizing methods are extremely valuable, since they are more easily amendable to video retargeting and allow interactive control of the resizing process [KLHG09]. (iii) The robustness of the retargeting operator is another key factor that affects the quality of the results, namely the smoothness, predictability and avoidance of unwanted self-intersections (foldovers) in the resized image. Preventing foldovers appears to be a difficult task [CFK$^+$10], requiring complex, time consuming optimization that is not always guaranteed to be feasible and hence may not be robust.

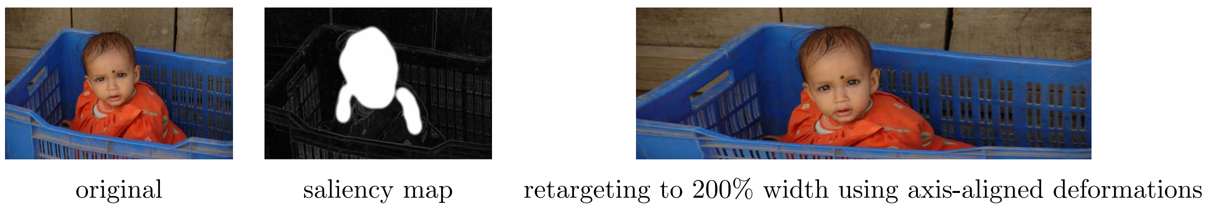


Figure 3.29: *Retargeting an image to wide format (200% the original width). The importance map was generated using a gradient filter and refined with a few strokes. The computation of the retargeted image took 4 milliseconds, and the overall process took 30 seconds of user time.*

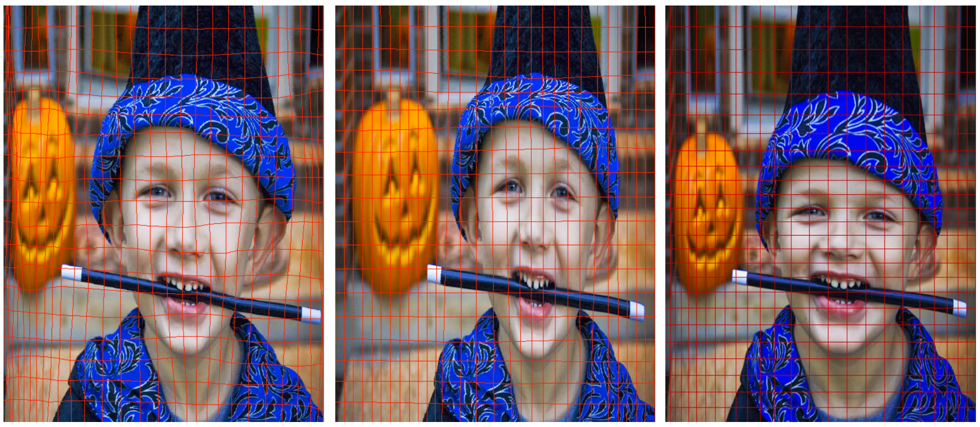


Figure 3.30: *Excluding local rotations leads the retargeting deformation to be* axis-aligned. *The* Child *dataset, retargeted to half the width using recent state-of-the-art techniques and our method. Observe that although previous approaches do not explicitly enforce it, their deformations are effectively almost axis-aligned.*

This work focuses on continuous (warp-based) retargeting, which readily allows controlling the smoothness of the retargeting operator. Discrete approaches such as [RSA08, SCSI08, BSFG09, PKVP09] are excellent at resizing and general editing of images especially rich in texture content, but they are known to have smoothness artifacts in some situations, and are typically somewhat slower than warping approaches.

It may seem that a high-quality, robust and foldover-free image retargeting by warping precludes realtime resizing and interactive adjustment of the saliency map due to the computational costs involved (in particular because prevention of self-intersection is not a linear constraint). The contribution of our work is to show that this does not have to be the case, if an appropriate space of image deformations is taken for the retargeting operator. Observing the behavior of the state-of-the-art warping methods, such as [KFG09, KLHG09, CFK$^+$10], we notice that they hardly introduce any local rotations in the image deformation. This indeed makes sense, since if the resizing operator contains local, varying rotations, they manifest themselves as "swirls", which are highly noticeable distortion artifacts. Lack of local rotation leads the retargeting deformation to be *axis-aligned*, i.e., the isoparametric lines remain straight and parallel after deformation, only changing the spacing between themselves (see Fig. 3.30). Our key observation is therefore, that *the space of axis-aligned deformations is the appropriate space for content-aware image retargeting.*

This observation has important consequences. First, the deformation space can be parameterized in 1D, since an axis-aligned deformation is determined solely by the intervals between the vertical and horizontal isoparametric lines. Previous retargeting methods parameterize the deformations in 2D, leading to optimization problems in the order of M$\times$N unknowns, where M, N are the vertical and horizontal input image resolution, whereas a 1D parameterization necessitates only $O(\mathrm{M}+\mathrm{N})$ unknowns. In addition, preventing foldovers, and more generally, controlling the minimal and maximal stretching of axis-aligned deformations is simple and robust, since this merely poses inequality constraints on the isoparametric line spacing. These constraints can always be posed in a feasible way, contrary to 2D constraints on lack of foldovers.

In this work, we show how to build a complete image retargeting system based on the axis-aligned deformation space. We demonstrate that several deformation energies which are meaningful for the content-aware retargeting application, such as the as-similar-as-possible and the as-rigid-as-possible energies, can be effectively optimized in this space while respecting foldover-free constraints. The resulting optimization problems can be cast as small quadratic programming (QP) problems thanks to the 1D parameterization, which allows for extremely efficient CPU-based computation using off-the-shelf QP solvers. Our image retargeting prototype runs in real time without requiring any precomputation, and enables interactive realtime resizing and editing of the saliency map. Our solution is robust, since we minimize convex energies under feasible constraints, guaranteeing the convergence of the solver and quality of the results. We test our method on the RetargetMe benchmark [RGSS10], showing that comparable and better results can be obtained within few milliseconds.

3.4.1 Related work on image retargeting

Warp-based content-aware retargeting methods define an energy functional that preserves the shape of salient image parts, and then minimize this functional given the boundary constraints of the target image size. The energy typically measures local deviation of the warp from a shape-preserving deformation such as translation [GSCO06], rigid transformation [KFG09] or similarity [WTSL08, KFG09, ZCHM09, KLHG09], weighted by the importance map. Additional energy terms are introduced to mitigate artifacts such as bending of straight lines [WTSL08, KLHG09, CFK$^+$10, LJW10] or edge blurring [KLHG09]. The energy is discretized over the 2D image domain, usually employing regular grids and finite differences (few works, e.g. [LJW10], use irregular triangle meshes). Earlier warping methods worked with quadratic energies which are highly efficient (only a sparse linear system needs to be solved, and its factorization can be precomputed and reused for arbitrary target image dimensions), but regrettably lead to artifacts, self-intersections and foldovers. Foldovers sometimes even result in "spills" of the retargeted grid outside of the image boundary; the spills are cropped, but the discontinuity in the result remains visible (see Fig. 3.31 and Figure 5 in [WTSL08]).

Later techniques moved away from the linear least-squares formulations and proposed nonlinear energies and/or inequality constraints to improve the retargeting quality. They

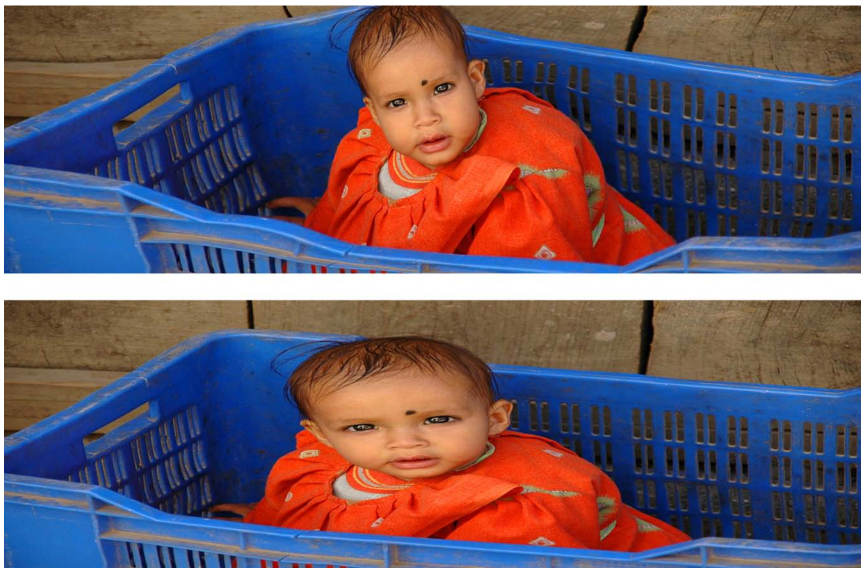


Figure 3.31: *Top: the image in Fig. 3.29 retargeted using [ZCHM09]. Bottom: the same image scaled homogeneously. The top image clearly shows a "spill-over" foldover near the child's head.*

prevent self-intersections by iteratively penalizing grid edge flipping [WTSL08], constraining the size of grid cells [KLHG09] or explicitly posing positive scaling constraints on grid cells' transformations [CFK$^+$10]. The method of Chen et al. [CFK$^+$10] offers formal guarantees on lack of self-intersections and vanishing grid cells, but their quadratic program is not guaranteed to have a feasible solution. Nonlinear constrained optimization significantly increases computation time, nearly excluding realtime usage; one notable exception is the streaming method of [KLHG09] that performs all computation on the GPU, achieving impressive framerates at pixel-level discretization. Heavy reliance on graphics hardware may hinder applicability to computationally modest platforms such as mobile devices.

It is interesting to note that, although not obliged to it by the formulation, recent techniques exhibit nearly axis-aligned deformations in practice, due to the inclusion of grid line bending penalties (formulated in e.g. [WTSL08, KLHG09]); Chen et al. [CFK$^+$10] even explicitly mention this behavior as an advantage. In this Section, we directly encode this behavior in the deformation space and demonstrate that it not only dramatically reduces the size of the optimization problem but also provides high quality retargeting results with guaranteed robustness.

For a detailed discussion of discrete retargeting approaches please refer to [RSA08, SCSI08, BSFG09, PKVP09] and the references therein. Discrete methods operate by removing or adding image pixels or patches to change the image's size and in general reshuffle its content (in a way they can be seen as generalizations of cropping or pasting). While these techniques do not guarantee smoothness and sometimes have discontinuity artifacts, they are very well suited for images with repetitive or stochastic content that should be removed or duplicated rather than squeezed or stretched. Combining the advantages of discrete and continuous retargeting methods is an exciting challenge; the MultiOp method [RSA09], albeit computationally expensive, was a successful attempt in terms of the achieved quality and user study ranking [RGSS10].

3.4.2 Algorithm

In this section, we show how to cast content-aware image resizing as a small quadratic program, solving in the space of axis-aligned deformations. The number of variables in our problem is linear in the size of the image boundary. The energy we minimize is convex, and finding the global minimum typically takes less than 4 ms.

Denote the input image width and height by W and H, respectively. Like most warp-based retargeting methods, we overlay a uniform grid over the image with N columns and M rows; the width of each column (and each cell) in the initial grid is then W/N and the height of each row is H/M. The task is to compute a new deformed grid for the resized image, with the desired overall width W' and height H'. In the continuous setting, an *axis-aligned deformation* can be fully described by the vertical and horizontal deformation derivatives along the boundary. In our discrete setting, we assume an axis-aligned deformation to be piecewise-linear (linear on each grid cell), such that it is fully determined by the widths of the deformed grid columns and the heights of the deformed grid rows.

Let $\mathbf{s}^{\text{rows}} = (s_1^{\text{rows}}, s_2^{\text{rows}}, \ldots, s_M^{\text{rows}})$ denote the unknown heights of the rows and $\mathbf{s}^{\text{cols}} = (s_1^{\text{cols}}, \ldots, s_N^{\text{cols}})$ the unknown widths of the columns. The axis-aligned deformation is therefore represented by the vector of unknowns $\mathbf{s} = (\mathbf{s}^{\text{rows}}, \mathbf{s}^{\text{cols}})^T \in \mathbb{R}^{M+N}$. We now give the general form of the optimization that computes the deformed retargeted image grid:

$$\begin{aligned} \text{minimize} \quad & \mathbf{s}^T Q \mathbf{s} + \mathbf{s}^T \mathbf{b} && (3.9)\\ \text{subject to} \quad & s_i^{\text{rows}} \geq L^h, \quad i = 1, \ldots, M, && (3.10)\\ & s_j^{\text{cols}} \geq L^w, \quad j = 1, \ldots, N, && (3.11)\\ & s_1^{\text{rows}} + \ldots + s_M^{\text{rows}} = H', && (3.12)\\ & s_1^{\text{cols}} + \ldots + s_N^{\text{cols}} = W'. && (3.13) \end{aligned}$$

The matrix $Q \in \mathbb{R}^{(M+N)\times(M+N)}$ and the vector $\mathbf{b} \in \mathbb{R}^{M+N}$ are determined based on the energy we want to minimize (see Sec. 3.4.2.1), and $L^h, L^w > 0$ are the minimum sizes allowed for rows and columns of the deformed grid, respectively. The inequalities (3.10) and (3.11) *guarantee* that our deformation is free from foldovers, since every cell on the grid cannot be smaller than the specified dimensions L^h-by-L^w and cannot invert (inside each cell the deformation is linear and therefore foldover-free). Equations (3.12) and (3.13) fix the total dimensions of the deformed grid to the desired target size.

For the above quadratic program (QP) to be feasible, we just need $L^h \leq H'/M$ and $L^w \leq W'/N$; simple homogeneous scaling then provides a feasible solution. The feasible domain is bounded, since $\forall i, 0 \leq s_i \leq \max\{H', W'\}$, such that the objective function in (3.9) is finite in the feasible region. The energy should be defined in such a way that Q is positive (semi)definite; our problem is then convex and can be solved with standard QP solvers.

3.4.2.1 Energy functions

Image retargeting methods rely on a saliency map $\omega(x, y)$ that assigns an importance value between 0 and 1 to every pixel of the image. Our goal is to compute a deformation that preserves the image in the salient zones as much as possible and concentrates the unavoidable distortion in less important areas. To integrate the saliency map ω in our formulation, we average its values inside every cell of the grid on the original image and we obtain the saliency matrix $\Omega \in \mathbb{R}^{M \times N}$. This per-cell integration of saliency is the proper FEM discretization in our piecewise-linear setting.

We consider two energies in our framework, often employed by prior successful retargeting methods: (1) the As-Similar-As-Possible (ASAP) energy [ZCHM09], which produces deformations that are locally close to similarities, and (2) the As-Rigid-As-Possible (ARAP) energy [KFG09], which penalizes all local deformations except translation and rotation. Fig. 3.32 shows an example result with these energies.

ASAP Energy. In the space of axis-aligned deformations, a similarity transformation is a combination of uniform scaling and translation, since rotations are not present in the deformation space. The ASAP energy thus minimizes non-uniform scaling:

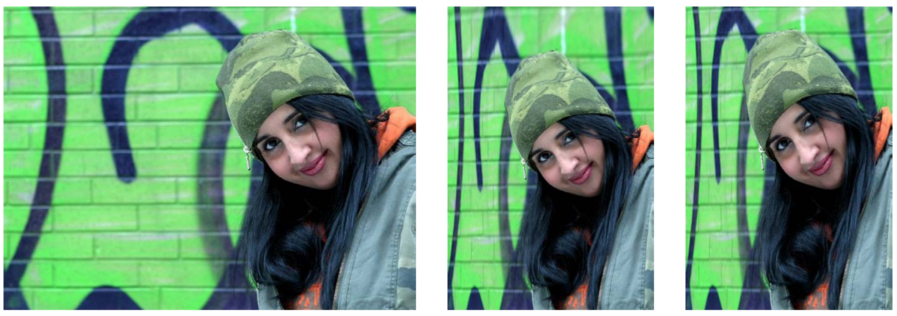


Figure 3.32: *The* Fatem *image resized with ASAP and ARAP energies.*

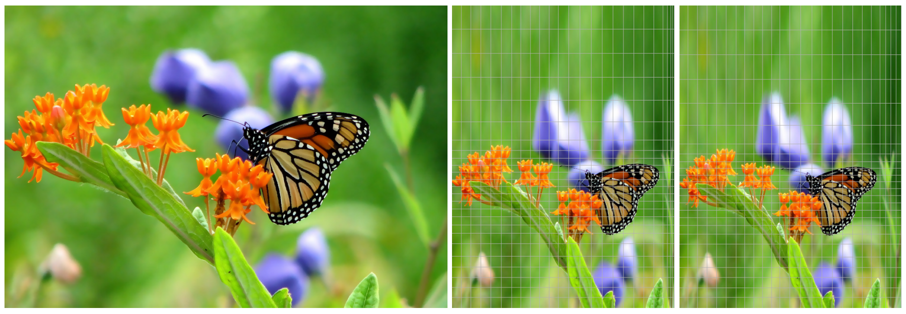


Figure 3.33: *The minimal column width L^w and row height L^h can be prescribed. In the middle image, each column may not be compressed to more that 10% of the original width, and on the right the minimal width is 30%, such that the deformation is less pronounced, since extreme squeezing is disallowed.*

$$E_{\text{ASAP}} = \sum_{i=1}^{M} \sum_{j=1}^{N} \left(\Omega_{i,j} \left(\tfrac{M}{H}\, s_i^{\text{rows}} - \tfrac{N}{W}\, s_j^{\text{cols}}\right)\right)^2 . \tag{3.14}$$

The two factors M/H and N/W compensate for the aspect ratio of the cells in the original grid.

To minimize this energy using our QP framework, we define the following matrix $K \in \mathbb{R}^{(MN)\times(M+N)}$:

$$K_{k,l} = \begin{cases} \Omega_{r(k),c(k)}\, \frac{M}{H} & \text{if } l = r(k), \\ -\Omega_{r(k),c(k)}\, \frac{N}{W} & \text{if } l = M + c(k), \\ 0 & \text{otherwise,} \end{cases} \tag{3.15}$$

where $r(k) = \lceil k/N \rceil$ and $c(k) = ((k-1) \bmod N) + 1$. From this equation, $K\mathbf{s}$ gives us the vector with energy terms per row, and $E_{\text{ASAP}} = \mathbf{s}^T K^T K \mathbf{s}$. Using the generic notation of Eq. (3.9), $Q = K^T K$ and $\mathbf{b} = 0$. Clearly, Q is a positive semidefinite matrix in this case, such that the energy is convex.

ARAP Energy. In our axis-aligned deformation space, a rigid transformation is reduced to a translation, since rotations are not allowed by definition. The ARAP energy thus minimizes uniform and non-uniform scaling:

$$E_{\text{ARAP}} = \sum_{i=1}^{M} \sum_{j=1}^{N} \left(\Omega_{i,j} (\tfrac{M}{H}\, s_i^{\text{rows}} - 1)\right)^2 + \left(\Omega_{i,j} (\tfrac{N}{W}\, s_j^{\text{cols}} - 1)\right)^2 . \tag{3.16}$$

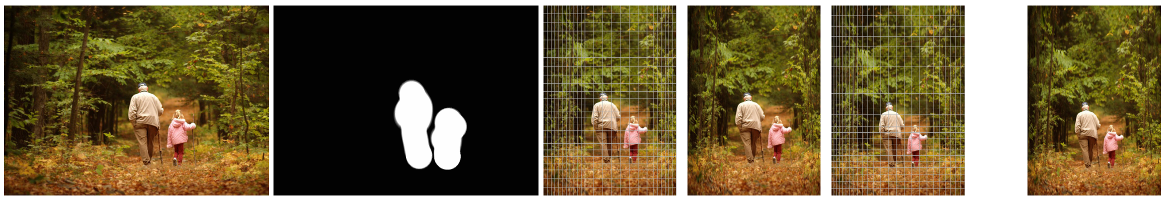


Figure 3.34: *The effect of the Laplacian regularization. The image on the left is retargeted to 50% width using a manually-painted saliency map. As seen in the middle, retargeting using unregularized ASAP energy leads to strong variation in column width. This effect can be mitigated by adding a weighted Laplacian regularization term, as shown on the right.*

To minimize this energy using our QP framework, we define the following two matrices $R^{\text{top}}, R^{\text{btm}} \in \mathbb{R}^{(MN)\times(M+N)}$:

$$R^{\text{top}}_{k,l} = \begin{cases} \Omega_{r(k),c(k)} \frac{M}{H} & \text{if } l = r(k) \\ 0 & \text{otherwise,} \end{cases} \tag{3.17}$$

$$R^{\text{btm}}_{k,l} = \begin{cases} \Omega_{r(k),c(k)} \frac{N}{W} & \text{if } l = M + c(k) \\ 0 & \text{otherwise,} \end{cases} \tag{3.18}$$

where $r(k) = \lceil k/N \rceil$ and $c(k) = ((k-1) \bmod N) + 1$. We also define the vector $\mathbf{v} \in \mathbb{R}^{MN}$, $v_k = \Omega_{r(k),c(k)}$.

We can now rewrite the ARAP energy using matrix notation:

$$E_{\text{ARAP}} = \left(\begin{bmatrix} R^{\text{top}} \\ R^{\text{btm}} \end{bmatrix} \mathbf{s} - \begin{bmatrix} \mathbf{v} \\ \mathbf{v} \end{bmatrix} \right)^T \left(\begin{bmatrix} R^{\text{top}} \\ R^{\text{btm}} \end{bmatrix} \mathbf{s} - \begin{bmatrix} \mathbf{v} \\ \mathbf{v} \end{bmatrix} \right). \tag{3.19}$$

In the generic notation of Eq. (3.9):

$$Q = \begin{bmatrix} R^{\text{top}} \\ R^{\text{btm}} \end{bmatrix}^T \begin{bmatrix} R^{\text{top}} \\ R^{\text{btm}} \end{bmatrix}, \quad \mathbf{b} = -2 \begin{bmatrix} \mathbf{v} \\ \mathbf{v} \end{bmatrix}^T \begin{bmatrix} R^{\text{top}} \\ R^{\text{btm}} \end{bmatrix}. \tag{3.20}$$

Again, the form of the Q matrix clearly indicates that it is positive semidefinite, such that the ARAP energy is convex.

Note that even though the intermediate matrices K, R^{top}, R^{btm} have MN rows, they are extremely sparse and fast to construct procedurally. The resulting Q matrices for the QP are square with $M + N$ rows/columns, meaning they are dense but small. Note also that other energies can be similarly formulated in our space of axis-aligned deformations; we have chosen to concentrate on the above two since they are commonly used and typically provide good results. Below we show an additional energy that can be added to serve as a regularizing (smoothing) term.

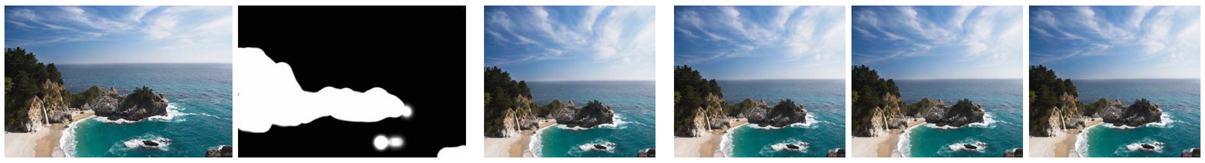


Figure 3.35: *The grid resolution does not have a dramatic effect on the energy minimization result (here, the ASAP energy was used).*

3.4.3 Laplacian regularization

We can enrich the energies shown above with a regularization energy that allows to increase the smoothness of the resulting deformation. Note that the deformation we compute is always guaranteed to be free of collapsing artifacts. The Laplacian regularization allows to distribute the deformation more evenly across the image, and is particularly useful for manually painted saliency maps, since they tend to concentrate the saliency on distinct parts of the image and fall off abruptly to zero elsewhere (i.e., such saliency maps are highly non-smooth). See Fig. 3.34 for an example of the effect of the Laplacian regularization.

The Laplacian regularization term is defined as follows:

$$E_{\text{reg}} = \sum_{i=1}^{M-1} \left(\tfrac{M}{H}(s_{i+1}^{\text{rows}} - s_i^{\text{rows}})\right)^2 + \sum_{j=1}^{N-1} \left(\tfrac{N}{W}(s_{j+1}^{\text{cols}} - s_j^{\text{cols}})\right)^2. \tag{3.21}$$

The regularization penalizes two adjacent rows or columns that have large differences in size, that is, we wish to minimize the Laplacian of the parametrization. Note that the deformation that minimizes the Laplacian is homogeneous scaling, such that this regularization term can be seen as a way to blend between homogeneous resizing and the ASAP or ARAP deformation, controlled by a weighting factor $w_{\text{reg}} \geq 0$.

To incorporate the regularization term into the QP framework (3.9), we simply add the term $\mathbf{s}^T (w_{\text{reg}} L)\, \mathbf{s}$ to the energy, where L is the standard Laplacian matrix corresponding to Eq. (3.21). In other words, we add the matrix $w_{\text{reg}} L$ to Q in Eq. (3.9). As well-known, the Laplacian matrix is positive semidefinite, such that this energy term does not hurt the convexity of the problem.

3.4.4 Cubic B-spline interpolation

The formulations of the ASAP and the ARAP energies, as well as the Laplacian regularization term, are proper linear FEM approximations of the continuous counterparts, such that convergence is expected under uniform grid refinement. We have observed that the results of the optimization are not greatly dependent on the grid resolution (see Fig. 3.35); this

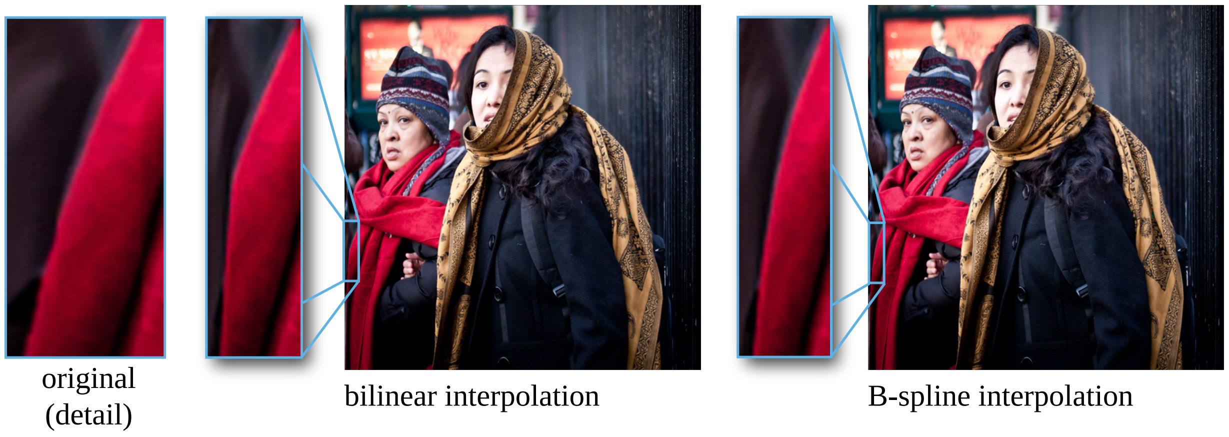


Figure 3.36: *Images of high resolution may benefit from higher-order interpolation when using coarse grids for the retargeting optimization. Here we show an example where the input image resolution is 2800×1800 and the retargeting uses a 25×25 grid. Bilinear interpolation on this grid leads to some smoothness artifacts (middle), while upsampling to a 100×100 grid using the spline technique described in Sec. 3.4.4 results in visually smooth interpolation.*

makes sense also because our constrained deformation space, parameterized in 1D, does not admit huge local variation. A coarse grid resolution of 25×25 (i.e, 50 optimization variables) suffices in most cases to faithfully describe the deformation map. However, such a coarse bilinear grid may be insufficient to provide high-quality results for images with very high resolution, because the bilinear interpolation is not smooth across grid lines.

To improve the interpolation results for high-resolution images, we can optionally employ B-spline interpolation to upsample the retargeted grid. We define a uniform cubic B-spline using the deformed grid vertices as control points. Note that this can be performed in 1D, i.e., we use two 1D cubic B-splines (one for rows and one for columns). We sample the splines for denser horizontal and vertical positions in order to produce a new grid of arbitrary resolution. We create the final retargeted image by bilinear interpolation on this finer grid. This more expensive bilinear interpolation can be performed using the GPU.

The deformation described by our deformed grid is guaranteed to always be a bijection and, thanks to the variation diminishing property, also the finer grid obtained with the spline is guaranteed to be foldover-free. An example of the improvement is given in Fig. 3.36.

3.4.5 Results

We implemented an interactive application for image retargeting that allows in realtime to change the image size, adjust the saliency, and tweak the other two parameters of the

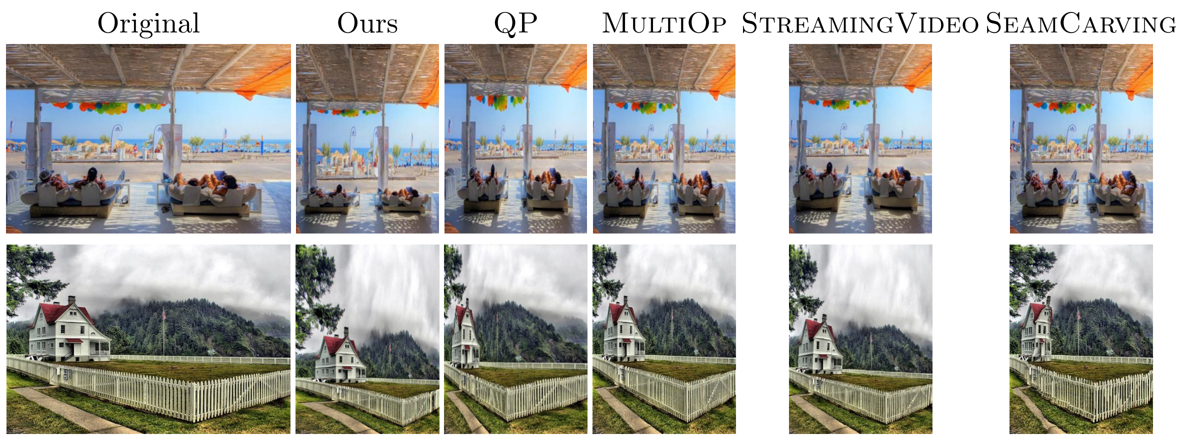


Figure 3.37: *Comparisons with recent image retargeting methods:* QP *[CFK+10],* MultiOp *[RSA09],* StreamingVideo *[KLHG09] and* SeamCarving *[RSA08]. We provide the complete comparison on the RetargetMe benchmark [RGSS10] in the supplemental material.*

optimization (the strength of Laplacian regularization and minimal grid cell size). We provide a full comparison with the RetargetMe benchmark [RGSS10] in the supplemental material. The accompanying video shows various recordings of interactive retargeting sessions. The binary of our prototype for Windows and MacOS 10.6 is provided in the supplemental material as well.

We ran our experiments on a Core2Duo at 2.4 GHz, using a single core. OpenGL was used to compute the bilinear interpolation; our test system is equipped with a low-end laptop graphics card (the NVIDIA 320M). We employed the easy-to-use and fast QP solver called CVXGen [MB10] to solve the QP in Eq. (3.9). We obtained interactive frame rates (always above 60 fps) in all our experiments. The average computation time to retarget an image using a 25×25 grid was 4 milliseconds.

As can be seen in Fig. 3.37 and in the supplemental material, our technique provides high-quality results that are comparable, if not better than the state-of-the-art methods. We observed that the ASAP energy usually produces slightly better results than ARAP, probably because of the additional flexibility of uniform scaling. Contrary to some previous approaches, our method always produces smooth, intersection-free images that preserve the salient features well, and tend to respect the input image structure in general. The experimental evidence supports the space of axis-aligned deformations as a useful and sufficiently rich space for image retargeting.

Saliency maps. Our approach can be successfully applied in a fully-automatic mode. We experimented with the automatic saliency detection method of [IKN98], although of

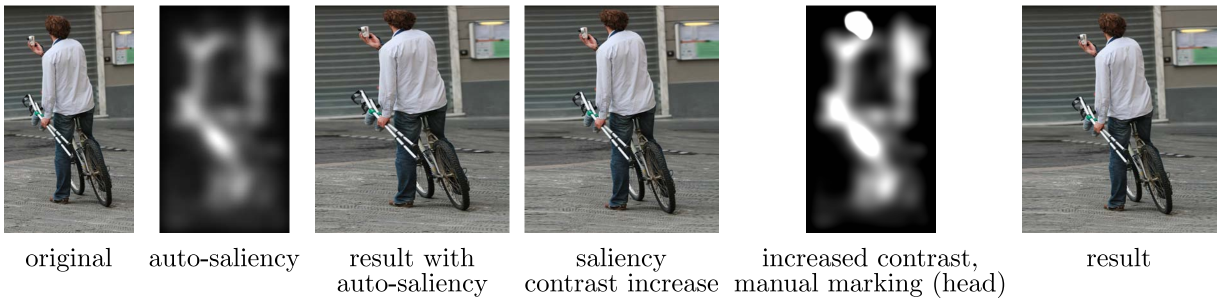

original
auto-saliency
result with auto-saliency
saliency contrast increase
increased contrast, manual marking (head)
result

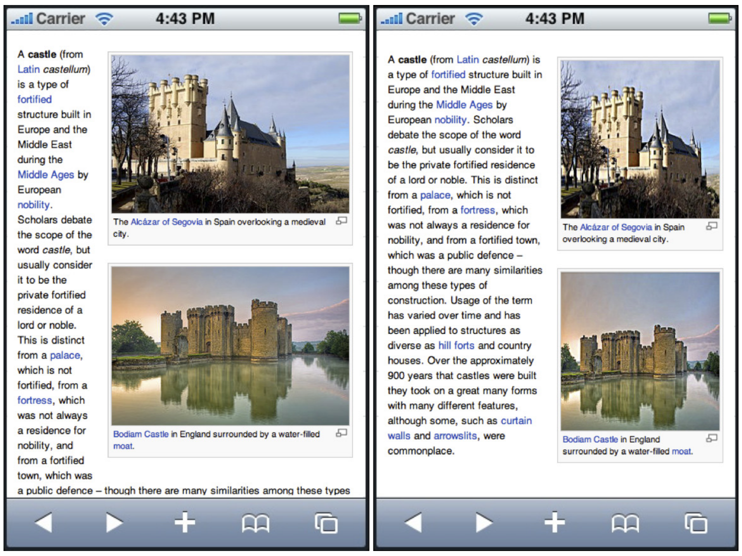
Figure 3.38: *The original image is retargeted to 150% width, first using an automatic saliency map of [IKN98]. The result is not perfect since the head has not been detected as a salient object. Increasing the saliency contrast improves the result (4th image), and a single stroke on the head produces even further improvement (right).*

course any other method can be used. The main advantages of our method, the lack of local rotations and foldovers, are independent of the importance map choice. As discussed earlier, automatic saliency may fail for certain images due to the subjectiveness and ill-posedness of the problem. In such cases, a minimal amount of user intervention can improve the results; fine-tuning by the user is readily enabled by the realtime speed of our approach.

Fig. 3.38 shows the results computed using the ASAP energy with three different saliency maps. The first image uses an automatic saliency map; the person's head is not detected as salient and is distorted. Our application allows to interactively change the contrast of the saliency map to improve the result (Fig. 3.38, middle), and to simply paint over it. A single user stroke on the head is sufficient to greatly improve the outcome (Fig. 3.38, right). This example took 15 seconds of user time. On average, in our experiments we spent 30 seconds per image to paint the saliency map and adjust the parameters. The process is very intuitive, since the application allows to watch the result of any manipulation in real time.

Interactive, user-controllable parameters. Our method exposes a small numbers of parameter to the user to allow a high degree of control over the final result. Please refer to the video and the executable for demonstration of the realtime tuning effect. The size of image can of course be changed interactively. The saliency map can be controlled by adjusting its brightness and contrast, and by manual painting (Fig. 3.38). The deformation can be controlled by changing the Laplacian regularization weight w_{reg} (Fig. 3.34). Finally, the minimal grid cell width and height can be selected (Fig. 3.33). We have found that saliency adjustment is the most useful control mechanism; the other parameters were usually left at their default values ($w_{\text{reg}} = 0.5$, $L^w = L^h = 20\%$).

Integration in a web browser. Optimizing website layouts for different screen resolutions is a difficult task. To obtain good results in terms of usability and presentation, it is often necessary to design customized views for every aspect ratio. Text can be easily rearranged to fit a window of any size, but images are only scaled homogeneously, limiting the layout optimization algorithm used in mobile web browsers and potentially leading to sub-optimal results. Image retargeting allows to change the aspect ratio of a picture, increasing the quality of the final layout and saving space.



A **castle** (from Latin *castellum*) is a type of fortified structure built in Europe and the Middle East during the Middle Ages by European nobility. Scholars debate the scope of the word *castle*, but usually consider it to be the private fortified residence of a lord or noble. This is distinct from a palace, which is not fortified, from a fortress, which was not always a residence for nobility, and from a fortified town, which was a public defence – though there are many similarities among these types

The Alcázar of Segovia in Spain overlooking a medieval city.

Bodiam Castle in England surrounded by a water-filled moat.

A **castle** (from Latin *castellum*) is a type of fortified structure built in Europe and the Middle East during the Middle Ages by European nobility. Scholars debate the scope of the word *castle*, but usually consider it to be the private fortified residence of a lord or noble. This is distinct from a palace, which is not fortified, from a fortress, which was not always a residence for nobility, and from a fortified town, which was a public defence – though there are many similarities among these types of construction. Usage of the term has varied over time and has been applied to structures as diverse as hill forts and country houses. Over the approximately 900 years that castles were built they took on a great many forms with many different features, although some, such as curtain walls and arrowslits, were commonplace.

The Alcázar of Segovia in Spain overlooking a medieval city.

Bodiam Castle in England surrounded by a water-filled moat.

Figure 3.39: *The "Castle" page on wikipedia. On the left, the original version rendered using a screen width of 480px. On the right, the same page with the images retargeted by our algorithm.*

To incorporate a retargeting system in a web browser, we should not only consider the retargeting quality but also its efficiency and space overhead, due to energy consumption and bandwidth limitations. Contrary to other methods, our algorithm can be easily integrated in a web browser with negligible time and space overhead. We see two possible ways of extending any image format to store the information needed to retarget it (assuming a 25×25 grid, which suffices for high quality retargeting results up to full HD resolution):

(i) Storing the integrated saliency map. Our optimization procedure only requires the saliency matrix Ω that can be stored in 625 bytes if we quantize every matrix entry to one byte. We can then retarget to arbitrary aspect ratio using the mobile CPU.

(ii) Storing precomputed aspect-ratios. An axis-aligned grid is parametrized by 50 floats, since only the boundary has to be encoded. We can efficiently store multiple grids inside an image with a very small space overhead. encoding a grid requires 200 bytes, so that a set of 10 retargeted grids uses less than 2 KB. In this setting there is no computational overhead

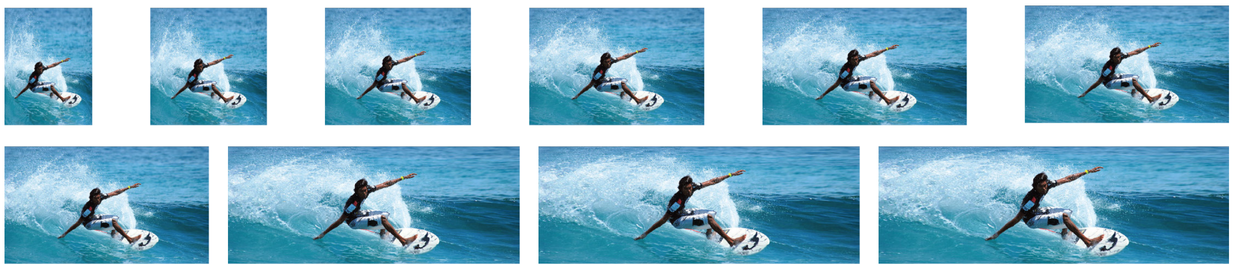


Figure 3.40: *10 resized grids are stored in an image to allow real-time retargeting without any computation. Since our grids are axis-aligned, only 2kbytes are needed to store them all. The preprocessing time required to produce the 10 grids is 38ms.*

for the browser, as it only needs to select the desired grid for bilinear interpolation to map the image onto the screen. An example of the same image retargeted using the proposed 10 aspect ratios is shown in Figure 3.40. It is also possible to linearly interpolate two grids to obtain any intermediate aspect ratio; in our experiments this is very close to the exact retargeting result.

User study. We conducted a user study with 305 participants, following the protocol of [RGSS10]. Eight methods have been compared: manual crop (CR), nonhomogeneous warping (WARP) [WGCO07], Scale-and-Stretch (SNS) [WTSL08], MULTIOP [RSA09], shift-maps (SM) [PKVP09], streaming video (SV) [KLHG09], energy-based deformation (LG) [KFG09] and our algorithm (AA). All datasets in the study have been created by the authors of the respective methods, manually tweaking parameter values and sometimes the saliency to show the strengths of the retargeting algorithm and produce the best possible result. We note that the study participants had no reason to prefer a retargeted image over a (manually) cropped one since the study did not place the images in any semantic context. This biases the study in favor of manual cropping as it does not introduce any distortion. For this reason, cropping should be considered as a reference, not as a proper retargeting algorithm (for more details see the original paper [RGSS10]).

The study statistics are provided in Appendix B. Fig. 3.41 provides a short summary that shows that our deformation subspace is a good choice for content-aware retargeting. Our results have been considered superior with respect to six state-of-the-art methods and achieved a quality statistically indistinguishable to SV [KLHG09], while being simpler to implement, faster and not requiring a GPU implementation to obtain interactive frame rates. Our findings are in accordance with the original study [RGSS10], providing further validation of the consistency in the users' preferences.

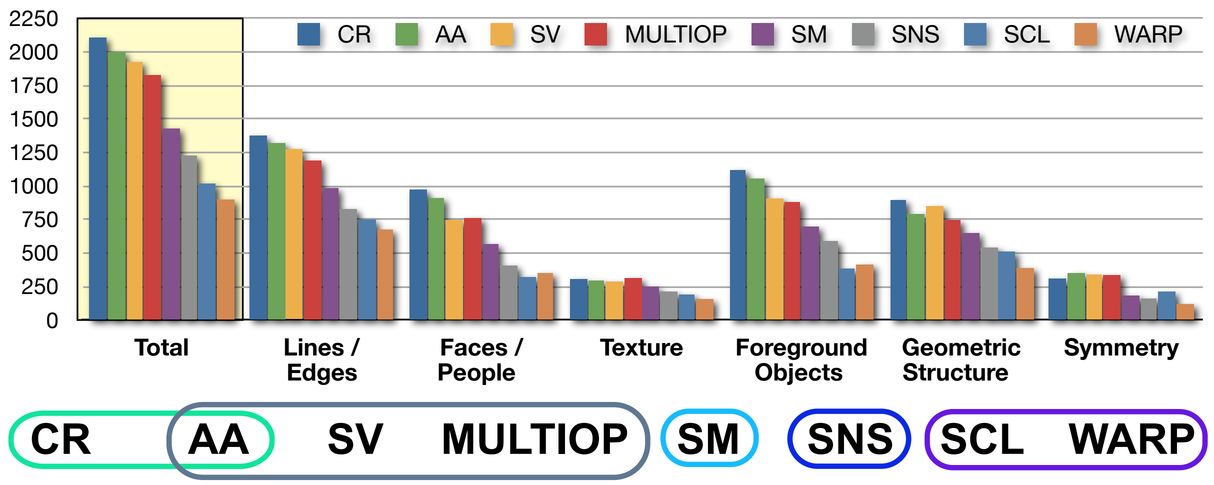


Figure 3.41: *The number of votes for the 8 methods considered in our user-study for each image attribute. In the bottom, the operators within a group are statistically indistinguishable in terms of user preference. Our method ranks higher than others and it is statistically indistinguishable from CR.*

3.5 Concluding remarks

We have proposed a method for producing quad-based domains for global mesh parametrization, which improves domain simplicity, while maintaining alignment to an input cross field. The method has been implemented to take in input a cross field induced by a quad mesh - which may be already the base domain of a parametrization - and it produces a parametrization having an abstract complex of axis-aligned rectangles as base domain. Our results exhibit simpler domains than other proposals at the state-of-the-art and parametrizations are directly suitable for most applications.

The method consists of two main ingredients: an algorithm for simplifying the topology of the cross field, and an algorithm for smoothing parametrization across abstract quad domains. Each such ingredient can be used independently in different contexts. The algorithm for field topology simplification is rather general, it could be applied to any N-symmetry field [RVLL08], provided that an initial graph representing the topology of the field is given in input. However, finding such a graph in general, e.g., from a field defined on a triangle mesh, is still an open problem : this is the reason why we have implemented our algorithm to work just on a field induced by a quad mesh.

Our algorithm naturally preserves alignment with the input cross field, which is taken into account during simplification. The unique parameter used in the whole method (value k in the energy) allows us to trade off between topology simplification and faithfulness to the input field. Our method also allows preserving sharp creases, either through hard constraints during graph simplification, or through a snapping mechanism during smoothing.

The main limitation of the proposed approach is clearly its reliance on a pre-existing cross field $\mathcal{C}$. (for example discretized in the form of a semi-regular quad mesh). The simplified version of a graph from a field with poorly placed, or too numerous singularities will be far too complicated to be useful in most contexts. However, in our experience, practically any graph is strongly improved, unless it is already optimal in terms of simplicity of the domain.

Another limitation consists in the occasional need of a few mismatch-sized adjacent rectangles in the domain D, which adds a scaling to the transition function across the corresponding cut, against our objective of domain simplicity. This limitation is somehow the natural drawback of having a domain made of only few patches to represent relatively complex shapes. intrinsic to this kind of domain, but we believe that its other merits compensates this minor shortcoming. The choice to preserve the irregular points, which is crucial to preserve quality of good input field, backfires.

We introduced a new paradigm for symmetry-aware field design on surface. Our key idea is to incorporate symmetry averaging of field values over symmetry orbits with existing Mixed-Integer field generation techniques. This required few intermediate steps: 1) introducing general reflections, not restricted to isometric reflections, and developing theory for field transport and symmetric averaging; 2) computational methods to compute symmetric map, transport and stationary sets on shapes deviating from perfect isometry; and 3) incorporating the symmetric averaging operator into the Mixed-Integer framework. Based on experiments on a variety of models, we believe the introduced algorithm can significantly improve visual aesthetics and symmetry-awareness of N-fields on models.

A limitation of our method, is that we cannot deal with intrinsic symmetry for higher genus objects. This is definitely one future research direction. Furthermore, we plan to explore more applications of symmetric field design in geometry processing. Another interesting research direction is to use the new transport operators on other tensors from geometry processing such as the curvature tensor.

Finally, we presented an image retargeting method that is based on axis-aligned planar parametrization. The space of axis-aligned maps appears to be suitable for the problem at hand, and has multiple advantages, such as robustness and guaranteed lack of foldovers, smoothness, and realtime performance. The general approach of controlling a deformation energy by the domain boundary falls into the category of boundary element methods and allows for very efficient solutions in cases like ours.

Axis-aligned deformations certainly have less freedom than general variational warps. We argue that in most cases, localized rotations are bad for image retargeting, because they lead to swirling or significant shearing. However, it is conceivable that in certain situations extreme shearing is preferable to axis-aligned scaling (for instance when the image background has completely uniform color, so that its shearing will not be visible). Since we completely exclude rotations from our warps, our method will not be capable of reproducing such effects.

We are interested in extending our method to video retargeting in future work, as its speed and absence of precomputation overhead would enable online (streaming) execution. Video retargeting is very challenging due to the additional temporal coherence requirements. Our technique can be potentially generalized to video by making the warps “track” salient moving objects via deformations that are consistent with the optical flow. Similarly to the Crop-and-Warp method of [WLSL10], we can incorporate cropping into the framework by allowing the size of rows and columns near the boundary to vanish.

Chapter 4

Implicit Hierarchical Meshes

Subdivision surfaces are becoming more and more popular in computer graphics and CAD. During the last decade, they found major applications in the entertainment industry, especially in the production of movies, videogames [DKT98] and in simulation [PAH06]. Several solid modelers, both commercial and open source, now support modeling based on subdivision [Ble, May, Mod, Sil]. From the point of view of users, subdivision surfaces come midway between polygonal meshes and NURBS, getting many advantages from both worlds. Being based on the recursive refinement of the faces of a polygonal mesh, they allow a designer to model a shape on the basis of a relatively simple control net, which can be handled as freely as a polygonal mesh, while automatically generating either a finer mesh at the desired level of detail, or a smooth limit surface.

The natural approach to modeling based on subdivision is coarse-to-fine. This can be a disadvantage with respect to polygonal modeling, since, e.g., it prevents exploiting the wealth of high resolution meshes that are generated through model acquisition. However, advances in reverse subdivision suggest that subdivision modeling may also be used fine-to-coarse [LMH00, Sab04, SB99, SMAB02, PPT$^+$11]. See Chapter 2 for a complete discussion and for a novel proposal for Catmull-Clark surfaces. Reverse subdivision is the process of taking a high resolution mesh and generating a coarse one which, when refined by subdivision, gives a good approximation to the original. Direct and reverse subdivision may thus be used together, to take any mesh and automatically generate a whole hierarchy of Levels Of Detail (LODs), both coarser and more refined than the base mesh. Such a hierarchy, however, will contain just models at uniform resolution, i.e., the same LOD is used throughout all parts of an object. In order to become really competitive with polygonal modeling, subdivision modeling should also support *selective refinement*, i.e., the possibility to vary LOD smoothly across a mesh and dynamically through time. To this aim, it is necessary to combine different levels of subdivision in the context of a mesh, without losing consistency with an underlying subdivision scheme (see Figure 4.1). This

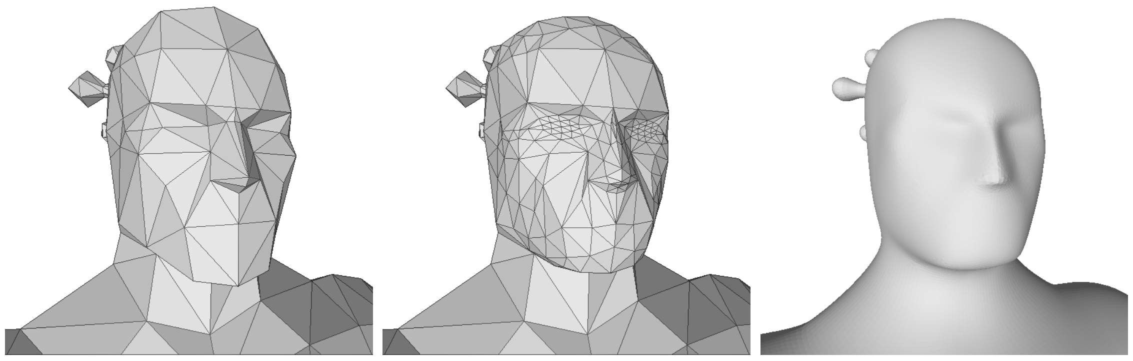


Figure 4.1: *A polygonal model (left) is selectively refined through adaptive triangular subdivision, by increasing level of detail in the parts representing eyes, nose, mouth and the top of the head (center). The limit surface of the adaptive subdivision is coincident with that of the Loop subdivision (right). Model courtesy of Silent Bay Studios* `http://www.silentbaystudios.com`

is the central issue investigated in this chapter.

Motivation

Most often subdivision is applied up to a certain level and the resulting mesh is used for further processing [ZS00]. Even when users are interested in processing/rendering the limit surface, subdivided meshes can be useful in intermediate computations. For instance, physical engines for animation, as well as system solvers for the finite element methods, work on polygonal meshes with a limited budget of cells.

In many cases, it may be desirable to refine different parts of the mesh at different levels of detail. For instance, the design of characters for videogames is constrained by a certain budget of polygons. The skin of a character is a mesh that may need more polygons in detailed areas and in the proximity of joints, while coarser and rigid parts are modeled with a smaller number of polygons. Similar arguments apply for adaptive meshes that discretize a domain to be analyzed by finite element methods. Manually adjusting the level of detail of the different parts of a mesh may be a tedious task, unless sophisticated tools to control LOD are made available. A designer or engineer would rather like to have a tool that allows her/him to set the desired level of detail on selected parts of the mesh, while letting the system automatically either refine or coarsen such parts and blend them with the rest of the mesh in a seamless way. Depending on the context, automatic criteria may also be applied, like refinement on the basis of morphological features (e.g., curvature), or view-dependent refinement [PS07].

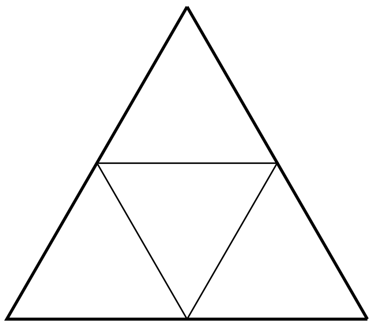


Figure 4.2: *The one-to-four triangle split pattern.*

This sort of mechanism is customary in Continuous Level Of Detail (CLOD) applied to free-form mesh modeling [LRC+02]. The resulting mesh must always be conforming (i.e., free of cracks) and transition between different LODs should be as smooth as possible. In order to support selective refinement dynamically and efficiently, it is crucial that a a mesh at intermediate LOD can be modified on-line through selective refinement in either way, by refining some parts of it while other parts may be coarsened. To this aim, refinement and coarsening operations must be based on local operators and be easily reversible.

In subdivided meshes, there is an obvious relation between level of detail and level of subdivision, thus adaptivity can be achieved only if cells at different levels of subdivision are combined in the context of a single mesh. However, classical subdivision schemes are based on the application of recursive patterns that act uniformly over the whole surface. For instance, the popular Loop [Loo87] and butterfly [DLG90] schemes for triangle meshes are based on recursive one-to-four triangle split (see Figure 4.2), which gives non-conforming meshes when applied adaptively at different levels of subdivision (see the left side of Figure 4.5). Similar arguments hold for schemes based on one-to-four split of quad meshes.

Quad-based meshes

Polygonal modeling is the main modeling paradigm for applications that require computational intensive tasks other then rendering, such as video games and finite element methods. In this context, quad-based meshes are often preferred to triangle-based ones, since they provide a more stable and better controllable framework for texturing, modeling and geometric computations. One notable property of quads is the possibility to be naturally aligned to anisotropic design features, as well as to line fields, or cross fields, such as those corresponding to principal curvatures [BZK09, DBG+06, KNP07, RLL+06].

A standard approach to polygonal modeling consists of starting from a coarse base mesh, which is then interactively edited and refined to model the features of the desired shape. One main goal is to obtain a mesh having a controlled budget of polygons, while being

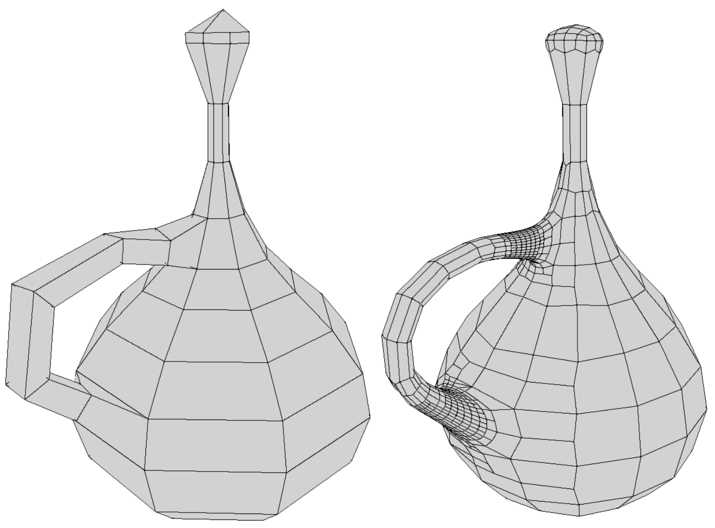


Figure 4.3: *Subdivision surface: (a) The control mesh of a smooth object, is adaptively subdivided (b) and its vertices are moved to their limit position.*

close to an ideal smooth surface. Mesh subdivision is often used to this purpose [MS01a]. Non-trivial shapes may require adaptive subdivision to model different parts: tiny but relevant features will require a much finer mesh than large uniform areas. But subdivision is generally meant as a global process, while adaptive refinement of quad meshes is non trivial: local refinement of quads produces non-quad faces and this process, if performed in an uncontrolled manner, can soon destroy the regular structure of a mesh.

On the other hand, several authors have remarked that quad-dominant meshes containing a small amount of non-quad elements can be more flexible and more effective than purely quad meshes in capturing surface features, and they may enrich the design space [MNP08, SL03]. For instance, triangular and pentagonal elements can be used to collapse, split and merge lineal features and line fields, as well as to model the surface in the proximity of singularities.

Examples of results obtained with our adaptive quad subdivision scheme are shown in Figures 4.22 and 4.3. Our method generates an implicit hierarchy of adaptively refined quad-dominant meshes, each containing a small amount of triangular and pentagonal transition elements. The adaptive subdivision patterns preserve the surface flows and lineal features, which are defined on the base mesh, at all meshes in the hierarchy.

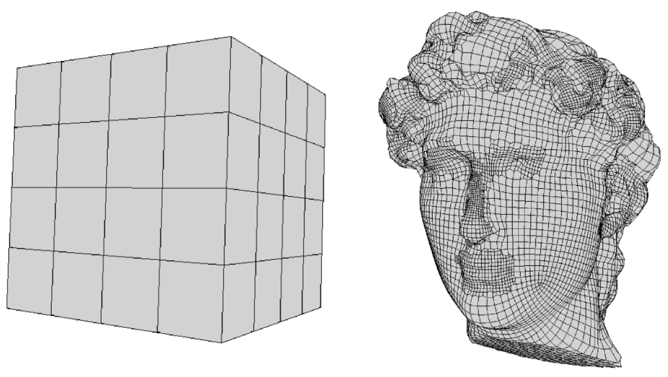

(a) (b)

Figure 4.4: *Remeshing: (a) A simple mesh used to impose the base topology; (b) an adaptively remeshed model is obtained by selective refinement and projection of vertices to the reference shape.*

Contribution

We introduce two adaptive subdivision schemes, one for triangle meshes and the second for quad meshes. Both shares similar properties, since they are based on the iterative application of local refinement and coarsening operators. It is possible to extend the schemes schemes to generate the same limit surface of the Loop or Catmull-Clark subdivision, respectively. They produce the same limit surface independently on the order of application of local operators, they support dynamic selective refinement and they generate conforming meshes at all intermediate steps. Our hierarchy is *implicit,* meaning that only a mesh at any intermediate level of refinement is encoded at each time, while all other meshes of the hierarchy can be easily obtained from it, by means of local conforming operators, which support both refinement and coarsening and work on a plain mesh without the need of cumbersome hierarchical data structures.

The main contributions in this chapter are the following:

1. We define local operators for both refining and coarsening a subdivision mesh of triangles/quads by inserting/deleting one vertex at a time;
2. On the basis of such operators, we define their transition space and we study them as purely combinatorial structures. We show that they are highly adaptive;
3. We provide traversal operators that work on an adaptively refined mesh M, while supporting navigation of any mesh coarser than M in the implicit hierarchy.
4. We describe a light data structure for adaptive subdivision, which does not need to store any hierarchy;

5. We provide a variant of the well-know Catmull-Clark subdivision scheme for our adaptive subdivision hierarchy;
6. We show that our adaptive scheme can be used as a flexible re-meshing tools

4.1 Related work

The work presented in this chapter has relations with work on subdivision surfaces and work on CLOD models for meshes. We review the literature on these two topics in separate subsections, by focusing just on results that have direct relation with our work.

Adaptive subdivision

The literature on subdivision surfaces is quite extended. The interest reader can refer to [WW02] for a textbook, [ZS00] for a tutorial and [Sab04] for a survey. Here, we will review only those works related to adaptive subdivision.

Red-green triangulations were introduced in the context of finite element methods [BSW83], and have become popular in the common practice, as an empirical way to obtain conforming adaptive meshes from hierarchies of triangle meshes generated from one-to-four triangle split. Red-green triangulations are usually built through a two-step procedure: first by applying one-to-four triangle split adaptively, and then by subdividing some triangles further, through predefined patterns, to fix non conforming situations (see Figure 4.5). Depending on the underlying subdivision scheme, the geometry of vertices (control points), which lie on the transition between different levels of subdivision, may not correspond to that of the same vertices in a uniformly subdivided mesh. This fact, which is often overlooked, may prevent the correctness of further subdivision or coarsening of a red-green triangulation, unless the subdivision process is repeated from scratch. This latter option is unwieldy, it prevents incremental editing of LOD, and it may be not sustainable for on-line processing.

A variant of red-green triangulations was used in [ZSS97] to support multi-resolution editing of meshes based on the Loop subdivision scheme. Adaptive meshes are computed by reverse subdivision, starting at the finest level and pruning over-refined triangles. Also in this case, a restricted non-conforming mesh is computed first, which is fixed next by further bisection of some triangles. Correct relocation of vertices is treated by using a hierarchical data structure that stores the positions of all control points in the uniform subdivision. Recently, another variant of red-green triangulations, called incremental subdivision, was presented in [PS07] for both the Loop and the butterfly schemes. In this case, a larger support area for refinement is used, in order to correctly compute the geometry of control points. The adaptive refinement algorithm seems to work level by level and coarse-to-fine

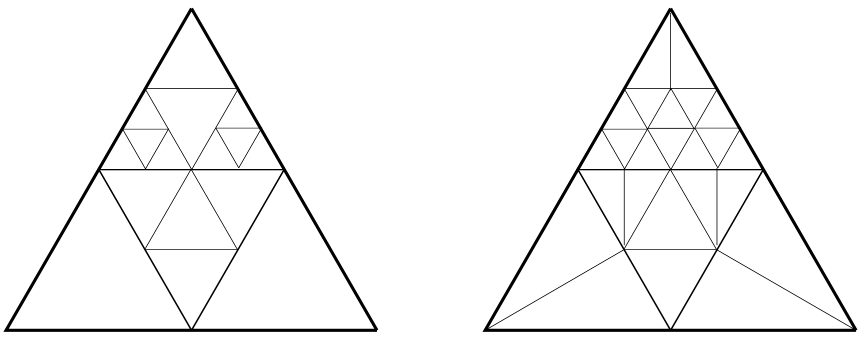


Figure 4.5: *Red-green triangulation: a non-conforming mesh obtained from adaptive one-to-four split (left) is made conforming by splitting some triangles further depending on the level of their neighbors (right).*

only. With respect to standard red-green triangulations, incremental subdivision requires a larger number of triangles to achieve the same level of adaptivity.

The RGB subdivision proposed in [PP09a, PP09b] extends red-green triangulations with the same subdivision schemes to a fully dynamic adaptive scheme supporting both local refinement and coarsening.

The schemes proposed in [MJ98] and [XK99] apply to quadrilateral meshes and extend the Catmull-Clark [CC78] and Doo-Sabin [DS78] subdivision methods, respectively, to become adaptive. These methods are in spirit similar to subdivision based on red-green triangulations, and suffer from the same drawbacks. In [Pup07], a scheme similar to the RGB subdivision presented here is sketched, which can be applied to quad meshes to obtain adaptive tri-quad meshes, while remaining consistent with the Catmull-Clark subdivision.

In [SHHG01], the one-to-four triangle refinement scheme is decomposed into atomic local operations, called *quarks*, based on the popular *vertex split* operation that is at the basis of Progressive Meshes [Hop96]. A red-green triangulation under the butterfly scheme [DLG90] is obtained through a sequence of quarks. No explicit algorithm for selective refinement is proposed in [SHHG01].

In [Vel03] the application of stellar theory to subdivision is investigated. The factorization of one-to-four triangle split into a sequence of edge split and edge swap operations is proposed: this is a subset of the local operators that we use in Section 4.3. A factorization of the Loop subdivision rule is also proposed, which makes it possible to compute the control points correctly through the sequence of local refinement operations. No operators for inverse subdivision are proposed and selective refinement is not investigated in [Vel03].

The $\sqrt{3}$ subdivision [Kob00] and the 4-8 subdivision [VZ01] schemes are not based on the classical one-to-four triangle split operator. They are naturally adaptive, being both based on local conforming operators.

The $\sqrt{3}$ subdivision alternates one-to-three triangle split (insertion of a new vertex at the center of each triangle) at one level, with edge swap at the next level. This scheme generates triangles that can be regarded as being of *green* and *blue* types in the terminology that we introduce in Section 4.3. In order to relocate vertices correctly, some over-refinement of neighbors of even (green) triangles is imposed. A closed form solution of the subdivision rule permits to compute control points for a vertex at any level on the basis of just its initial position and its limit position. Adaptive refinement is supported, while adaptive coarsening is not investigated explicitly in [Kob00].

The 4-8 subdivision is based on edge split, as in our case, applied to a special case of triangle meshes, called *tri-quad meshes*. An initial tri-quad mesh can be obtained from any triangle mesh by doubling its number of triangles and changing its topology [VZ01]. The correct position of control points is addressed and resolved also in this case with a certain amount of over-refinement of the mesh. Only basic operations are investigated in [VZ01], while no selective refinement algorithm is proposed.

CLOD models

Also the literature on Continuous Level of Detail models is very wide. The interested reader may refer to [LRC$^+$02] for a book on this subject. Here we review only some concepts and contributions that are related to the rest of the Chapter. Generally speaking, a CLOD model consists of a base mesh at coarse resolution, plus a set of local modifications that can be applied to the base mesh to refine it. Such modifications are usually arranged in a hierarchical structure, which consists of a directed acyclic graph (DAG) in the most general case. Meshes at intermediate level of detail correspond to cuts in the DAG, and algorithms for selective refinement work by moving a front through the DAG and doing/undoing modifications that are traversed by this front. This general framework, as shown in [Pup98], encompasses almost all CLOD models proposed in the literature and it can be applied to the hierarchies generated by $\sqrt{3}$ subdivision and 4-8 subdivision as well.

In [KL03], a CLOD model is introduced, which achieves better adaptivity by using local modifications more freely than in previous models. In this case, modifications cannot be arranged in a partial order and encoded in a DAG. In Sections 4.2.2 and 4.3.4, we use the idea of transitive mesh space proposed in [KL03] to study the expressive power of our adaptive subdivision schemes.

CLOD models can provide meshes at intermediate LOD, where detail can vary across the mesh and through time, at a virtually continuous scale and with fast procedures that work on-line even for huge meshes. The scheme proposed in [DWS$^+$97] is very popular and most authors refer to it in order to implement their selective refinement algorithms. Refinement modifications are dynamically done/undone on a current mesh at intermediate LOD, on the basis of two priority queues and user-defined selection criteria, which drive the choice of modifications needed to adapt the mesh to used needs.

There exist a few CLOD models based on recursive subdivision patterns. The model proposed in [DWS$^+$97] is based on the recursive bisection of right triangles. This rule is also used by several other authors, and may be regarded as a subdivision scheme. It can be applied just to meshes obtained from regular grids (typically representing terrains), while its extension to more general triangle meshes is not straightforward. One generalization is given by 4-k meshes [VG00], which have in fact a strong relation with 4-8 subdivision [VZ01].

We are not aware of any CLOD model developed upon the one-to-four triangle split pattern.

4.2 Adaptive quad subdivision

We deal with manifold polygonal meshes containing triangles, quads and pentagons, adopting standard terminology. A *quad mesh* is a mesh where all faces are quads; a *quad-dominant* mesh is a mesh where most faces are quads. A vertex v in a quad mesh (or in a quad-dominant mesh containing just quads in the star of v) is *regular* if it has valence four; otherwise it is said to be *extraordinary.* The *edge neighbors* of a vertex v in a quad[-dominant] mesh are those vertices connected to v through edges; the *face neighbors* of v are those vertices opposite to v on its incident quads.

We edit the mesh by local refinement operations that split one edge by inserting a vertex, and local coarsening operations that merge a pair of edges by removing a vertex. A local operation eliminates faces in the neighborhood of the vertex to be inserted/removed, and re-tessellates the hole with new faces. This will be the subject of Section 4.2.1. The iterative application of local operators define an implicit hierarchy, described in Section 4.2.2. In Section 4.2.3 the concept of topological angles and lengths are introduced and used to define navigation algorithms that allows us to navigate through the subdivided mesh and across the different levels of the implicit hierarchy.

4.2.1 Topological operators

The most common pattern for uniform subdivision of quad meshes is *quadrisection*: each face is subdivided into four new faces by splitting each edge with a new vertex, and connecting each such vertex with another new vertex at the center of the face. In the following, a vertex that splits an edge will be said to be of type E, while a vertex inserted in the middle of a face will be said to be of type F.

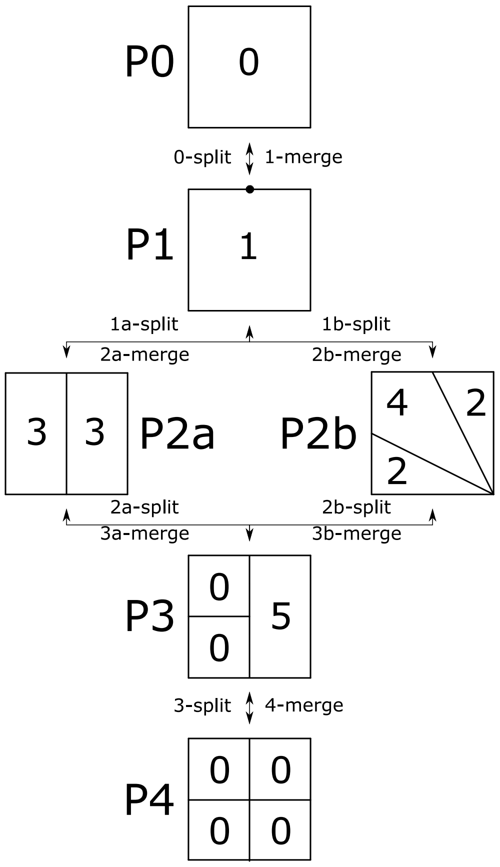


Figure 4.6: *The diagram of patterns for adaptive subdivision of a quad. Types of faces and patterns are denoted by labels placed inside and beside them, respectively. Transitions between adjacent patterns are labeled with the corresponding refinement and coarsening operators.*

Since quadrisection splits all edges of a face, it cannot be applied selectively while maintaining the mesh conforming. In order to support transitions between different levels of subdivision, we devise alternative patterns that split one edge at a time. Figure 4.6 illustrates such a set of patterns, and related operators. These patterns produce quad-dominant meshes containing some triangular and pentagonal faces, which preserve the flow of lines of the base mesh (see Section 4.2.4 for a discussion). Other patterns, e.g. containing just quads and triangles could be also used with straightforward modifications of the method described below.

A key idea is that local operators subdivide a mesh by splitting one edge at a time, they always produce conforming triangulations, and they can be controlled just on the basis of attributes of local entities, i.e., types and levels of vertices, edges and faces. All rules that control operators are purely topological. Just for the sake of clarity, in the figures we will use fixed shapes to depict the different types of faces that may appear in our adaptive meshes: squares (type 0); rectangles(type 3); diamonds (type 4); right triangles (type 2); and pentagons with three collinear vertices, having the shape of either a square or a rectangle (type 1 and 5, respectively).

Consider a quad mesh Γ_0, called the *base mesh.* We assign level zero to all its vertices and edges, and type *standard* to all its edges. A selectively refined mesh will also contain vertices and edges labeled according to their level of subdivision; edges will be labeled with either type *standard* or type *extra.* The only extra edges are those internal to patterns P2a and P2b, which have the same level l of their parent face (i.e., face 0 in pattern P0); the remaining edges are standard and they have level either l or $l+1$, as in the standard subdivision. The level of a face in a mesh Γ is defined to be the lowest among the levels of its edges. Note that the type of a face is uniquely defined by the types and levels of its edges, and the type of a pattern is also uniquely defined by levels of edges in its boundary, as follows:

- Type 0 (also called *standard*) is a quad with all four edges at the same level (and of standard type);
- Type 1 is a pentagon with three consecutive (standard) edges at level l and two consecutive (standard) edges at level $l+1$;
- Type 2 is a triangle (with one standard edge at level l, one standard edge at level $l+1$ and one extra edge at level l);
- Type 3 is a quad with two non-consecutive edges at level l (one of which is an extra edge) and the other two (standard) edges at level $l+1$;
- Type 4 is a quad with two consecutive (extra) edges at level l and the other two (standard) edges at level $l+1$;
- Type 5 is a pentagon with four (standard) edges at level l and one (standard) edge at level $l+1$.

4.2.1.1 Refinement operators

According to definitions above, all faces in the base mesh are standard at level zero. Local subdivision operators can be applied iteratively to Γ_0 to generate a conforming mesh Γ composed of faces of the six types illustrated in Figure 4.6.

We say that an edge e at level $l \geq 0$ is *refinable* (i.e., it can split) if and only if it is standard and its two adjacent faces f_0 and f_1 are both at level l. In case of a boundary edge, only one such face exists. We split an edge e at level l, by inserting at its midpoint a new vertex v at level $l+1$. The edges generated by the two halves of e are standard at level $l+1$.

Note that levels of vertices and standard edges comply with the standard subdivision.

Splitting an edge e at level l may affect an area as large as that of the standard faces incident at e at level l. Tessellations on the two sides of e can be treated independently and each of them depends on the type of the face f incident at e and on its configuration, as explained in the following, and depicted in Figure 4.6:

- **0-split:** if f is of type 0, then it is changed into a face of type 1, having its edges of level $l+1$ at the two halves of e incident at the new vertex v.
- **1a-split:** if f is of type 1 and both its edges adjacent to e are at level l, then two faces of type 2 are generated by connecting v to the vertex of f opposite to e (which is at level $l+1$). The edge shared by the two new faces is extra and at level l.
- **1b-split:** if f is of type 1 and one edge e' adjacent to e is at level $l+1$, then another vertex v' is inserted inside the face and a tessellation composed of one face of type 4 at level l and two faces of type 3 at level l is created; for the sake of brevity, we refer to the figure to describe how this tessellation is made; the new edges inserted inside the face are extra at level l.
- **2a-split:** if f is of type 2, then it is necessarily adjacent to another face f' of type 2 through an extra edge opposite to e. Another vertex v' is inserted that splits the extra edge between f and f'; f is decomposed into two standard faces at level $l+1$ by connecting v to v'; and f' is changed into a face of type 5. The new edges are all standard at level $l+1$.
- **2b-split:** if f is of type 3, then it is necessarily part of a tessellation of a standard face at level l, formed by f as well as of another face f' of type 3 and a face f'' of type 4 placed between f and f'. Such three faces are removed and the hole is tessellated with the same pattern described in the previous case;
- **3-split:** if f is of type 5, then it is split into two standard faces at level $l+1$ by connecting v to the vertex opposite to e with a standard edge at level $l+1$.

It is readily seen from Figure 4.6 that in all cases the type of face f incident at splitting edge e and the levels and labels of edges of f are sufficient to characterize the type of operator to be applied. This fact allows us to pre-compute and store in a lookup table the local tessellations to be deleted from, and to be plugged into a mesh. Note that all split operators affect the whole area covered by a pattern, except for 3-split, which affects just the area covered by face of type 5 in pattern P3. In fact, the other two (standard) faces at level $l+1$ may be actually refined independently at higher levels before this operator is applied.

By simple combinatorial analysis, it is easy to verify that the set of refinement operators is closed with respect to the meshes obtained, i.e.: if we start at a mesh Γ_0 containing all standard faces at level 0 and we proceed by applying any legal sequence composed of the operators above, the resulting mesh will be composed of faces of the types defined above, and all its refinable edges can be split through the same set of operators. In particular, all vertices of a standard subdivision up to a given level l can be added without adding any vertex of a level higher than l and the same uniform mesh generated from the standard subdivision scheme will be obtained.

If an edge e at level l is of standard type, but it is not refinable, we trigger recursive refinement of each face f incident at e and having a level $< l$. Recursive refinement is performed by recognizing the type of f and forcing refinement of either two (for pattern P1) or one (for all other patterns) of its edges at level $< l$.

4.2.1.2 Coarsening operators

Local merge operators invert edge split operators defined above, by removing one vertex v at level $l + 1$ that splits an edge e at level l. As before, at most the area spanned by the faces incident at e at level l may be affected, and the tessellations of such two areas are treated independently.

A vertex v at level $l + 1$ is *potentially removable* if the levels of its incident faces are: $l + 1$ for standard faces (type 0), and l otherwise. A potentially removable vertex is *removable* if it is of type E (i.e., it splits an edge of the previous level) and faces in its neighborhood can be arranged to form two patterns of the diagram, sharing a pair of edges at level $l + 1$ incident at v. Vertices of type F (i.e., splitting a face of the previous level) do not trigger any merge operator, because they are removed together with vertices of type E from operators 3a/b-merge. Such vertices are discarded easily because a potentially removable vertex v at level $l + 1$ is of type F if and only if either it has exactly four adjacent vertices at level $l + 1$, or its star is formed by two standard faces and one face of type 5.

We divide the neighborhood of (internal) vertex v of type E in two halves as follows: there are at most four (standard) edges at level $l + 1$ incident at v; among them, only two edges e' and e'' have the other end vertex at level $\leq l$; thus the pair e', e'' cover the edge e that was split by v and they divide the neighborhood. For each half neighborhood, we have the following possible cases:

- **4-merge**: both faces incident at e' and e'' are standard, and the faces adjacent to each of them on the edges opposite to e' and e'', respectively, are different; then v is removable and the two faces incident at e' and e'' are replaced with a single face of type 5 (note that the region covered by the other two faces of pattern 4 is not affected);
- **3a-merge**: both faces incident at e' and e'' are standard, and they are both adjacent to the same face (of type 5) along the edges opposite to e' and e''; then v is removable

and a pattern of type 3 is replaced with a pattern of type 2a;

- **3b-merge**: one standard face and one face of type 5 are incident at e' and e''. Just in this case, we must check that these two faces are both adjacent to another face f of type 0 at level $l+1$ (which completes pattern 3). If this condition is not fulfilled, it means that the third face of pattern 3 (which is not incident at v) has been subdivided further, therefore v is not removable. Otherwise, v is removable and a pattern of type 3 is replaced with a pattern of type 2b.
- **2a-split**: both faces incident at e' and e'' are of type 2; then v is removable and a pattern of type 2a is replaced with a pattern of type 1;
- **2b-split**: one face of type 3 and one face of type 4 are incident at e' and e''; then v is removable and a pattern of type 2b is replaced with a pattern of type 1;
- **1-merge**: both e' and e'' are incident at the same face of type 1; then v is removable and a pattern of type 1 is replaced with a pattern of type 0.

For each half neighborhood, we recognize the pattern adjacent to the pair e', e'' and we apply the corresponding merge operator, as depicted in Figure 4.6. Operators 1-merge and 2a/b-merge can be applied by checking just the types of faces f' and f'' incident at e' and e''. In order to discriminate between operators 3a-merge and 4-merge it is also necessary to check the type of face(s) adjacent to f' and f'' inside the pattern. Finally, operator 3b-merge needs checking also the other face of type 0 adjacent to the face of type 5 within pattern P3: in fact, v is not removable if such a face has been refined further.

It is easy to verify that the merge operators are consistent with the split operators and they have similar properties: all merge operators affect the whole area covered by a pattern, except for 4-merge, which affects just half a pattern; local tessellations to implement operators are precomputed and stored in a lookup table (in fact, the same lookup table that is used for the split operators).

The set of refinement operators is also closed with respect to the meshes obtained. If we start at a mesh Γ obtained from Γ_0 through refinement, we can apply merge operators in any legal order to go back to Γ_0; moreover, any intermediate mesh could be refined through split operators. So we can mix split and merge operators in any order while preserving consistency.

4.2.2 Transition space and implicit hierarchy

Following the approach of [PP09b], it is possible to define a transition space for our adaptive subdivision scheme. A transition space is a graph where each node corresponds to an adaptive mesh, and each arc corresponds to the application of an atomic local operation, as defined in Section 4.2.1. The arcs can be divided in two groups: the first contains all refinement operations while the second contains the inverse operators that coarsen the mesh. In the graph, a path from node a to node b corresponds to a set of atomic operations that transform the mesh associated with a into the mesh associated with b. Since all operators are invertible, it is always possible to execute also the inverse sequence of operations. Note that multiple paths may exist between a pair of nodes.

Consequently, the application of the local operators to a base mesh defines a (virtually unlimited) subdivision hierarchy. We do not need to store the hierarchy explicitly. Starting from a single node n, we are able to determine what operations are valid and subsequently what arcs of the graph are incident with n. In other words, we always know the subdivision hierarchy locally, around our current mesh, and we are able to navigate the graph by applying local operators, even if we do not store it explicitly.

4.2.3 Topological angles and lengths

Fetching stencils in an adaptively refined mesh requires navigating the mesh. In order to support navigation efficiently, we assign a *topological width* to every angle defined by a pair Face-Vertex (F, V) in a mesh. The values are assigned to faces of the various types by using the following rules: (see Figure 4.7):

1. *Types 0 and 3*: each angle has a topological width of 3;
2. *Type 1*: the angle at the vertex with higher level has a topological width of 6, while the other angles have a topological width of 3;
3. *Type 2*: the angle at the vertex with higher level has a topological width of 2; the one at the vertex incident at both standard edges has a width of 3, and the third angle has a width of 1;
4. *Type 4*: the angles at the two vertices with higher level have a topological width of 4; the one at the vertex incident at both extra edges has a width of 1, and the fourth angle has a width of 3;
5. *Type 5*: each angle has a topological width of 3, except the one at the higher level that is connected by its incident edges to other two vertices at the same level, which has a width of 6;

An angle with topological width of 6 is said to be *flat*. Such values are not related to geometrical values, we call them “angles” since they satisfy some properties of geometrical angles, which we will show in the following. We do not need to store angle widths, since they can be found efficiently from types of faces and edges, and levels of vertices.

We give next some invariants on angles that will be useful for mesh navigation.

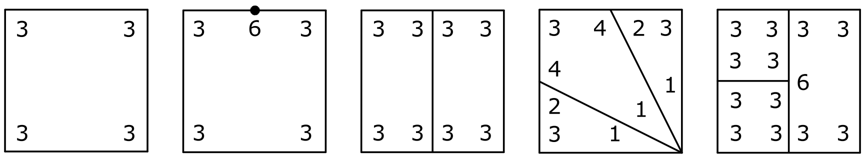


Figure 4.7: *Topological angles: a width is assigned to each vertex in each face.*

Lemma 8 *If an edge e is split into two edges e_0 and e_1 by adding a vertex v, both angles formed by e_0 and e_1 are flat.*

Proof. Figure 4.7 shows the only possible ways to split an edge. It is readily seen that in all cases the sum of angles on each side of a pair e_0e_1 is 6. □

Lemma 9 *The width of a topological angle between a pair of edges is invariant upon editing operations on the mesh.*

Proof. Consider a pair of edges e and e' incident at v and one of the two angles they form at v. It is sufficient to analyze editing operations that affect faces spanned by such an angle. For each such face f, there are three possible cases, which are readily verified by comparing transitions depicted in Figure 4.6 with angles depicted in Figure 4.7:

If the editing operation neither splits f with an edge incident at v, nor merges t with an adjacent face around v, then the angle of f at v is unchanged;

If the angle of f at v is split into two angles, then the sum of widths of such angles is equal to the width of the angle of f at v before split;

If f is merged with another face f' adjacent to it around v, by deleting their common edge, then either e and e' are merged into a single edge, or the width of angle at v of the new face is equal to the sum of widths of angles of f and f' at v. □

Lemma 10 *No matter how an edge e is subdivided into a chain of edges $e_0, \ldots, e_k$, angles between two consecutive edges e_{i-1} and e_i, $i = 1, \ldots, k$ are flat.*

Proof. The proof follows from the above two lemmas by noting that every split produces flat angles and such angles are invariant upon subsequent editing operations. □

Topological lengths are assigned to standard edges inductively: an edge at level 0 has unit length; an edge at level $l + 1$ has half the length of an edge at level l.

The previous definitions and lemmas allow us to define a set of operators for mesh navigation, which help us extracting from an adaptively refined mesh a view of the same mesh at a lower level of subdivision. We define switch operators similar to those proposed in [Bri93], plus two new operators, called **rotate** and **move**, that are specific for our meshes. All operators use a unique identifier of position in a mesh, called a *pos*, which contains a vertex v, an edge e incident at v, and a face f bounded by e. Given a *pos* **p**, we will denote by **p.v**, **p.e** and **p.f** its related vertex, edge and face, respectively.

1. **p.switchVertex(), p.switchEdge()** and **p.switchFace()** move to the adjacent *pos* which differs from **p** just for the vertex, the edge and the face, respectively. (switches are equivalent to the operators defined in [Bri93]).
2. **p.rotate(i)**: executes an alternate sequence of **p.switchEdge()** and **p.switchFace()** operators until a topological angle of width **i** has been spanned.
3. **p.move(l)**: executes an alternate sequence of **p.switchVertex()**, possibly followed by **p.rotate(6)** and **p.switchFace()** operators until the length of an edge at level $\leq l$ has been traversed. If the first edge traversed has a level $< l$ (i.e., its length is larger than required) the operation has no effect.

The effect of operators is exemplified in Figure 4.21: single arrows correspond to switch and rotate operators; the expanded view shows the decomposition of a rotate(6) operator in terms of switches; sequences of vertical arrows correspond to move operators for the length of one edge at the coarsest level.

Lemma 11 *The rotate and move primitives are invariant during editing, every result that we obtain on a level of the subdivision is invariant in any deeper level. For invariant we mean that if we consider a uniformly refined mesh at level l and we apply one of the previous operation on it, we obtain exactly the same result that we would get on another mesh at any further subdivision level.*

Proof. The rotate primitive is invariant since the angle it spans is invariant by Lemma 9. The move primitive is invariant since if we refine a mesh we can only add vertices at a higher level, and any angle we add along the line traversed by the Move operation has a width of 6, which is skipped by the Move operation; moreover, the topological length of any chain of edges splitting an edge e is also invariant by definition. □

The invariance lemmas shown in the previous section guarantee that, starting at a splitting edge **p.e** at level l, we can navigate the mesh by moving to adjacent faces of the stencil at level l (through a **p.rotate(3)** operation) and we can follow chains of edges until we reach the other end of an edge at level l (through a **p.move(l)** operation).

4.2.4 Alignment with surface flows

In many cases, the alignment of edges of a quad mesh are relevant to the modeled shape. One example is when edges are aligned with shape features, another is when they are aligned with a cross field defined on the surface [BZK09]. For instance, both in the finite element methods and in shape approximation, good anisotropic meshes for a given budget of elements can be obtained if elements are aligned to principal directions of curvature [She02].

If quads are properly aligned with a line field defined on the surface, then edges can be colored with two colors, say red and blue, depicting two orthogonal flows on the surface (see Figure 4.10). These flows are obviously preserved through quadrisection of quads - see Figures 4.8(a) and (b) - but they can be deviated by some adaptive patterns - see Figure 4.8(c).

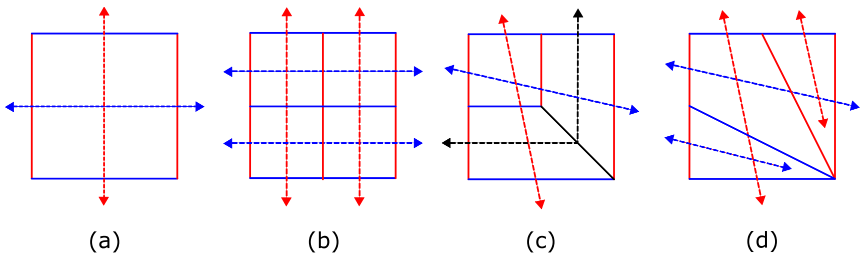


Figure 4.8: *Preserving flows in a quad mesh: (a) flows traverse a quad element in orthogonal directions; (b) flows are preserved by uniform subdivision; (c) some adaptive patterns break flows and do not allow for consistent labeling of edges; while others maintain flows but introduce non-quad elements (d). Arrows denote flows traversing elements.*

Triangular and pentagonal faces in a quad-dominant mesh collapse and split/merge flows, respectively, as depicted in Figures 4.9(a) and (b). Note that the direct application of Catmull-Clark patterns to non-quad meshes would *not* preserve flows. See Figures 4.9(c) and (d). We have used the pattern depicted in Figure 4.8(d) instead of the more popular Y pattern of Figure 4.8(c) for configuration P2b of our adaptive scheme, since it allows us preserving the same flows of its parent quad. It is straightforward to see that also the other patterns of our scheme preserve flows, therefore the flows defined by edges of elements in an adaptively refined mesh will be consistent with those of the base mesh.

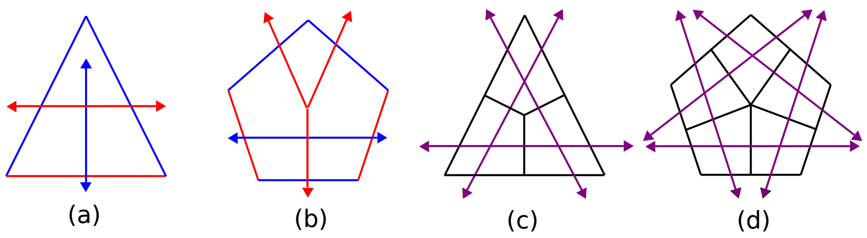


Figure 4.9: *Triangular and (a) pentagonal (b) transition elements collapse and split/merge the flow in one direction, respectively. Catmull-Clark subdivision of triangles (c) and pentagons (d) does not preserve the flows, though.*

Figures 4.10 shows a base mesh representing a torus, with edges aligned with principal directions of curvature, and an adaptively subdivided mesh obtained from it. Flows are shown on edges and faces by means of textures.

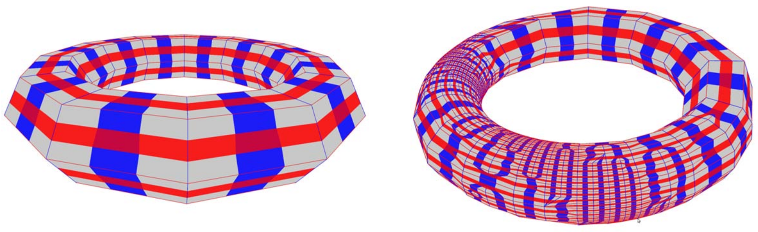


Figure 4.10: *A torus with edges aligned to principal curvature directions and an adaptive subdivision of it.*

4.2.5 Implementations details

The data structure to represent a mesh has been extended just with attributes to keep the level and type of each edge, and the level of each vertex. Assuming a maximum of 16 levels of subdivision, which is more than sufficient for practical purposes, such attributes can be maintained with one byte per edge, and one byte per vertex. For adaptive subdivision only, also summations and counters to compute control points are maintained, requiring additional six floats per vertex.

In order to analyze space occupancy, we note that our scheme implicitly encodes the subdivision hierarchy corresponding to a quad-tree representation, by representing just its leaves. In the case of a complete tree, which encodes a uniform subdivision, we encode about 67% of the total number of quad-tree nodes. Our scheme uses about 33% less space than the multi-resolution half-edge data structure presented in [KCB09], and about 3% more space than a full quad-tree, encoded as in [KCB09].

Our prototype running on a single core of a T9300 Intel Core Duo at 2.5 Ghz can insert/remove about 40K vertices per second. The framework can thus easily support interactive LOD editing even with large meshes.

4.3 Adaptive triangular subdivision

RGB triangulations are defined as all those triangle meshes that can be built through iterative application of given operators for local modification, starting at a base mesh Σ_0. In this section we introduce their combinatorial structure and the basic rules to manipulate them in a consistent way. In Subsection 4.3.1 we define local subdivision operators. The essential idea is that such operators subdivide a mesh by introducing one vertex at a time, they always produce conforming triangulations, and they can be controlled just on the basis of color and level codes. In Subsection 4.3.2 we introduce local coarsening operators, which reverse refinement operators, while in Subsection 4.3.3 we add one neutral operator.

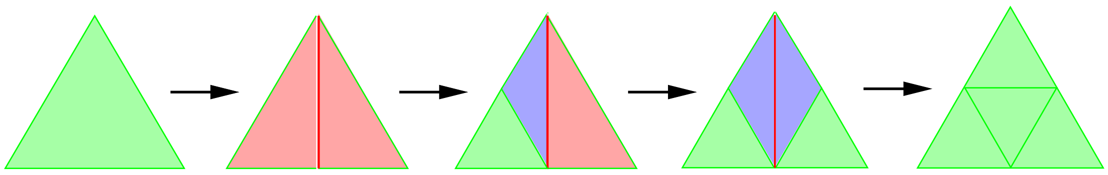


Figure 4.11: *One-to-four triangle split of a green triangle is factorized into three edge split operations plus an edge swap operation.*

Subsection 4.3.4 has a more theoretical flavor: we define and study the transitive space of RGB triangulations, in order to show their expressive power and adaptivity; we also prove some results useful to warrant correctness of the selective refinement algorithm described in the next section.

All rules defined in this section are purely topological. Just for the sake of clarity, in the figures we will use meshes composed of equilateral triangles, right triangles and isosceles triangles to depict the three different types of triangles that may appear in a RGB triangulation. Actually, the shape of triangles is totally irrelevant in the subdivision process, while just level and color codes matter.

4.3.1 Local subdivision operators

Consider a base mesh Σ_0. We assign level zero to all its vertices, edges and triangles, and color green to all its edges and triangles. In the following, we define local subdivision operators that, when applied iteratively to Σ_0, will generate a conforming mesh where triangles will be colored of green, red and blue; edges will be colored of green and red; and vertices, edges, and triangles will have different levels. Color and level codes allow us to control the application of subdivision operators on a local basis.

Our aim is to factorize one-to-four triangle split by introducing one vertex at a time. Following [Vel03], we can do that through a sequence of three edge split operations, plus an edge swap operation, as depicted in Figure 4.11. Since the mesh must remain conforming at each step, each edge split operation must affect both triangles incident at a splitting edge.

We say that an edge e at level $l \geq 0$ is *refinable* (i.e., it can split) if and only if it is green and its two adjacent triangles t_0 and t_1 are both at level l (in case of a boundary edge, only one such triangle exists). An edge e at level l is split by inserting a new vertex, at level $l+1$, at the midpoint of e. This induces the simultaneous bisection of triangles t_0 and t_1 incident at e. The edge split operator comes in the following variants (see Figure 4.12):

- **GG-split:** t_0 and t_1 are both green. The bisection of each triangle t_0 and t_1 at the midpoint of e generates two red triangles at level l. Each such triangle will have: one

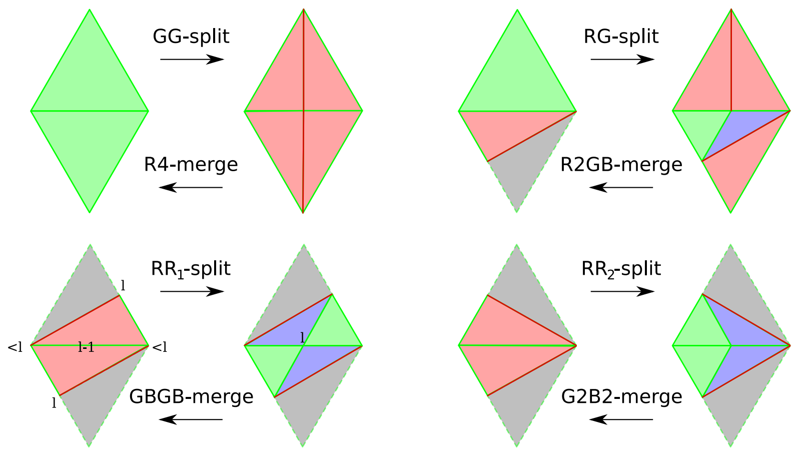


Figure 4.12: *Edge split and edge merge operators. Labels denote the level of vertices and edges.*

green edge at level l (the one common with old triangle t), one green edge at level $l+1$ (one half of e) and one red edge at level l (the new edge inserted to split t).

- **RG-split:** t_0 is green and t_1 is red. Triangle t_0 is bisected and edge e is split as above. The bisection of t_1 generates one blue triangle at level l and one green triangle at level $l+1$. The green triangle is incident at the green edge at level $l+1$ of old triangle t_1 and also its other two edges are at level $l+1$ (the edge inserted to subdivide t_1, and one half of e). The blue triangle is incident at the red edge of old triangle t_1 and has also two green edges at level $l+1$ (the edge inserted to subdivide t_1, and the other half of e).

- **RR-split:** t_0 and t_1 are both red. Triangles t_0 and t_1 are both bisected as triangle t_1 in the previous case and each of them generates the same configuration made of a blue triangle at level l and a green triangle at level $l+1$. This case may come in two variants: RR$_1$-split and RR$_2$-split. Each variant can be recognized by the cycle of colors of edges on the boundary of the diamond formed by t_0 and t_1: this may be either red-green-red-green for RR$_1$-split, or red-red-green-green for RR$_2$-split.

Edge split operations applied to boundary edges will affect just one triangle and can be applied to any green edge. The resulting configuration depends only on the color of the triangle incident at e.

BB-swap is another operator necessary to obtain all green triangles at the next level of subdivision (see Figure 4.13). It can be applied to a pair of blue triangles at level l, which are adjacent along their red edge at level l. In this case, such edge is eliminated and the other diagonal of the quadrilateral formed by such two triangles is inserted. The result is a

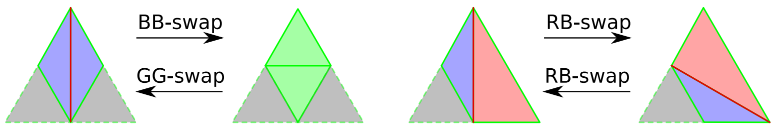


Figure 4.13: *Edge swap operators.*

pair of green triangles at level $l+1$. Note that, by construction, one of the two new green triangles will have all three vertices at level $l+1$. Note also that just green edges can be split, while red edges are only swapped.

By simple combinatorial analysis, it would be easy to verify that this set of operators is closed with respect to the meshes obtained, i.e.: if we start at an "all green" mesh Σ_0 at level 0 and we proceed by applying any legal sequence composed of the five operators above, all refinable edges in the resulting mesh can be always split by one of the four variants of edge split. Rather than proving this claim, in Section 4.3.4 we prove a more general result that also implies this fact.

4.3.2 Reverse subdivision operators

We define also local operators that invert edge split and edge swap on a RGB subdivision. Edge merge is the reverse operator of edge split and can be applied to triangles incident at vertices of valence four. The same cases depicted in Figure 4.12 occur (modifications apply right-to-left in this case):

- **R4-merge** inverts GGsplit;
- **R2GB-merge** inverts RG-split;
- **GBGB-merge** inverts RR_1-split;
- **G2B2-merge** inverts RR_2-split.

A little care must be taken in applying GBGB-merge in order to avoid inconsistencies. Referring to Figure 4.12, note that the quadrilateral must have two vertices at the same level l and two other vertices at a level lower than l. GBGB-merge must be performed by removing edges incident at the vertices of level l.

Similar rules apply to pairs of triangles along the boundary.

GG-swap, which inverts BB-swap, can be applied to a pair of adjacent green triangles t_0 and t_1 at level $l>0$ if one of them, say t_0, has all three vertices at level l. This condition is necessary and sufficient to guarantee that t_0 and t_1 have the same parent triangle t in the subdivision and t_0 is the central triangle obtained by subdividing t.

4.3.3 Neutral operator

We finally introduce **RB-swap**, a reflexive operator that is neutral with respect to subdivision (i.e. it neither refines nor coarsens). RB-swap takes a pair formed by a red and a blue triangle at the same level l of subdivision, which are adjacent along a red edge, and swaps the diagonal of the trapezoid formed by such a pair, thus obtaining another red-blue pair of triangles at level l (see Figure 4.13). This operator may seem redundant. On the contrary, it is very important for both theoretical and practical reasons, as we will discuss in the next section.

4.3.4 The transition space of RGB triangulations

We now have a set of eleven atomic operators: four split operators, four merge operators, and three swap operators. The family $\mathcal{RGB}_{\Sigma_0}$ of RGB triangulations subdividing base mesh Σ_0 is defined inductively as follows:

- Σ_0 is a RGB triangulation (where all triangles and edges are green, and all entities are at level zero);
- If Σ is a RGB triangulation and Σ' is obtained from Σ by applying one of the eleven atomic operators, then also Σ' is a RGB triangulation.

Following the approach of [KL03], we define the *transition space* of $\mathcal{RGB}_{\Sigma_0}$ as a graph where:

- $\mathcal{RGB}_{\Sigma_0}$ is the set of nodes (where each mesh is taken as an atomic entity);
- There is an arc between two meshes Σ and Σ' if and only if it is possible to transform Σ into Σ' by applying just one atomic operator.

In Figure 4.14, we show the initial fragment of transition space for a single triangle, which shows all possible ways to subdivide such triangle at levels zero and one of subdivision, and all possible transitions among such configurations.

Note that the transition space is not a strict partial order, because of RB-swap operators. So, one may think that we would better define RGB triangulations without using such operator. In fact, a transition space defined without RB-swap would be a strict partial order, but it would also contain minimal elements different from Σ_0. For instance, if we do not use RB-swap, the "fan" configuration depicted in Figure 4.15 becomes a minimal element in the transition space. There are also more practical reasons for using RB-swap. Consider for instance the "strip" configuration shown in Figure 4.16. This configuration has been obtained from an "all green" mesh by applying a sequence of edge split operators. The only possible way to coarsen such a mesh without using RB-swap consists in reversing the

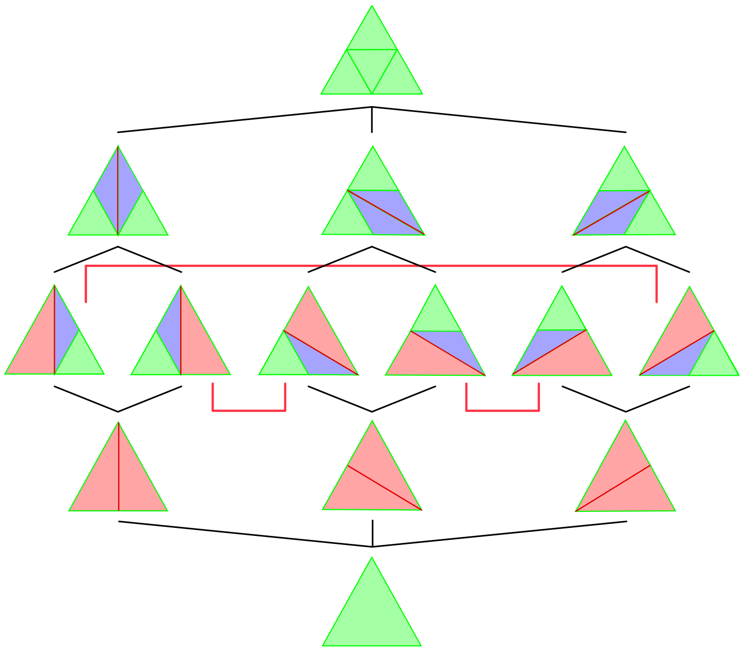


Figure 4.14: *The first few nodes of the transition space for a mesh formed by a single triangle. There we find all patterns that subdivide the triangle between level zero and level one. Arcs in black correspond to transitions through refinement operators (upward) and coarsening operators (downward); arcs in red correspond to RB-swap operators.*

refinement sequence. In other words, it is not possible to remove any vertex v introduced at an intermediate step, without removing also all vertices following it in the strip. On the contrary, as we will show in the following, any intermediate vertex can be removed by applying a single RB-swap followed by a merge, without affecting the other vertices of the mesh. In summary, RB-swap allows us to obtain monotone and more flexible sequences of refinement and coarsening operators.

For the sake of simplicity, in the rest of this section we will assume Σ_0 to be watertight. Generalization of the following results to meshes with boundary is straightforward.

Let Δ_{Σ_0} be the set of all triangles that appear in some mesh of $\mathcal{RGB}_{\Sigma_0}$, and let $\mathcal{T}_{\Sigma_0}$ be the set of all possible (conforming and watertight) triangle meshes that can be built by combining elements of Δ_{Σ_0}. Note that combination of triangles is arbitrary, provided that they match at common edges. We now show that all elements of $\mathcal{T}_{\Sigma_0}$ are RGB triangulations.

Lemma 12 *The transition space of $\mathcal{RGB}_{\Sigma_0}$ spans $\mathcal{T}_{\Sigma_0}$.*

Proof. Let us first analyze the nature of triangles in Δ_{Σ_0}: since they come from meshes of $\mathcal{RGB}_{\Sigma_0}$, each such triangle t is endowed with a color and a level l. For $l > 0$, t must have been generated from one of the eleven local operators, and it must subdivide a parent triangle t' at level $l-1$. Moreover, along the edge(s) internal to t', triangle t can only

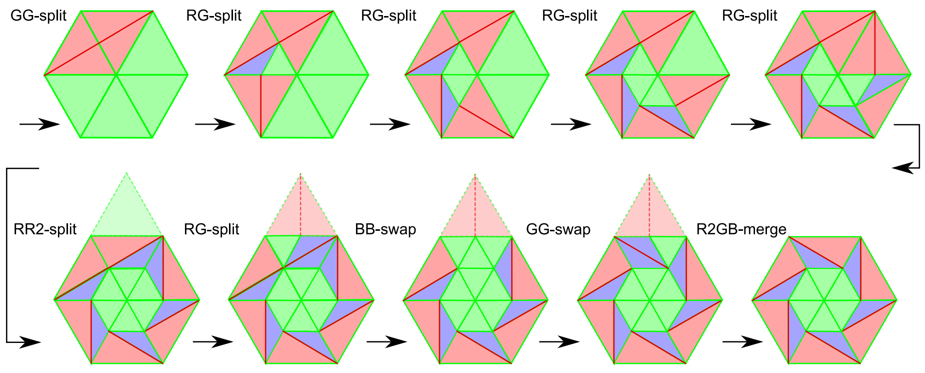


Figure 4.15: *A* fan *configuration. Without RB-swap: the fan is obtained from an "all green" mesh by a sequence of refinement operators followed by a GG-swap and a R2GB-merge; it cannot be simplified without using refinement operators. With RB-swap: the fan is obtained in a smaller number of steps by applying RB-swap right after the RR_2-split; it can be reversed without using refinement operators.*

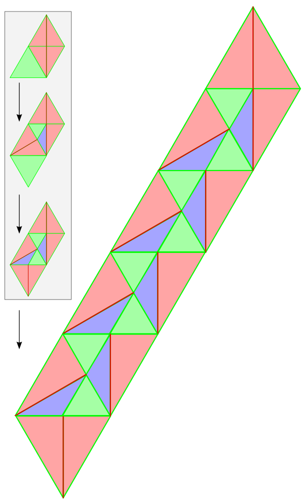


Figure 4.16: *This strip has been obtained from an "all green" strip by applying a GG-split (at its top end) followed by a sequence of RG-splits (proceeding downwards). Without RB-swap, the only possible way to coarsen the strip is by reversing the refinement sequence. With RB-swap, followed by a merge operation, we can remove any intermediate vertex introduced during refinement, without affecting the other vertices.*

be adjacent to other triangle(s) that also subdivide t'. In other words, no matter how we combine triangles to form a mesh Σ of $\mathcal{T}_{\Sigma_0}$, if Σ contains t, then it must contain a group of triangles that subdivide t' exactly. Vertices of triangles in Δ_{Σ_0} also have a level: if a vertex v belongs to Σ_0, then its level is zero; otherwise, v has been generated splitting an edge at level $l-1$, thus its level is l. It is straightforward to see that all triangles incident at that vertex have a level greater than, or equal to $l-1$.

Now let Σ be a mesh of $\mathcal{T}_{\Sigma_0}$. Let us define the level m of Σ to be the maximum level of its vertices. Proof is by induction on m.

If $m = 0$, then Σ can be formed just from green triangles at level zero. Since Σ is watertight and all its triangles come from Σ_0, then we have necessarily that $\Sigma \equiv \Sigma_0$, thus Σ is a RGB triangulation.

Now let us assume all meshes of $\mathcal{T}_{\Sigma_0}$ up to level $m-1$ are RGB triangulations. Given Σ at level m, we know that the level of its green triangles is at most m, while the level of its red and blue triangles is at most $m-1$. We build another mesh Σ' at level $m-1$ as follows: we remove all green triangles at level m and all red and blue triangles at level $m-1$ from Σ; as a consequence, all vertices at level m have been also removed; this means that the holes left after removing such triangles can be filled exactly with green triangles at level $m-1$. Let us call Φ_{m-1} this set of triangles, which are in fact the parent triangles of those we have removed. Now since also triangles of Φ_{m-1} belong to Δ_{Σ_0} then Σ' must belong to $\mathcal{T}_{\Sigma_0}$. Since we have removed all vertices at level m, Σ' is at level $m-1$, thus by inductive hypotesis it is a RGB triangulation.

Now let us consider all vertices at level m that we have eliminated from Σ to obtain Σ'. By construction, they all lie on green edges at level $m-1$ that are shared by pairs of triangles of Φ_{m-1}. Thus all such edges are refinable. Let us consider an arbitrary sequence of edge splits that insert such vertices back into Σ', generating another mesh Σ''. Mesh Σ'' has the same set of vertices of Σ and it coincides with Σ at all green triangles of level smaller than m, and on all red and blue triangles of level smaller than $m-1$. In fact, the edge splits we have performed affect only the triangles of Φ_{m-1}. Now each triangle of Φ_{m-1} has been split by one of the patterns depicted in the middle levels of Figure 4.14 (where we now assume that the root triangle has level $m-1$). Let t be one triangle of Φ_{m-1}, and let us consider the two patterns decomposing t in Σ and in Σ''. Since we have introduced all and only those vertices that were removed, the two patterns may be different, but they subdivide the edges of t in the same way. By referring to Figure 4.14, and comparing the two patterns, we have the following cases:

- If they subdivide just an edge of t, then they must be equal;
- If they subdivide two edges of t and they are different, then it is possible to obtain one from the other by applying an RB-swap;

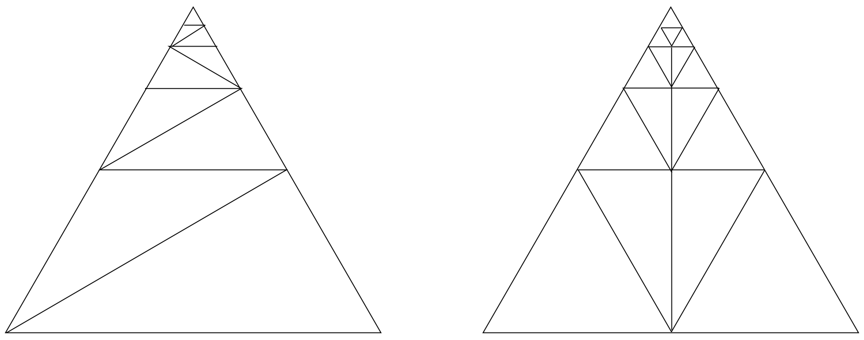


Figure 4.17: *Fast transition of LOD from the base to the apex of the big triangle with RGB triangulations (left) and red-green triangulations (right). In both cases, the same patterns can be nested for an arbitrary number of levels. About twice the number of triangles is necessary in red-green triangulations.*

- If they subdivide all three edges and they are different, then it is possible to obtain one from the other by applying a BB-swap and/or a GG-swap.

All operators listed above only affect triangles that subdivide t, so they can be carried out independently on all subdivisions of triangles of Φ_{m-1}. This means that we can obtain Σ from Σ'' through a sequence of local operators. Concatenating such a sequence with the sequence that transforms Σ' into Σ'', we have a sequence of local operators that transforms Σ' into Σ. Therefore, also Σ is a RGB triangulation.

It is an open question whether or not the same set of triangulations can be generated by using just combinations of the first ten operators, without using RB-swap. So the set of operators we use is sufficient to generate the transition space but we do not claim it to be minimal.

Concerning comparison with other known schemes, note that a uniform "all green" subdivision at any level belongs to $\mathcal{T}_{\Sigma_0}$, therefore it is a RGB triangulation. Also red-green triangulations belong to $\mathcal{T}_{\Sigma_0}$: these are in fact a subset of triangulations made just from green and red triangles. Figure 4.17 presents an example, which shows how RGB triangulations may manage fast transitions of LOD better than red-green triangulations.

Next we state some results useful to ensure that the selective refinement algorithm (described in the next section) does not get stuck in configurations that cannot be either refined or simplified further.

Corollary 13 *Any RGB triangulation can be obtained from Σ_0 by applying a sequence of operators composed just of edge split and swap operators. Σ_0 can be obtained from any RGB triangulation by applying a sequence composed just of edge merge and swap operators.*

Proof. The first statement follows from the proof of Lemma 12 by considering the operators we have used to obtain Σ from Σ'. The second statement follows from the first one by considering that each split operator it inverted by a merge operator and each swap operator is inverted by a swap operator.

Sequences used in Corollary 13 are not always monotone in the span space. In fact, refinement [coarsening] sequences to obtain meshes that contain configurations in the second upper row of Figure 4.14 may require using GG-swap [BB-swap], which is actually a coarsening [refinement] primitive. On the other hand, such configurations are not really interesting: the decomposition of a parent triangle with a configuration that contains two green and two blue triangles is usually better substituted with the standard decomposition made of four green triangles. A mesh containing no configuration made of two blue triangles adjacent along a red edge will be called *stable*; otherwise it will be called *unstable*. In our implementation of selective refinement, we will use unstable configurations just as transitions. We will perform refinement by using just subdivision operators, and coarsening by using just reverse subdivision operators and RB-swap. During refinement, a BB-swap will be forced every time an unstable configuration arises. During coarsening, on the contrary, GG-swap and RB-swap will be used to locally modify the mesh in order to allow a vertex to be removed from a merge operator. The mesh just before applying the merge operator may be unstable, but it will become stable right after it.

We study next the local configurations corresponding to vertices that can be removed during coarsening. Let v be a vertex at level $l > 0$ in a RGB mesh. We say that v is *removable* if and only if all its adjacent vertices are at a level $\leq l$. Since v was introduced by splitting an edge at level $l-1$, by combinatorial analysis we have that the star of triangles surrounding it can have only 28 possible configurations, which are obtained by mirroring from the 18 configurations depicted in Figure 4.18. For each such configuration, the graph in the figure provides a sequence of operators to remove v. Notice that BB-swap is necessary only if we start from one of the unstable configurations. Notice also that the local configurations at the end of sequences (i.e., after vertex removal) are all stable.

Corollary 14 *If a RGB mesh Σ is stable, then:*

1. *Σ can be obtained from Σ_0 by a sequence made just of refinement operators and RB-swaps;*
2. *Σ_0 can be obtained from Σ by a sequence of just coarsening operators and RB-swaps.*

Proof. We prove the second statement first. Let m be the level of Σ. We can obtain Σ_0 from Σ by deleting all vertices of level > 0 level by level, starting at level m. As shown above, if Σ is stable, a vertex can be removed without need to apply BB-swap and the resulting mesh will again be stable. Thus the whole sequence will need just coarsening operators and RB-swap. The first statement follows from the second by considering the inverse operators.

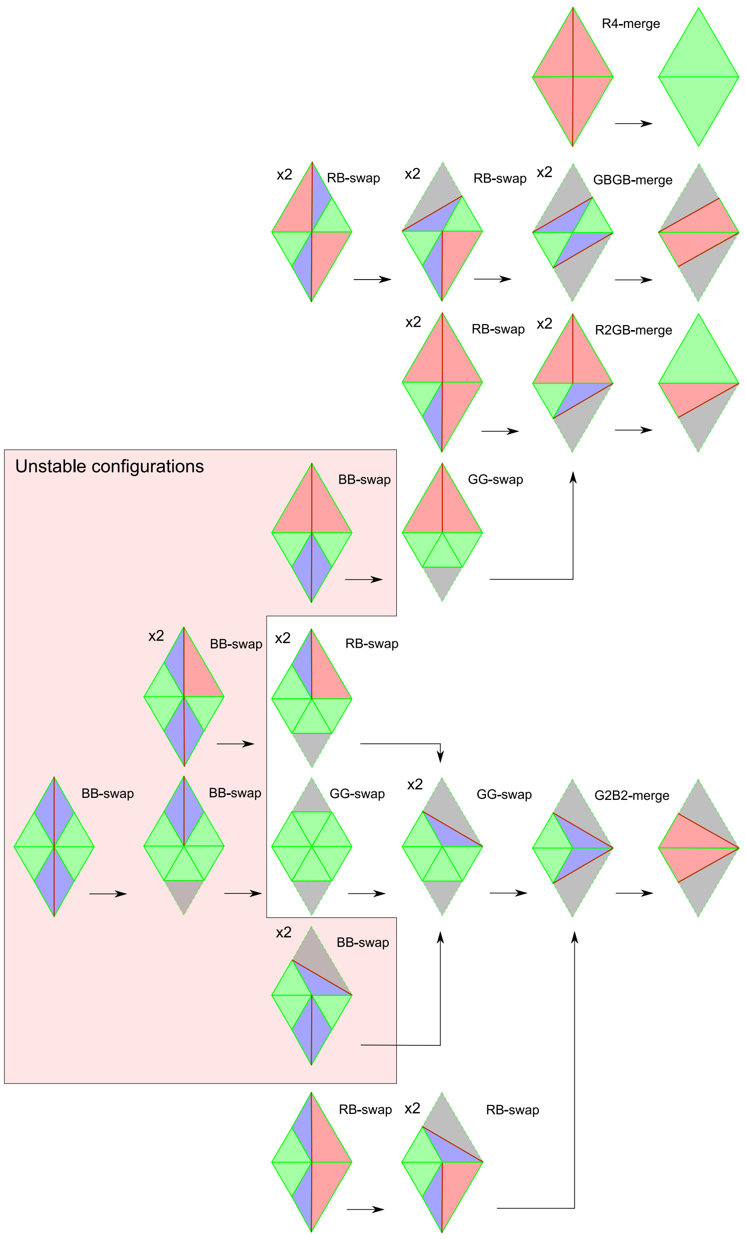


Figure 4.18: *Sequences of operators to remove a vertex. There exist 28 configurations of triangles incident at a removable vertex, obtained by mirroring from the ones depicted in the figure (except those in the last column, which correspond to the configurations after deleting the vertex). Each configuration labeled with* x2 *has a mirror configuration. Triangles in gray correspond to areas of the parent triangles that are not affected by transitions and may be further refined. Except for the unstable configurations, a vertex can be removed without using any refinement operator.*

We now know refinement and coarsening sequences that are monotone in the transition space. Once the star of either a refinable edge, or a removable vertex is known, the sequence of operations necessary to perform the corresponding either refinement, or coarsening operation, respectively, can be retrieved from a lookup table and performed on such a star without affecting the rest of the mesh. These sequences will provide the basic ingredients to implement the selective refinement algorithm described next.

4.3.5 Implementation details

A RGB triangulation can be maintained in a standard topological data structure for triangle meshes. One possibility is using three dynamic arrays, for vertices, edges, and triangles, respectively, with a garbage collection mechanism to manage reuse of locations freed because of coarsening operators. The following simplified version of the incidence graph [Ede87] can be adopted: for each triangle, links to its three edges are maintained; for each edge, links to its two vertices and its two adjacent triangles are maintained; for each vertex, just a link to one of its incident edges is maintained (this is sufficient to compute the star of a vertex in optimal time).

If the mesh contains n vertices, we can roughly estimate its number of triangles and edges to be about $2n$ and $3n$, respectively. By assuming unit cost to represent a pointer or a number, the total cost for topological information in this base structure is about $19n$, and an additional $3n$ is necessary to maintain the coordinates of vertices.

This data structure is extended as follows. For each vertex, we maintain: its level of insertion and the current level of its control point (one byte is sufficient for both); two triples of coordinates rather than just one (position at time of insertion and limit position); six flags to keep track of neighbors that have given their contribution for computing the limit position (see Section 4.4.2).

For each edge and each triangle, we maintain its color and its level. Edges come in just two colors. It is convenient to encode two different types of red triangles, and two different types of blue triangles, depending on their orientation: a red triangle will be said to be either Red$_{RGG}$, or Red$_{GGR}$, depending on the colors of its edges, traversed in counter-clockwise starting at the vertex with the highest insertion level; a blue triangle will be said to be either Blue$_{RGG}$, or Blue$_{GGR}$, depending on the colors of its edges, traversed in counter-clockwise starting at the vertex with the lowest insertion level. We thus use five different colors for triangles: two for red triangles Red_{RGG} and Red_{GGR}, two for blue triangles $Blue_{RGG}$ and $Blue_{GGR}$, and one for green triangles. Since three [one] bits are sufficient for the color of triangles [edges], and levels in subdivision are usually not many, one byte is sufficient to store both color and level.

Summing up, by assuming one unit of cost to be equal to four bytes, we have an additional cost of $4.75n$. This corresponds to a 25% overhead with respect to the base data structure.

Since selective refinement is meant to be used dynamically, a caching mechanism can be used to save vertices and their related control points when edge merge operators remove them from the mesh. In this case, each vertex must receive a unique label which can be computed from location codes of triangles in the hierarchy [LS00]. An additional cost of two units per vertex is necessary in the data structure to maintain the location code. A LRU policy can be adopted to manage the cache. A vertex in cache is restored together with its control points (insertion and limit), when an edge split operator reinserts it in the mesh. The use of cache saves the cost of fetching the vertices in the stencil of an odd vertex, and possibly the cost of computing the limit position.

4.4 Adaptive Catmull-Clark subdivision

In this Section, we will use our method to edit the LOD of a mesh through operations that modify the mesh locally, while maintaining it compatible with the Catmull-Clark scheme (henceforth called *the standard subdivision*). Compatibility is defined as follows. Given a base mesh Γ_0, then:

1. An adaptive mesh Γ built starting at Γ_0 may contain all and only those vertices that appear in the standard subdivision of Γ_0;
2. If all vertices of a given face f appearing in a the standard subdivision of Γ_0 belong to Γ, then either f, or a subdivision of it also belongs to Γ;
3. If Γ contains a vertex v introduced at level l in the standard subdivision, then the control point $p^l(v)$ will be the same both in Γ and in the standard subdivision. If Γ contains also all vertices in the standard (even) stencil of v at level l, then for any $k \geq l$ the control point $p^k(v)$ will be the same both in Γ and in the standard subdivision.

Our mechanism provides a suitable topological basis to implement an adaptive scheme for the quadrisection pattern. In particular, a uniform subdivision at a given level l computed incrementally by adding all its vertices through our scheme will be both topologically and geometrically coincident with the standard subdivision at level l, hence also the limit surface will be the same.

Note that the meshes refined selectively as described in the previous section naturally fulfill requirements 1 and 2. Thus in the following we will concentrate on requirement 3.

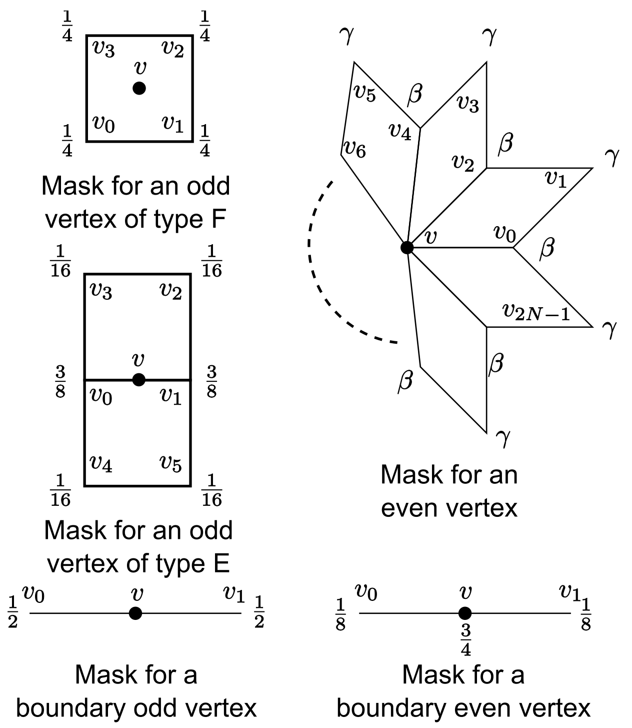


Figure 4.19: *The stencils of the Catmull-Clark subdivision. Numbers are weights of vertices in the linear combination: n is the valence of the even vertex;* $\beta = \frac{3}{2N}$ *and* $\gamma = \frac{1}{4N}$.

4.4.1 Catmull-Clark subdivision

The Catmull-Clark subdivision [CC78] is an approximating scheme for subdivision surfaces that can be applied to any polygonal mesh and converges to a C^2 surface. The subdivision pattern is quadrisection. A new vertex v introduced at level $l+1$ of subdivision is called an *odd* vertex, and the position of its control point $p^l(v)$ at level $l+1$ is computed as a weighted average of control points of vertices surrounding it that belong to level l, according to the stencils and weights depicted in Figure 4.19.

Vertices already present at level l, called the *even vertices* are relocated at level $l+1$, with a weighted sum of their position and the positions of their edge and face neighbors at level l, according to the stencils depicted in Figure 4.19. Therefore, for each vertex v introduced at level l, there exist an infinite sequence of control points $p^l(v), p^{l+1}(v), \ldots, p^\infty(v)$, that define the positions of v at level l and all successive levels, $p^\infty(v)$ being the position of v on the limit surface.

In principle, it is possible to adopt either exact methods [Sta98b], or fast approximated methods [LS08] for the direct evaluation of the limit position of any point of a subdivided mesh. However, evaluation requires that: the parametric coordinates of each point, with respect to the face of the base mesh containing it, are known; and the geometry of all

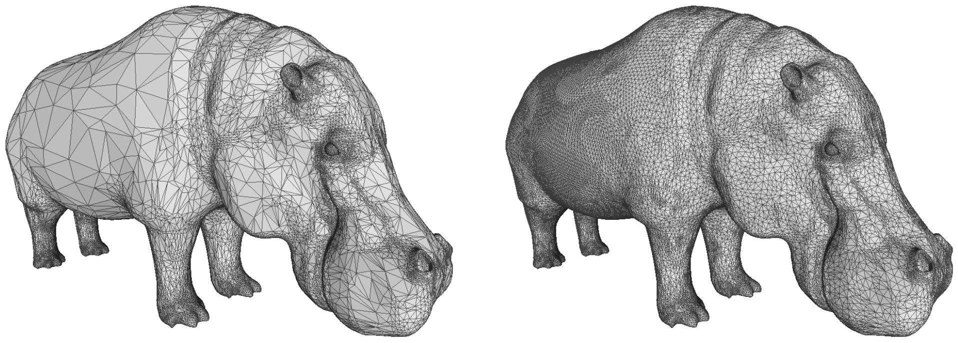


Figure 4.20: *The original mesh contains triangles of very different size. Selective refinement is run on the whole object until a mesh of 100,000 triangles is obtained, giving higher priority to the refinement of longer edges. The triangles in the resulting mesh are much more uniform.*

control points of the base mesh in the neighborhood of such a face are retrieved. Since we only encode the adaptively refined mesh, retrieving such information may be unpractical, involving traversal of a large area.

We therefore develop an alternative method that makes use just of close neighbors of any given vertex. Any control point $p^k(v)$ for a vertex v introduced at level l, with $0 \leq l < k$ can be computed directly just from the positions p^l of v and of all its even and odd neighbors at level l. In Section 4.4.2.1, we derive a multi-pass closed form for computing directly control points at an arbitrary level k, which provides a basis for the effective and efficient computation of correct control points in an adaptively subdivided mesh.

4.4.2 Computing control points

Since we work selectively, it is not trivial to find the right vertices to use for a stencil, and to compute the control points at their proper levels. In this section, first we introduce some tools for the computation of control points, and next we discuss how to use them to maintain geometry up-to-date. In Subsection 4.4.2.1 we present a multi-pass formula for the Catmull-Clark subdivision, which allows us to compute in closed form the control point of any vertex at any level of subdivision. On this basis, in Subsection 4.4.2.2 we introduce a mechanism for computing the control point of a given vertex incrementally, as its neighbors are inserted into the mesh. Updates to control points must be made for odd vertices during refinement, and for even vertices both during refinement and during coarsening. We discuss such operations in detail in Subsections 4.4.2.3, 4.4.2.4 and 4.4.2.5, respectively.

4.4.2.1 Multi-pass subdivision

Let us consider a vertex v inserted at level l. If $l = 0$ then v belongs to the base mesh and its geometry $p^0(v)$ is known, otherwise its control point $p^l(v)$ is computed on the basis of stencils for odd vertices (see Figure 4.19).

By applying the concept of multi-step subdivision rule [Kob00] to the analysis of the Catmull Clark scheme developed in [Sta98b], we derive equations that compute the control point of v and any level k on the basis of its initial position and on the positions of its neighbors at level l.

Lemma 15 *[PP10] The control point $p^k(v)$ of an internal vertex v, inserted at level l, for $k > l$ is given by*

$$p^k(v) = s_{11}^k v + s_{12}^k \sum_{i=1}^{N} v_{2i} + s_{13}^k \sum_{i=1}^{N} v_{2i-1} \tag{4.1}$$

where s_{11}^k, s_{12}^k and s_{13}^k are defined below, and vertices in the summations are the edge and face neighbors of v at level l, respectively.

$$\begin{aligned} s_{11}^k &= \frac{\sqrt{\alpha_n}(11N-35)(\beta_{n,k}-\gamma_{n,k}) + 5\alpha_n(\beta_{n,k}+\gamma_{n,k})}{2\cdot 8^k \lambda_n} + \frac{N}{N+5} \\ s_{12}^k &= \frac{-2\sqrt{\alpha_n}(4N-16)(\beta_{n,k}-\gamma_{n,k}) - 4\alpha_n(\beta_{n,k}+\gamma_{n,k})}{2\cdot 8^k \lambda_n N} + \frac{4}{N(N+5)} \\ s_{13}^k &= \frac{-\sqrt{\alpha_n}(3N-3)(\beta_{n,k}-\gamma_{n,k}) - \alpha_n(\beta_{n,k}+\gamma_{n,k})}{2\cdot 8^k \lambda_n N} + \frac{1}{N(N+5)} \end{aligned}$$

with $a = \frac{3N-7}{N}$, $b = \frac{\sqrt{\alpha_n}}{N}$, $\beta_{n,k} = (a+b)^k$, $\gamma_{n,k} = (a-b)^k$, $\lambda_n = 5N^3 - 5N^2 - 101N + 245$

Similarly, if v is a boundary vertex we have

$$p^k(v) = sb_{11}^k v + sb_{12}^k (v_0 + v_1). \tag{4.2}$$

Note that, by computing the limits for $k \to \infty$ of equations 4.1 and 4.2 we obtain the well known limit positions of internal and boundary vertices, respectively:

$$p^\infty(v) = \frac{N}{N+5} v + \frac{4}{N(N+5)} \sum_{i=1}^{N} v_{2i} + \frac{1}{N(N+5)} \sum_{i=1}^{N} v_{2i+1}$$

and

$$p^\infty(v) = \frac{2}{3} v + \frac{1}{6}(v_0 + v_1).$$

4.4.2.2 Incremental summations

In a standard subdivision, if a vertex v is inserted at a level $l > 0$, two of its edge neighbors already belong to the mesh, while the other two edge neighbors as well as its four face neighbors are inserted at the same time and level as v, and the control points at level l of all such vertices are computed. In an adaptive subdivision, some neighbors of v might be inserted at a later time. Therefore, it is not always possible to apply Equation 4.1 right after inserting v in the mesh.

We use an approximation of Equation 4.1 as long as not all neighbors of v are available. In the data structure encoding the mesh, for each vertex v inserted at level l, we store its control point $p^l(v)$ and we reserve two other fields to store the sums $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$ of control points at level l of its edge and face neighbors, respectively. We also store two counters of contributions already stored in $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$. At startup, we fill such fields for all vertices of the base mesh.

For a generic vertex v inserted at level $l > 0$, its control point $p^l(v)$ is computed and stored when creating v as an odd vertex (see Section 4.4.2.3), while $\mathrm{SUM}_f(v)$ and $\mathrm{SUM}_e(v)$ are computed incrementally, as the control points of neighbors of v become available. Initially, we set both $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$ to $4 * p^l(v)$. Every time a control point $p^l(v_i)$ for a neighbor of v at level l becomes available, its value is accumulated by substituting it to an instance of $p^l(v)$ in either $\mathrm{SUM}_e(v)$ or $\mathrm{SUM}_f(v)$, depending on the position of v_i in the stencil of v.

A vertex v is represented in the mesh with a control point at level k, where k is the smallest level of its incident edges. As long as the correct value of the two sums is not known, v will be represented with an approximation of its position computed by Equation 4.1 with the values of sums currently stored at $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$. Note that the quality of the approximation progressively increases as new control points are inserted, and the formula becomes exact as soon as all contributions from neighbors of v become available.

4.4.2.3 Control points for odd vertices during refinement

In the sequel, for the sake of brevity, we will address only internal vertices. Boundary vertices are handled similarly.

Let v be a vertex of type E introduced at level $l + 1$ of subdivision by splitting a refinable edge e at level l. In order to compute control point $p^{l+1}(v)$, we need to fetch the vertices in the stencil of v at level l.Referring to Figure 4.19, vertices v_0 and v_1 of the stencil of v are the endpoints of edge e, so they are found immediately. The other vertices of the stencil are not trivial to fetch since the faces defined by v_0, v_1, v_2, v_3 and v_1, v_0, v_4, v_5 may have been refined further. This can be done with a combination of move and rotate operators. Since

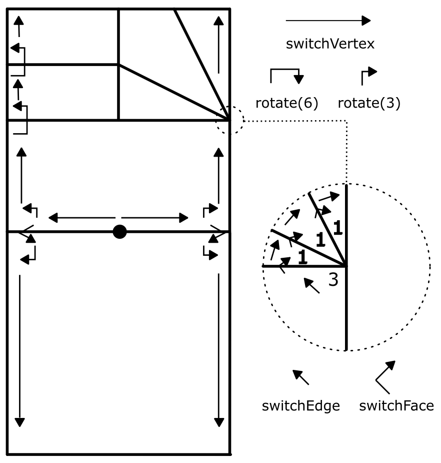


Figure 4.21: *An example of mesh traversal to fetch the stencil of an odd vertex v. A spanning tree of the neighborhood of v is traversed through move and rotate operations. Move operations are decomposed into sequences of switchVertex, rotate and switchFace; rotate operations are decomposed into sequences of switchEdge and switchFace.*

these operators are not influenced by editing operations, then the stencil will be fetched correctly also if the mesh has been refined (see Figure 4.21).

Any odd vertex v of type F is inserted by operator 2a/b-split together with a vertex v' of type E, and the stencil of v is in fact a subset of the stencil of v', so we do not need to fetch it separately.

If the control point at level l is not available for some vertex v_i in the stencil of v, then recursive refinement must be triggered to insert in the mesh all neighbors of v_i at its insertion level l_i. To this aim, it is necessary to recursively split the edges in the neighborhood of v_i until all faces incident in it are of level greater or equal to l_i. This can be done with a recursive split of all standard edges incident at v_i and having a level lower than l_i, followed by an analysis of the incident faces. If an incident face f is of type 4, then a recursive split of the standard edge with lower level of one of the two triangles that share an extra edge with f is necessary.

This can be done simply by traversing in counter-clockwise order the edges incident at v_i. Every time an edge at a level lower than l_i is found, we recursively split it. The algorithm halts when a complete scan of the edges is performed without performing any split. The additional splits required for faces of type 4 can be performed by traversing the faces and splitting a single edge for every face of type 4 found. Note that a regular vertex may have standard incident edges that differ for at most two levels, thus the number of new vertices to be computed during this operation is usually quite small.

Computation of control points may trigger some over-refinement of the mesh, which might be not necessary to fulfill LOD requirements. This is similar to what happens in other adaptive subdivision schemes [Kob00, SHHG01, VZ01]. Since we wish to avoid over-refinement of the result, edge splits performed during computation of control points, which are unnecessary according to LOD requirements, are marked as temporary and inserted in a queue. A temporary vertex becomes permanent in case one of its incident edges undergoes a standard edge split. At the end of selective refinement, this queue is scanned, and all vertices that are still temporary are removed by performing corresponding edge merge operators.

After v has been inserted at level l, we must find all possible vertices that give contribution to compute summations $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$. This is done again with a navigation algorithm based on topological angles. The stencil that we want to identify can be incomplete, i.e. some vertices may not be present in the mesh, and we have no a priori information on the level of refinement the portions of stencil currently available. We traverse the stencil with a breath-first strategy, starting at v and navigating on subsets of all possible paths that we can use to reach a vertex v_i of the stencil. The algorithm starts by identifying the standard edges at level l_i incident at v. For each edge e_i, connecting v with another vertex v_i, the algorithm navigates the stencil using rotate and move operators until either v_{i-1} and v_{i+1} are found, or it is detected that they are not present in the mesh.

For each candidate v_c found, we must check what kind of contribution is needed:

1. If the level of v_c is l, we simply accumulate $p^l(v_c)$ on $\mathrm{SUM}_e(v)$ or $\mathrm{SUM}_f(v)$, depending if v_c being an edge or a face neighbor of v, respectively;
2. If the level of v_c is $< l$ and all the neighbors in the even stencil of v_c are present, we compute the correct control point, and we accumulate it on $\mathrm{SUM}_e(v)$ or $\mathrm{SUM}_f(v)$, exactly as before.

In both cases, we keep track of the fact that v_c has given its contribution to v by keeping a counter for each vertex. This is necessary both to understand when all neighbors have given their contribution and to avoid accumulating a contribution twice, since a vertex can be deleted from the mesh and introduced again at a later time. This operation is performed also in the opposite direction, since every candidate can need the contribution of v to compute its own summations.

As soon as a vertex v at level l receives the contribution from all the vertices in its stencil, we cal it **complete** and its correct control point can be computed at every level $\geq l$. In this case, we immediately provide its contribution to all vertices of level $\geq l$ that are present in its stencil.

4.4.2.4 Control points for even vertices during refinement

The case of even vertices is simpler. When a new vertex v is inserted to split an edge e, this may affect the control points of the candidate vertices in its neighborhood. For each such

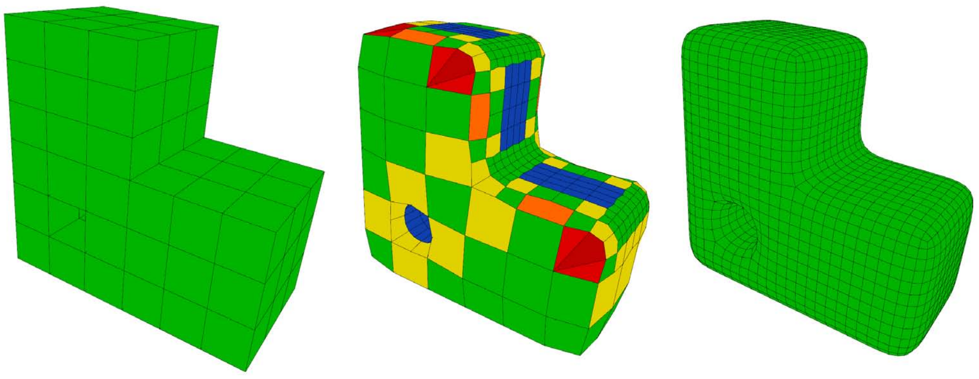


Figure 4.22: *A base mesh (a) is adaptively refined to smooth three sharp edges and round the hole (b); the same mesh uniformly refined by using the standard Catmull Clark subdivision scheme (c).*

vertex v_i, if the minimum level of edges incident at v_i has increased to level k, then the current position of v_i is updated to $p^k(v_i)$. Otherwise no action is required. Note that value $p^k(v_i)$ will be approximated as long as v_i is not complete. This approximation reduces the popping effect during selective refinement and it converges to the correct position as more points in the stencil of v are inserted.

4.4.2.5 Control points during coarsening

The removal of a vertex v must undo the updates to the contibution counter and to the contribution accumulators SUM_e and SUM_f for every vertex in the stencil of v. This is done by straightforward adaptations of the algorithms described above. We do not remove contributions from complete vertices, which already have sufficient information to compute their correct control points at all levels.

4.4.3 Results

We have developed an interactive application that allows us editing the LOD of a subdivided mesh by means of a brush tool. Figure 4.3(a) shows a rough polygonal model designed in Blender by merging a cylindrical handle to a lathe object. In Figure 4.3(b) the mesh has been edited by a few strokes of brush in order to refine the joints between the handle and the bottle, as well as to improve its overall shape. A coarsening brush as well as single local refinement operations have been used for adjusting over-refined parts and fine-tuning. Note that the pot-bellied part has been refined anisotropically in order to better approximate shape in the direction of higher curvature. Transitions across different LODs involve pentagonal faces.

Figure 4.22 shows a simple L-shaped block with a hole. We have refined the hole anisotropically, we have smoothened two convex and one concave edge with two levels of subdivision, and we have refined anisotropically a strip traversing the top faces of the object with one level of subdivision. The different colors represent the different types of faces (see Figure 4.6): green, blue and dark red faces are quads; yellow and orange faces are pentagons; and light red faces are triangles. As it can be seen by comparing Figures 4.22 (b) and (c) the adaptively refined mesh contains a much smaller number of faces than the uniform Catmull Clark subdivision at level two, while it approximates well the shape in the regions of interest. In fact, it can be seen that in the adaptively refined mesh, the most refined parts are identical to the corresponding parts in the uniformly refined mesh.

A similar construction can be applied to our adaptive triangular subdivision scheme presented in Section 4.3 to produce surfaces that are compliant with a loop subdivision scheme. For the sake of brevity, we omit the description, that can be found in [PP09b]. Results obtained with this construction are shown in Figures 4.1 and 4.20.

4.5 Semi-regular remeshing via adaptive subdivision

In this section we present a remeshing method, which can be either user-assisted, or fully automatic, and it is able to produce a semi-regular adaptively refined mesh representing a given shape. The method starts with a sketched base mesh, which is refined and fitted to the input shape. Selective refinement can be either user-assisted or error-driven.

4.5.1 Generation and fitting of a base mesh

The target shape is given as a mesh T at high resolution. We start form a coarse quad mesh M, providing a drastically simplified, yet consistent, version of the shape. Mesh M can be generated either manually, or automatically by means of a simplification method, such as the one in [TPC$^+$10]. The overall assumption is that M is roughly *projectable* to T along surface normals, i.e., the projections of points of M roughly subdivide the surface of T into patches with the same connectivity of M. There is no need that M achieves perfect projectability, since this will improve during subsequent refinement. For a shape of genus zero without large protrusions or cavities, such as the head in Figure 4.4, a base mesh as simple as a meshed cube can be used to the purpose; for more complex shapes with non-zero genus and/or having important protrusions and cavities, such as the horse of Figure 4.23, or the gargoyle of Figure 4.24a, a more elaborated base mesh may be necessary.

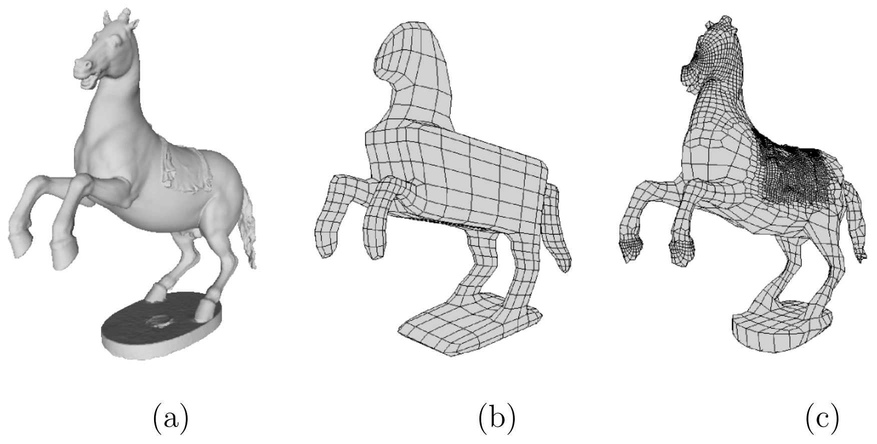


Figure 4.23: *Remeshing the Rampart dataset: (a) target model; (b) base mesh manually sketched; (c) remeshing of the dataset that highly refines the head and the saddle.*

The connectivity of M affects final meshing, since the directions of edges will be preserved during refinement.

Given M, we fit it to the target mesh T through spatial projection: for every vertex v of M, we compute its normal $\mathbf{n}_v$; we shoot a ray in direction of $\mathbf{n}_v$ and another ray in the opposite direction; and we displace v to the closest point hit from a ray. Ray shooting is supported by means of a spatial index that contains all faces of mesh T, which is created once and for all at the beginning of computation. We use a simple regular grid, but more complex and efficient data structures may be adopted for huge datasets [Sam05].

This fitting procedure works well in most cases, supporting interactive editing speed. We have noticed stability problems only if the initial fitting of mesh M to T is very poor, i.e., M lacks entire features of the shape, or the shape contains very thin features. Better and more stable results can be obtained if a parametrization of T is available, which is defined on M. In this case, ray shooting is not necessary, and both surface mapping and smoothing (see also next subsection) can be done via parametrization.

4.5.2 Editing operations and tangent space smoothing

Editing operations described in Section 4.2 are applied to selectively refine mesh M. For every refining operation, a vertex of type E is created, which is initially placed at the midpoint of the splitting edge; an additional vertex of type F is also created in pattern P3, which is initially placed at the barycenter of the subdivided face. The normal of each new vertex is estimated, and the vertex is displaced to its projection to the target mesh, as before. Interactive refinement is supported through brush tools, similarly to the case of adaptive subdivision. In this case, either refinement or coarsening operations are applied, up to a prescribed level of subdivision, in the area spanned by the brush. Automatic error-driven refinement is explained in the next section.

To increase the quality of meshing, smoothing is performed in tangent space, by displacing the position of vertices tangentially on the surface of T. The aim of tangential smoothing is to obtain quad faces with a better (i.e., more rectangular) shape. After each local operation, we consider all vertices in the 2-ring of faces affected by the operation. For each such vertex v, we execute a step of Laplacian smoothing - i.e., v is displaced to the barycenter of its neighbors - followed by a re-projection to the target mesh. In case a conformal parametrization of T has been defined on M, computation can be carried out more easily and accurately in parametric space.

4.5.3 Error-driven remeshing

While a user-assisted approach may be useful in many contexts, some times a fully automatic algorithm is to be preferred. We define the approximation error associated to every face f of the current mesh M as the RMS difference between f and the patch spanned by its projection on T:

$$\frac{1}{Area(f)}\sqrt{\int_f (p-\phi_T(p))^2 dp},$$

where function ϕ_T provides the normal projection of a given point of M to the target mesh T. Computation is discretized on a set of samples selected uniformly on each face f:

$$\frac{1}{k}\sum_{i=0}^{k} |s_i - \phi_T(s_i)|$$

where k is the number of samples, and s_i is a sample point inside f. To obtain a uniform sampling, the number of samples per face f depends on its area:

$$k = \frac{Area(f)}{Area(M)} * v_t$$

with $Area(f)$ the area of f, $Area(M)$ the total area of mesh M, and v_t the number of vertices of the target mesh T.

To sample triangles, we pick points on the unique plane that contains the triangle. For quads, we pick samples in the bilinear patch that interpolates the four vertices. For pentagons, we first triangulate and then sample each triangle separately.

Faces of M are maintained in a priority queue and refinement is perfomed iteratively. The face with the largest error is extracted from the queue at each iteration, and it is refined

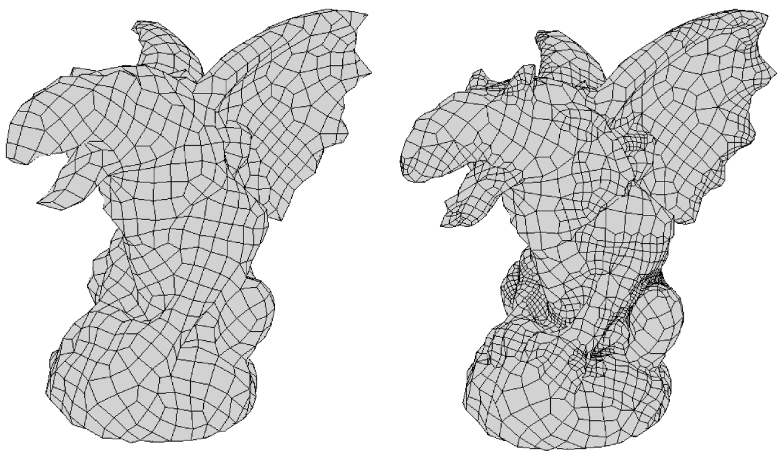


Figure 4.24: *Fully automatic remeshing of the Gargoyle dataset. (a) The base mesh generated using [TPC+10]. (b) Remeshed model.*

with a local operation. For a face f of type 0, 1, 3, or 5, at level l, its edge e at level l yielding the largest error is selected for split. The error associated to an edge is computed similarly by sampling along it. For a face of type 2 at level l, its standard edge at level l is split. For a face f of type 4, its adjacent face of type 2 yielding the largest error is refined, and a vertex of type F is inserted consequently inside f.

After each iteration, error is computed for the new faces resulting from refinement, and they are inserted in the priority queue accordingly. Iteration may be carried out until either a certain budget of faces has been reached, or when error gets below a given threshold, depending on user's needs. A result of this automatic remeshing procedure is shown in Figure 4.24. The original target mesh has been simplified with [TPC+10] to obtain the base mesh M, and then it has been remeshed with our algorithm.

4.6 Concluding remarks

Our schemes have several advantages over both classical and adaptive subdivision schemes, as well as over CLOD models: they supports fully dynamic selective refinement; they are better adaptive than previously known schemes based on the one-to-four triangle split pattern; they does not require hierarchical data structures; selective refinement can be implemented efficiently by plugging faces inside the mesh, according to rules encoded in lookup tables, thus avoiding cumbersome procedural updates.

We believe that this approach to adaptive subdivision may give valid substitutes or complements to standard subdivision for solid modelers and simulation systems. Combined with reverse subdivision techniques, it may also offer a valid alternative to CLOD models for free-form objects in computer graphics.

Chapter 5

Interference-Aware Geometric Modeling

Many applications of geometric modeling require constructed shapes to be physically realizable. Shapes created using computer-aided design systems need to be manufactured; if the model is used in a physical simulation—either for special effects, animation, or engineering analysis—its geometric properties should be consistent with those of a real object. One significant impediment to this consistency is intersections within or between modeled objects. These (self-)intersections appear as glaring artifacts, and eliminate the ability to use the final model further down many software pipelines.

Despite its importance, there has been little research on the modeling of surface interference for geometric design. While a number of recent algorithms for collision *detection* are sufficiently fast for interactive applications, collision *response* in the context of geometric modeling has not received much attention. Once interference has been found through detection, the response algorithm is responsible for modifying positions and trajectories to remove it.

Most existing contact response algorithms are developed for physically-based simulations and try to follow the logic of physical laws. A a result, these methods are either too slow (due to strict physical requirements) or cannot be extended for application to general modeling scenarios. Meanwhile, free-form shape design is primarily concerned with surface quality, interactive control, and aesthetics, and typically is not governed by physical equations. The few works on interactive surface deformation that do handle collisions do so as a side effect of their particular modeling paradigm, and are thus limited to those specific tools to model intersection-free surfaces.

In this Chapter, we present a response algorithm for preventing interference between and within meshed surfaces, formulated in a purely geometric setting. Objects do not have

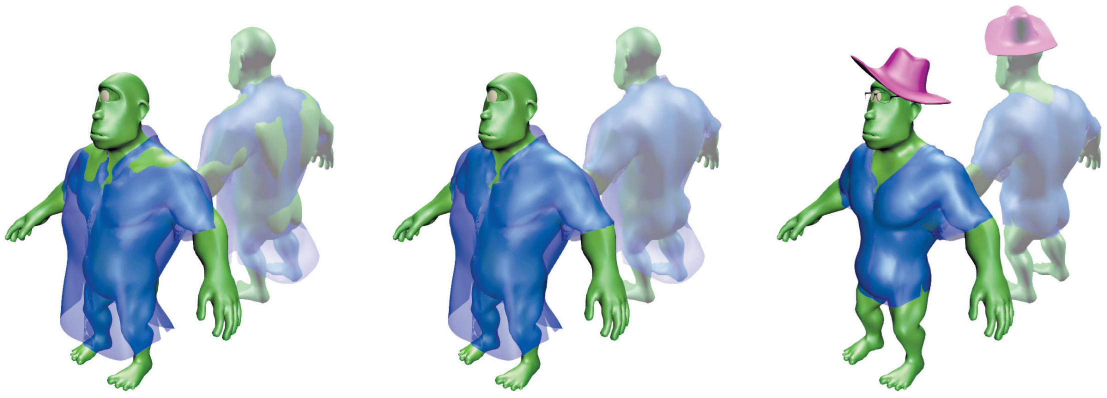


Figure 5.1: *Interference-aware modeling greatly simplifies many complicated modeling tasks. We interactively fit the ogre with a shirt made for a human. We use our ability to fix existing intersections in a mesh and then "shrink-wrap" the shirt on the ogre, ensuring a perfect fit.*

any physical attributes, and their deformation is not necessarily driven by forces. We formulate non-interference constraints on *space-time interference volumes* (STIVs), defined as volumes in space-time traced out by parts of the surface after interpenetration occurs. The advantages of this formulation are two-fold: first, a trajectory-based method can robustly handle problems with thin features, boundaries, and rapid large deformations, without restrictions on the type of geometry. Second, by formulating the non-interference constraint to be zero for each STIV, rather than at the geometric primitive level, the dimension of the numerical problem to be solved is vastly reduced, improving response speed and robustness. Surfaces, while usually deformed through the manipulation of a low-dimensional modeling subspace, can deform and intersect in complex ways; a response algorithm must interactively handle this type of interference in geometric modeling, while reliably resolving and maintaining surfaces free of intersections. Our proposed method has the following features:

- **Independent of deformation model.** Our algorithm is not tied to a specific modeling paradigm; We demonstrate its applicability for several types of surface deformation techniques, with the steps outlined to interface with any others required. it can resolve intersections while keeping the surface in the same subspace (*e.g.*, a subdivision surface remains a subdivision surface with the same controls, or a mesh modified using Laplacian editing still minimizes the same energy). The only input the algorithm requires for a specific deformation technique is the gradient of the function mapping control handles to surface point positions.
- **Handles general geometric data.** Our algorithm is able to robustly handle large deformations, complex intersections, sharp features, as well as self-collisions within a surface. It is capable of processing intersections between surfaces with boundary

and large numbers of disconnected components.

- **Controllable.** Different modes of response to collisions suitable for modeling applications fit into our framework. For example, we allow the user to specify whether objects move rigidly and maintain their shape or deform due to interference.
- **Fast.** Interference detection typically dominates runtime costs. We treat detection as a black-box, so our interference algorithm can freely leverage state-of-the-art research in collision detection methods, which currently allow for interactive editing of large models.

We demonstrate the applicability of our method on a variety of scenarios using a range of modeling techniques, including subdivision surfaces, free-form deformation, and Laplacian surface editing. We demonstrate that in many cases, STIVs can also resolve self-intersection in existing meshes, making it easier to use our technique with existing geometry. We are able to achieve interactive rates for a number of realistic geometric modeling scenarios (Section 5.6 discusses performance in greater detail).

Extending existing modeling environments with our method allows artists to easily create shapes that contain no intersections, so that they not only are free of unsightly artifacts, but also can be used in a pipeline where this property is a requirement, such as in physical simulation, mechanical engineering or manufacturing.

5.1 Related work

While in this Chapter we are concerned with the domain of geometric modeling, we would be remiss to neglect the vast literature that exists for simulating contact. Below we give background on intersection and contact handling in simulation and modeling literature.

Collision detection. Collision detection is usually treated separately from collision response. This paper focuses on the response algorithm, so we refer the reader to a recent survey ([TKZ+04]) and book ([Eri04]) which take a comprehensive look at collision detection techniques. We note that the literature has long recognized the need for so-called *continuous time* formulations in detecting interference, which we complement by presenting an appropriately paired response algorithm that operates in the same space-time domain. For example, Cameron [Cam90] formulated continuous time detection between rigid bodies as a problem in space-time. Provot [Pro97] presented a general method for deformable surfaces, where the roots of cubic polynomials (in time) are found to detect collisions. However, these works focused on the 4-dimensional *detection* problem, without offering a matched *response* model.

We base our interference detection on the hash grid approach of Teschner *et. al.* [THM$^+$03], but our algorithms are independent of the specific detection method used. Recent work has shown significant speed in general collision detection, with Pabst *et. al.* [PKS10] achieving speeds of fractions of a second for the detection of complex collisions in large materials. Promising research by Tang *et. al.* [TMT10, TMLT11] shows continual development in real-time interference detection as well.

Physical simulation. The study of contact mechanics began over a century ago with the analytical description of forces between spherical elastic bodies [Her82]. The following years were used to further develop and understand the nature of contact problems [Fic65]. With the arrival of computers came the numerical analysis of computational contact problems. However, over time the growing computational capabilities have been met with the increasing complexity of applications. As a result, algorithms have become a three-way tug-of-war between robustness against interpenetrations, algorithmic speed, and physical accuracy. Much work in computer graphics has downplayed the latter for the sake of the two former, and we eagerly explore this route. Most of the work on *response* to collisions and intersections is done in the context of physical simulation, and we build on some of the techniques from this domain. However, our problem is different: on the one hand, we do not have stringent requirements on physical accuracy, especially in the context of dynamic effects (we only need natural and intuitive behavior suitable for geometric modeling applications). On the other hand, physically-based response (*e.g.*, penalty forces) does not fit well into a pure geometric framework, and the requirements for robustness, generality and efficiency are more restrictive, compared to a typical physical simulation. We can broadly divide physically based response methods into three categories: penalty forces, impulses, and constraints.

Penalty forces are additional forces acting to separate the surfaces, or maintain contact. These forces are well-understood in the contact mechanics community [WL07] and were introduced to computer graphics by the early work of Terzopoulos *et. al.* [TPBF87]. While penalty forces are easy to implement, difficulties often arise in adjusting the stiffness of these forces—too weak, and objects simply pass through one another, too strong, and the system becomes poorly conditioned. Harmon *et. al.* [HVS$^+$09] address this problem through asynchronous timestepping, albeit at a significant computational cost. For our application, even if we were to cast geometric modeling as a physical problem in order to introduce forces, we cannot afford the high runtimes necessary to ensure robustness.

An alternative is to view collision response as an instantaneous reaction (an *impulse*) representing an abrupt change in momentum. This means for physical accuracy the system must be timestepped up to the time of contact, resolved, and then restarted with new initial conditions [WB01b]. While using sequentially applied impulses for response is possible [MW88, MC95], impulses quickly becomes computationally challenging even with a small number of collisions. Furthermore, impulses are known to not always converge, and we need a method that reliably resolves intersections. Many circumvent these problems either by using time integration schemes that handle such discontinuities ([ST96]), or accepting the error in treating all collisions during a single timestep as though they were simultaneous, such as in Bridson *et. al.* [BFA02]. The latter approach is called a *velocity filter*, as it passes over velocities, correcting motion in offending directions. It has proven popular, not just for its original purpose of self-collisions in cloth simulation, but also in a variety of collision scenarios, *e.g.*, hair simulation [SLF08]. Despite the fact that their three-pass algorithm robustly resolves intersections, we could not use it while keeping the shape in the modeling subspace, which is necessary to preserve the features that drew the user to choose a specific modeling algorithm in the first place (see Fig. 5.10).

Lastly, contacts can be viewed as hard, inviolable constraints within the system. This alleviates many problems of sequential impulses, in particular the "bouncing" back and forth between active collisions. Initial research formulated these constraints at the acceleration level, and focused on rigid bodies [Löt84, Bar89]. Baraff [Bar94] noted that these constraints do not always have a solution, and thus proposed constraining motion at the velocity level. These constraint-maintaining impulses have enjoyed popularity, with progress in custom-designed numerical methods [ST96] as well as a growing interest in friction [KEP05], a particularly challenging problem. Unfortunately, they only apply to rigid and quasi-rigid objects, and do not scale well to general highly deformable surfaces. Our approach is most similar in spirit to the constraint-based approach of Allard *et. al.* [AFC$^+$10]; we compare to this work in greater detail in Section 5.2.

Despite Anitescu and Potra [AP97] demonstrating insolvable configurations with this approach, it has remained visible in the literature, particularly for rigid and quasi-rigid bodies [PPG04, KSJP08].

For the sake of graphics-specific applications, researchers have relaxed the strict physical requirements to obtain an increase in performance. For example, Milenkovic and Schmidle [MS01b] take a more geometric, rather than physical, approach to resolving contacts between rigid bodies occasionally reaching unsolvable configurations.

There also exists a branch in collision handling literature dealing with interruptible algorithms, those that may need to be halted in the middle of processing, usually due to computational constraints. Gissler *et. al.* [GST09] distribute response across subsequent frames, tracking the intermediate penetrations. O'Sullivan and Dingliana [OD99] descends a sphere-tree hierarchy breadth-first, and then apply response as far down the depth as they reached. Similarly, Mendoza and O'Sullivan [MO06] use a coarse mesh to guide the sphere fitting and traversal. These responses can be quite coarse, and relies on a good approximation of the surface with spheres or coarse meshes, which can be expensive to obtain.

Haptics are concerned with fast interference detection and response, due to the demands of interactive object manipulation. Barbič and James [BJ08], inspired by McNeely et. al. [MPT99], use a hierarchy of point-shells to approximate objects and obtain extremely stable collision response. Their method is limited to interactions between rigid objects or a rigid and a deformable object, and requires pre-processing to obtain distance fields and well-sampled surfaces.

In summary, while the contact literature is vast, and given a particular setup, good algorithms can be found to meet its needs, there lacks any single approach that satisfies all our desiderata set forth in Section 5.

Geometric modeling. Research in interference response for geometric modeling has been quite limited. The work of von Funck *et. al.* [VFTS06] offers a modeling tool that deforms surfaces via integration of a smooth vector field. As a by-product of this smoothness property, the mesh is free of local self-intersections. A similar result is obtained from the method of Swirling Sweepers [ACWK06]. In the context of preventing local self-intersections, this behavior is a result of the deformation method under consideration, limiting its applicability to other models.

Spatial deformations affect all geometry it overlaps, preventing interference as long as it defines a bijective map. Gain and Dodgson [GD01] specifically discuss intersections within free-form deformation (FFD), and the mathematical requirement to construct a FFD scheme that does not introduce self-intersections. In particular, they prove that injectivity of the FFD mapping is sufficient to guarantee no self-intersections, and offer a modified FFD that performs such injective deformations. This work does not easily extend to other modeling techniques. To the best of our knowledge, this Chapter is the first work that specifically addresses the needs of contact response within a general modeling framework.

Snyder [Sny95] presents a method for placing objects in a scene while avoiding intersections. It only supports rigid bodies and works by applying pseudo-forces and various backtracking methods to find a realistic configuration. In contrast, we desire for a more general modeling environment not restricted to only those configurations that are realistically physical.

Aldrich *et. al.* [APH11] deforms volumes through the use of collisions. The basic response model pushes vertices outside of the interfering object. The method we present could be substituted for this step, offering far more robust collision handling in the context of their volume-preserving model.

5.2 Space-time interference volumes

Let S denote a collection of meshed surfaces in space, described by a vertex position vector $\mathbf{q} \in \mathbb{R}^{3N}$; $\mathbf{q}_i$ is the i-th 3-dimensional vertex of the mesh. For the purposes of our algorithm, we do not distinguish between different objects—they are all regarded as a single surface (possibly with disconnected components). An arbitrary point $\mathbf{r} \in S$ can be written as a weighted sum of vertices, $\mathbf{r} = \sum_i w_i \mathbf{q}_i$, where $w_i \neq 0$ for the primitive vertices which contain $\mathbf{r}$.

During editing, the user modifies a low-dimensional *control* mesh with configuration $\mathbf{p} \in \mathbb{R}^{3M}$, where $M \leq N$. From the change in the low dimensional configuration $\mathbf{p}$, we update our high-dimensional configuration $\mathbf{q} = f(\mathbf{p})$, where f may be a linear or non-linear function given explicitly or as a solution of an optimization problem. Many methods for surface representation and deformation can be easily cast in this form, including subdivision surfaces [CC78], free-form deformation [SP86], various linear surface editing schemes [BS07], PriMo surface modeling [BPGK06], and as-rigid-as-possible surface modeling [SA07]. The function f is a concatenation of possibly distinct deformation functions defined on different subsets of geometry.

We organize the deformation of $\mathbf{q}$ into a sequence of edits, $\mathbf{q}^{(0)}, \mathbf{q}^{(1)}, ..., \mathbf{q}^{(n)}$, where $\mathbf{q}^{(0)}$ is the initial mesh configuration. Interpolating the edits linearly, we define a continuous deformation of the shape parametrized by t:

$$\mathbf{q}^{(n)}(t) = \mathbf{q}^{(n-1)} + t\Delta\mathbf{q}^{(n)}, \text{ for } 0 \leq t \leq 1,$$

where $\Delta\mathbf{q}^{(n)} = \mathbf{q}^{(n)} - \mathbf{q}^{(n-1)}$, This also defines a piecewise-linear trajectory for every point on the surface $\mathbf{r}^{(n)}(t) \in S^{(n)}(t)$. We use this trajectory to detect and respond to mesh interference.

5.2.1 Defining interference

Consider the case where a pair of deforming points, $\mathbf{r}$ and $\mathbf{r}'$ coincide, $\mathbf{r}(t_I) = \mathbf{r}'(t_I)$. $0 < t_I \leq 1$ is the moment of intersection along the trajectory. We call the triplet $(\mathbf{r}, \mathbf{r}', t_I)$ an *interference event*. For a given point $\mathbf{r}$ in an interference event, we define $\mathbf{r}' = I(\mathbf{r})$ as the complementary point and $t_I(\mathbf{r})$ as the "time" parameter; the moment along the trajectory where the two points coincide. Note that by requiring $t_I > 0$ we require the surface at the start of an edit, $S(0)$, to be intersection free.

The set of all interfering points forms a subset of the surface which we call the *interference surface*, $S_I \subset S$. For a response algorithm to be considered robust, it must reliably reduce S_I to the empty set by appropriately modifying trajectories.

After an interference event, the trajectory $\mathbf{r}(t)$, along with a small area dS of the surface

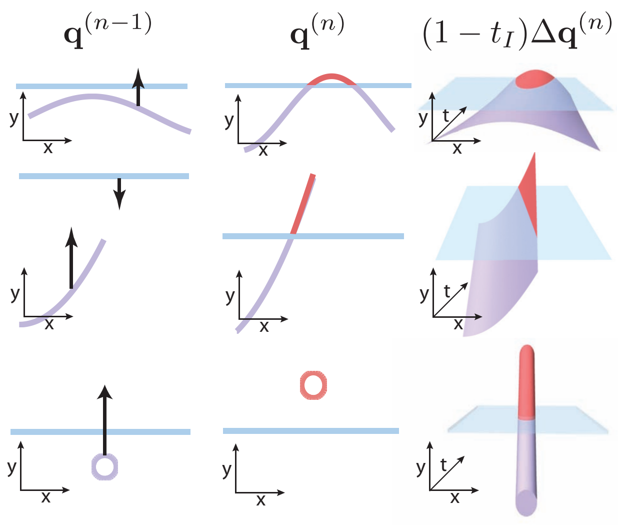


Figure 5.2: *The variety of interference captured by STIVs. Left: start configuration. Middle: configuration after edit. Right: interpolated configurations, resulting in a space-time surface. Top row shows a typical intersection between two surfaces. STIVs are always well-defined, even for boundaries (middle), and never miss interference, even when completely passing through a thin surface in a single edit (bottom).*

around $\mathbf{r}$, sweeps out a tube in space. We can use the volume of this tube to measure the severity of interference. Informally, a *space-time interference volume* (STIV) can be thought of as the sum of the volumes of these tubes for all points that passed through another surface at some instance in time.

Observe that the tubes for different areas can overlap, so STIVs do not correspond to volume swept out by the surface in space. Rather, it is a volume of a 3-dimensional subset in space-time $\mathbb{R}^3 \times \mathbb{R}$, consisting of points $(\mathbf{r}(t), t)$ for an interval of values of t (see Figure 5.2).

More formally, for the geometry deformation from configuration $\mathbf{q}^{(n-1)}$ to $\mathbf{q}^{(n)}$ we define a STIV as

$$V = \int_{\mathbf{r}\in S_I} \left[(1-t_I(\mathbf{r}))\Delta\mathbf{r}^{(n)} \cdot \hat{\mathbf{n}}(I(\mathbf{r}))\right] dS, \tag{5.1}$$

where $\hat{\mathbf{n}}$ is the normalized surface normal of the *other* point $\mathbf{r}'$ at the time of intersection, oriented so that each integrand is negative.

We have only modeled the surface interference. Before describing the process by which we eliminate this interference, we emphasize the following important features of this definition:

- The continuous integral over an arbitrary surface has no concern for the underlying object model, shape, or movement of the surface. It is completely general and can describe interference between thin objects, surfaces with sharp features, and surfaces

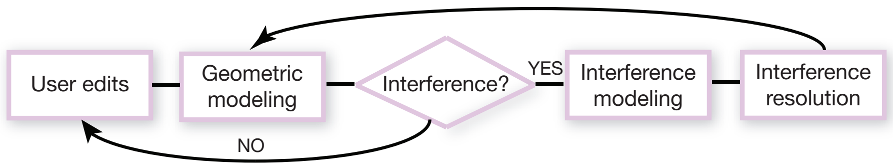


Figure 5.3: *Our workflow reacts to user edits, checking for interference and responding when necessary.*

with boundary.

- Considering continuous trajectories ensures that no intersection goes undetected, compared to sampling the geometry at fixed-interval configurations, such as in Faure *et. al.* [FBAF08], which can miss intersections between relatively fast-moving objects or geometry with thin features.
- The dot product with $\hat{\mathbf{n}}$ takes into account the angle between the linear trajectory of a point, and the normal of the surface it hits at the time of intersection: for near-sliding motions of $\mathbf{r}$, the volume is smaller, while for nearly orthogonal motion it is larger. This ensures only the motion which contributes to interference is penalized and allows surfaces to slide smoothly across one another.
- Our measure of nterference is captured by a single volume rather than a separate measure for each interfering point, drastically decreasing the problem dimension. We explore the effect of partitioning measures in Section 5.2.3.

This formulation has all the desired properties we set out in Section 5. Nevertheless, their assurance depends on the exact manner we approximate the integral and choose to resolve interference numerically.

5.2.2 Resolving interference

As discussed in Section 5.1, many choices exist for interference response in the context of physically-based simulation. However, as addressed there, none of these apply to geometric modeling without sacrificing speed, generality, controllability, or robustness.

With our STIV construction, we have two general methods for reducing the magnitude to zero. The first is penalty forces formulated through an energy term (generally of the form $\frac{1}{2}kV^2$) that penalizes STIVs, and thus interference, until resolved. Unfortunately, such a force / energy pair has no place in our purely geometric setup where forces have no meaning. Furthermore, penalty forces are well-known to be insufficient for robustly resolving collisions due to the difficulty in tuning the parameter k [Bar89, HVS$^+$09].

Our alternative is constraint-based methods. Instead of relying on a force with arbitrary stiffness k to resolve the interference over time, we can constrain V to be exactly 0 and allow the numerical method to find the exact stiffness necessary. Constraints are well-studied in optimization theory [BV04], thus making it simple to develop a purely geometric formulation. STIV constraints can be resolved by a straight-forward application of methods from numerical optimization. For completeness, we present the relevant material.

Constrained optimization. The general form of the constrained optimization problem we are solving is

$$\begin{aligned} \text{minimize} \quad & E(\mathbf{p}^{(n-1)}, \mathbf{p}^{(n)}) \\ \text{subject to} \quad & V(\mathbf{q}^{(n)}) = 0, \end{aligned} \tag{5.2}$$

where E is an energy measuring proximity to the desired trajectory. Many methods from geometric modeling use an energy to define the deformation, which could be used as E. However, even methods which do not explicitly formulate an energy (such as subdivision surfaces) can be used if we imagine the function $f(\mathbf{p})$ as a minimizer of some energy, whether or not it is explicitly given.

We opt for a different choice for two reasons. First, as the number of active constraints during any given edit varies, the Lagrange multiplier system for the problem will need to be reconstructed, making it difficult to use any pre-computation or pre-factoring for interactivity. Secondly, we want to keep our technique sufficiently general, with a minimal dependence on the choice of the deformation technique.

We regard the interference response as a post-process, where we directly apply constraints on the configuration of the mesh. The user performs an edit, followed by computation of a *candidate* configuration $\tilde{\mathbf{q}}^{(n)} = f(\mathbf{p}^{(n)})$. We then perform interference detection on the candidate trajectory $\Delta\tilde{\mathbf{q}}^{(n)}$, and apply response to $\mathbf{p}^{(n)}$ to obtain the intersection-free, final configuration $\mathbf{q}^{(n)}$. Figure 5.3 shows the entire editing workflow. Our interference resolution algorithm is summarized in the pseudocode of Algorithm 1, which gives the analytic solution to Equation 5.2 subject to a single constraint.

Algorithm 1 Resolving interference through STIV constraints

```
1: $\Delta\mathbf{q}^{(n)} = f(\mathbf{p}^{(n)}) - \mathbf{q}^{(n-1)}$
2: while $V = \text{computeSTIV}(\mathbf{q}^{(n-1)}, \Delta\mathbf{q}^{(n)}) \neq 0$ do
3:     $\mathbf{p}^{(n)} = \mathbf{p}^{(n)} + \nabla V(-V/\nabla V \nabla V^T)$
4:     $\Delta\mathbf{q}^{(n)} = f(\mathbf{p}^{(n)}) - \mathbf{q}^{(n-1)}$
5: end while
6: $\mathbf{q}^{(n)} = \mathbf{q}^{(n-1)} + \Delta\mathbf{q}^{(n)}$
```

The procedure computeSTIV is discussed in Section 5.4. Note that the function f in Line 1 is a black box operation representing the transformation from a geometric subspace $\mathbf{p}^{(n)}$

to the high-dimensional space $\mathbf{q}^{(n)}$. Response is also performed on this modeling subspace $\mathbf{p}^{(n)}$, ensuring that generated surfaces retain the desired formulation (Section 5.5.1). The gradient of the STIV, ∇V, is given in Appendix D.

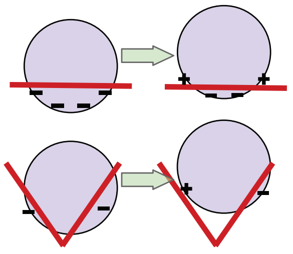
This algorithm is iterative, continuing until all intersections are resolved. Due to the linearization of a non-linear 4D volume, the STIV may not be completely removed in a single iteration. However, it is guaranteed to continuously decrease in magnitude. Additionally, by altering the trajectory, new intersections may be introduced that must be detected and resolved (imagine a car applying brakes to prevent a frontal collision and being rear-ended). By requiring that $S(0)$ be free from intersections, a solution (albeit extreme) is guaranteed by eliminating all deformation, in which case $\mathbf{q}^{(n)} = \mathbf{q}^{(n-1)}$.

Comparison to related techniques. Our formulation is related to the work of Pauly *et. al.* [PPG04], which also formulates constraints on regions of interference to resolve contact between quasi-rigid bodies. on quasi-rigid bodies on contact surfaces. However, they perform discrete interference detection, computing inside / outside tests of points on objects. This works well for the purpose of quasi-rigid volumes, but can miss intersections between surfaces or objects that move quickly or instantaneously, both common scenarios in geometric modeling.

Allard *et. al.* [AFC+10] formulated a similar constraint on intersection volumes in three dimensions, rather than in space-time, in the context of elastic object simulation. In this case, they also perform interference detection on static configurations, using their GPU-based technique of Layered Depth Images (LDI) [HTG04]. They constrain disjoint intersection volumes, computed by summing intersecting pixel areas in the LDI, to be 0. Unfortunately, LDIs only work for watertight volumes; boundaries violate the algorithm's assumptions. Furthermore, since intersection tests are static, *i.e.*, performed at fixed moments in time, the motion of objects is limited to small deformations. For closed surfaces and small motions, STIVs closely approximate intersection volumes.

5.2.3 Multiple STIVs

In Equation 5.1 the contribution of each interference point is negative, resulting in a negative integral. However, we resolve interference by following gradient directions, which may move an individual point into a state where its differential volume is positive. This is why the algorithm is iterative; a "solution" may involve some separated elements (positive integrand) and some intersecting elements (negative integrand), which average out to zero: a numerical solution that still contains interference (see inset figure).



This does not present a problem because in the next iteration we again only include active interference points, reducing the set over which we integrate. In physical simulation we would form a single constraint per intersecting point and resolve as a Linear Complementary Problem (LCP) or a linear projection [HVTG08]. Line 3 in Algorithm 1 would be replaced by a call to an LCP or a linear solver, respectively.

The motivation behind partitioning into multiple constraints is physical. In particular, intersection response involving friction often requires handling such local constraints for physically accurate solutions. We have no such requirements. In fact, the extent to which we need a "physical" solution is only as much as required for intuitive editing of surfaces. As such, we find our solution by resolving a single constraint; we have observed no adverse effects in how the algorithm feels to a user, *e.g.*, no artificial sticking or random "bumps".

One practical concern is that by integrating over a single (potentially large) region we slow down the overall algorithm by requiring additional iterations, each carrying expensive collision detection. For comparison, in Section 5.6 we include results with a single STIV as well as multiple STIVs, one per disjoint interference region of the mesh. Further refinement of STIVs into additional constraints would improve physical accuracy of the solution, but in our experience offers no practical advantage for geometric modeling. Between a single constraint and one constraint per region, we find that the difference is negligible, both in feel and in the total number of iterations; this data is presented in Table 5.1. Intuitively, this can be explained since while the total integral may penalize some intersecting subsets, it may help others by encouraging separation where otherwise exact contact would be enforced.

5.3 Geometric Deformation Algorithms

For the sake of completeness, in this section we review a selection of the most important geometric modeling algorithms that can be enriched with our algorithm to become interference-aware. A common traits of all the reviewed methods is that the deformation is a linear function. Even if it is not a strict requirement for the application of our algorithm, linear deformation algorithms require less iterations of our response algorithm to reach interference-free configurations.

5.3.1 Subdivision Surfaces

We already discussed editing with subdivision surfaces in Chapter 4. For the purpose of this Chapter it is interesting to note that modeling with subdivision surfaces allows to change the position of vertices of a small control grid to produce a more complex and smooth surface using subdivision rules. Modeling with subdivision surfaces is linear. The position of the final vertices $\mathbf{q}^{(n)}$ can be expressed as a linear combination of the control vertices:

$$\mathbf{q}^{(n)} = \mathbf{S}\mathbf{p}^{(n)}.$$

The matrix $\mathbf{S}$ is called the subdivision matrix and is computed once before the user interacts with the control mesh. Every time a control vertex is moved, $\mathbf{q}^{(n)}$ is recomputed by a single matrix-vector product. See [Far96] for additional informations.

5.3.2 Laplacian Modeling

The core idea in Laplacian modeling is to find a deformation that keeps the local details of a surface by converting the surface in a different representation. In this Section, we will present a simplified version of the Laplacian modeling described in [Sor06] that is not able to handle rotation in the deformation. For the purpose of extending the method to handle intersection, this simpler formulation suffice.

Usually the coordinates of vertices are represented as points in the euclidean space, but the same information can be represented in a different ways that better captures the local characteristics of a surface. In Laplacian modeling, the coordinates of the vertices are transformed in a differential representation using the discrete Laplacian operator applied to the Euclidean coordinates:

$$\delta_i = \nabla(\mathbf{p}_i)$$

Proper discretization of the Laplacian operator and the pseudocode for computing it robustly on triangle meshes are provided in Appendix C.

Intuitively, the differential coordinate δ_i can be seen as a displacement of the vertex p_i with respect to the barycenter of the vertices in its 1-ring. It is easy to visualize in 2D, as shown in Figure 5.4. The discrete Laplacian operator is linear, thus the conversion between Euclidean and differential coordinates is a single matrix product. The opposite conversion requires to solve a linear system and care should be taken since the matrix that represent the Laplacian is singular.

The differential coordinates can be used to define a powerful deformation scheme, by minimizing the following energy:

$$E(\mathbf{p}') = \sum_i \|\nabla \mathbf{p}'_i - \delta_i\|^2 + \sum_j \|\mathbf{p}'_j - \mathbf{p}_j\|^2$$

where the first sum runs over all mesh vertices, while the second only on the handles, that are a subset of the mesh vertices. The first term of the energy tends to produce a

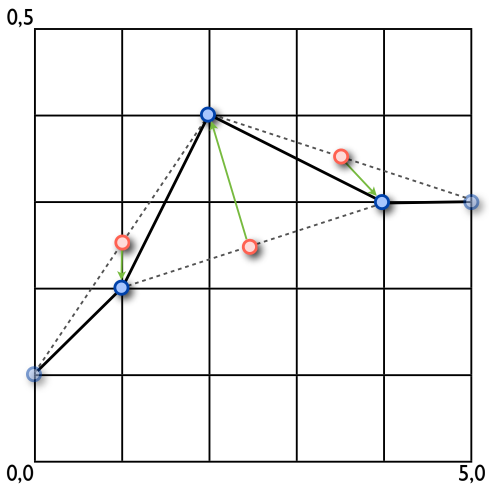


Figure 5.4: *Differential coordinates in 2D. The blue points represent the Euclidean coordinates, the red points are the barycenter of the two neighbors and the green arrow are the differential coordinates.*

new surface, whose points have coordinates $\mathbf{p}'$, such that its locally similar to the original surface. The second term instead forces some vertices to move in a different position. Minimizing this energy produces a new smooth surface that resembles the original one but with a smooth deformation applied depending on the positions of the handles. An example of deformation using Laplacian editing is shown in Figure 5.15.

5.3.3 Free-Form Deformation

Differently from the methods we discussed until now, it is possible to deform the space where the surface is embedded instead of the deforming the surface itself. Clearly, a deformation of the space will reflect in a deformation of the surface. This approach has several advantages, since the number of degrees of freedom of the deformation applied to the space is unrelated to the shape of the object we want to deform. This means that topological inconsistencies, non-manifold singularities, degenerate triangles get deformed without problems. A drawback is that the control is usually not as fine as working directly with the surface. For an example of free-form deformation see Figures 5.11, 5.12 and 5.13. Formally, a free-form deformation (FFD) is expressed as a trivariate function $d : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ that transforms $\mathbb{R}^3$, implicitly deforming any surface embedded into it. Different types of space deformation are usually used, we will discuss in this section the Lattice-Based and Cage-Based deformations.

Lattice-Based Freeform Deformation The classical FFD [SP86] represents the deformation as a trivariate tensor product spline:

$$d(x, y, z) = \sum_i \sum_j \sum_k c_{ijk} N_i(x) N_j(y) N_k(z)$$

If we consider a point $\mathbf{p}$ in $\mathbb{R}^3$ then we can rewrite its transformed position as a linear combination of the control points c and we get:

$$d(\mathbf{p}) = \sum_i c_i w^i_{\mathbf{p}}$$

where the weights w are combinations of the basis N that depend on $\mathbf{p}$. Note that since the basis have local support, most weights will be 0. The smoothness of the deformation can be controlled by using splines of different degree.

It is possible to precompute the weights $w_{\mathbf{p}}$ for all the mesh vertices we want to deform and then write a linear function that produces the coordinates of all vertices of the surface starting from the position of the control points of the trivariate spline:

$$\mathbf{q}^{(n)} = \mathbf{S}\mathbf{p}^{(n)}.$$

where the i-th row of S contains the weights $w^i_{\mathbf{p}}$, with i that varies 1 to n.

Cage-Based Freeform Deformation Cage based deformation algorithms are a generalization of lattice-based freeform deformation, where the control points are not forced to be disposed in a grid. The control points form a cage that surrounds the object, and the deformation on the cage is translated in a deformation of the contained space that implicitly deforms the contained surface. Exactly as before, the position of the deformed vertices can be expressed as a linear combinations of the control points:

$$d(\mathbf{p}) = \sum_i c_i w^i_{\mathbf{p}}$$

where the weights are generalized barycenter coordinates (see [LLCO08] for a recent proposal). Also in this case, the function that deforms the surface can be written as a matrix product, after the weights have been precomputed for all surface vertices.

5.4 Computing interference volumes

Equation 5.1 defines a STIV for a continuous surface; our algorithm works on a mesh approximating the surface, which we assume consists of triangular faces. We do not make assumptions about the number of connected components, or manifold property, but we do require that the initial meshes have no self-intersections. At the same time, our technique, in combination with a skeletonization process, can be used to eliminate pre-existing self-intersections (Section 5.5.2), although without a guarantee of success.

We follow a natural discretization of Equation 5.1 where the summation is performed over vertices. Our discrete approximation has the following form:

$$V \approx \sum_{i \in S_I^{(n)}} \left[(1-t_i)\Delta \mathbf{r}_i^{(n)} \cdot \hat{\mathbf{n}}_i(t_i)\right] \frac{1}{3} \sum_{k \in N(i)} |A_{ik}|. \tag{5.3}$$

A_{ik}, is the area of the k-th triangle connected to vertex i. Each entry in the summation can be thought of not as a tube, but a prism surrounding each vertex, whose base is the barycentric region around that vertex.

Triangle meshes can come into contact in one of two ways: either a vertex strikes a face, or two edges meet. A vertex intersecting a vertex or an edge is considered a special case of the former.

Interference prisms. $S_I^{(n)}$ is the discrete surface subset composed of points $\mathbf{q}_i$ involved in surface interference. We include in $S_I^{(n)}$ all vertices whose barycentric region A_i contains some point $\mathbf{r}$ involved in interference during a deformation. Furthermore, we use the time, trajectory, and normal corresponding to the earliest interference event in the region of $\mathbf{q}_i$ to correspond to t_i, $\mathbf{r}_i$, and $\hat{\mathbf{n}}_i$. Concretely, define the barycentric region corresponding to area A_i as the set of points

$$R_i = \{r_j = \sum w_k \mathbf{q}_k, \text{ s.t. } w_i \geq w_k, \forall\ 1 \leq k \leq N\}.$$

Note that $A_i = \int_{R_i} d\mathbf{r}$. With this notation in hand, we use the pair

$$(r_i, t_i), \text{ s.t. } t_i = \min(t_j(\mathbf{r}_j)), \forall\ \mathbf{r}_j \in R_i,$$

for the trajectory and time to represent the i-th region.

5.4.1 Detecting interference

We have yet to compute each intersecting point $\mathbf{r}$ and its time of intersection. Triangle meshes can come into contact in one of two ways: either a vertex strikes a face, or two edges meet. A vertex intersecting a vertex or an edge is considered a special case of the former. We must find all such events to construct the sets R_i.

Computing the discrete volumes, then, relies on our ability to quickly detect the points along trajectories, t_i, where intersection occurs. These tests are performed on the high-dimensional surface whose shape is defined by $\mathbf{q}_i$. Fortunately, this sort of collision detection is standard, and we can use off-the-shelf algorithms with minimal modifications for our purposes. We have tested our system with the Self-CCD library [UNC10] as well as the hash grid approach of Teschner *et. al.* [THM+03], with complete interchangeability. For low-level tests between vertex-triangle and edge-edge pairs we solve the cubic polynomials presented in Provot [Pro97]. The roots of these polynomials provide our times of intersection t_i. Along with the point trajectories $\mathbf{r}_i$, we can compute the STIV of Eqn. 5.3. Where multiple, disjoint STIVs are used, we partition the interference surface S_I based on the one-ring connectivity of interference points.

Following the usual conventions, we break the process of collision detection down into a broad-phase and a narrow-phase.

Broad-phase. The purpose of broad-phase collision detection is to quickly eliminate large batches of potentially colliding features. For this stage, we use a variation of a spatial partitioning structure called a hash grid [THM+03]. We are detecting interference of primitives in motion, so we build an axis-aligned bounding box (AABB) around the entire trajectory, $\mathbf{q}^{(n-1)}$ and $\mathbf{q}^{(n)}$, in order to conservatively detect continuous interference. This part of our algorithm can easily be treated as a black box, replaced with the fastest method available at the time, or one optimal for a particular task.

Spatial partitioning methods work by dividing space up into regions of fixed size, called *cells*. Based on their position, primitives are assigned to cells. Co-location in a cell is a necessary condition for two primitives to be intersecting. Hence, any one feature only needs to be checked for collision with the other features contained in its cell.

We use a variation of this called a hash grid. Storing grids of these cells can be memory-intensive, especially for vast scenes. Alternatively, we can assign each cell a unique value which is run through a hash function, then store a reference to this hash code. Primitives inside cells which have identical hash codes are co-located and tagged for narrow-phase interference detection. We are detecting interference of primitives in motion, so we build an axis-aligned bounding box (AABB) around the entire trajectory, $\mathbf{q}^{(n-1)}$ and $\mathbf{q}^{(n)}$, which we then store in the grid. See Teschner *et. al.* [THM+03] for full details.

Additionally, Pabst *et. al.* [PKS10] further optimize hash grids by parallelizing simple blocks and utilizing the GPU, both for traversing the hash grid and for narrow-phase processing. We borrow pieces of this process, parallelizing the AABB computation and the hash code processing.

We do take advantage of a few unique points about geometric modeling applications. For instance, only a subset of vertices in a scene are usually edited at any one time. Instead of performing detection of the entire mesh, we only check the vertices that have moved against all the rest of the mesh triangles, a conservative optimization. Furthermore, we build AABBs around disjoint meshes. Only those which overlap the AABB of moving vertices are inserted into the hash grid for the next stage of interference detection. For scenes

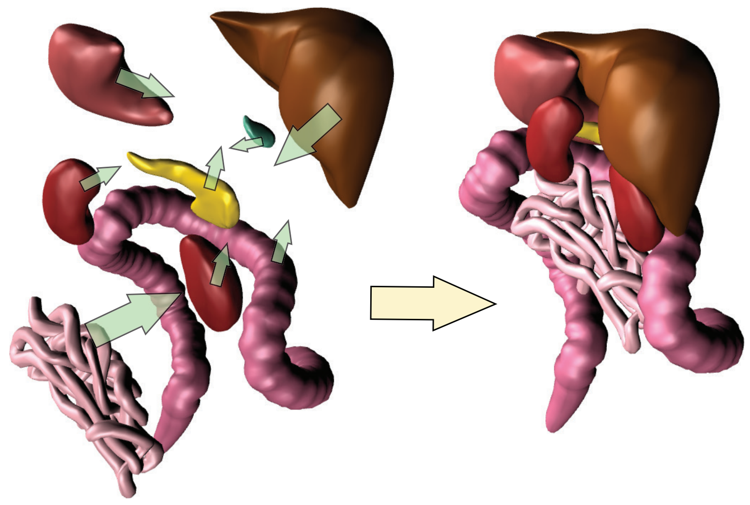


Figure 5.5: *The modeling task is to pack these bodily organs tightly together. Doing so without consideration for interference is likely to result in intersecting meshes.*

consisting of a large number of disjoint meshes, like Figure 5.5, this culls a considerable number of triangles from consideration.

Narrow-phase. The broad-phase pass returns a list of vertex-triangle candidate intersections. We perform a more expensive, but exact test on these before continuing on to interference modeling. Determining the intersection point and time (t_I) between a vertex and a triangle is a standard operation. For completeness, we reproduce the process here.

Parametrized vertex $\mathbf{x}_3(t)$ and triangle $(\mathbf{x}_0(t), \mathbf{x}_1(t), \mathbf{x}_2(t))$ are coplanar at the value(s) of t which solve the equation

$$((\mathbf{x}_3(t) - \mathbf{x}_0(t)) \cdot [(\mathbf{x}_1(t) - \mathbf{x}_0(t)) \times (\mathbf{x}_2(t) - \mathbf{x}_0(t))] = 0.$$

This is a cubic polynomial function of the parameter t, on which we can use standard root-finding methods. We use the Jenkins-Traub algorithm [JT70], well-known for its reliability. This yield three candidate intersection times; the actual intersection time t_r satisfies $\|\mathbf{x}_3(t_r) - \sum_{i=0,1,2} \alpha_i \mathbf{x}_i(t_r)\| < \epsilon$. Following Wong [Won05], we implement a series of fast tests on the polynomial's coefficients to quickly cull candidate intersections which have no root in the interval $[0, 1]$. ϵ depends on the accuracy of the root-finding algorithm. Assuming the root is accurate within δ, we use the following for ϵ:

$$\epsilon = \delta \left(\Delta\mathbf{x}_3(t_r) - \sum_{i=0,1,2} \alpha_i \Delta\mathbf{x}_i(t_r) \right) \cdot \hat{\mathbf{n}}.$$

The system attempts to exactly resolve each STIV, $V(\mathbf{q}^{(n)}) = 0$. However, due to numerical imprecision, primitives may be left touching or slightly intersecting. To remedy this, we return a value of t_I sooner than the actual value, to account for error in the root finding and response computation. In our examples we return the maximum of 0 and $(t_i - \frac{1}{100})$.

5.5 Editing

User interfaces implementing interference-aware geometric modeling require little modification from standard modeling software. We describe simple editing primitives and modes that we use in our examples. Most of them are standard, but with their power significantly enhanced by interference awareness.

Primitives. In our examples we use standard translation, rotation and scaling of control points or groups of vertices acting as handles. However, in our system, a simple rigid transform on the controls may result in a complex deformation due to interaction with other objects.

Less standard tools we found essential both for object positioning and removal of self-intersections are *contraction* towards its skeleton (Section 5.5.2), and *expansion* back to its original configuration.

Modes. Editing modes define the effect primitive operations have on geometry, and are specific to interference-aware modeling. We have three basic modes: we can enable / disable interference processing, enforce rigid response or allow the mesh to deform, and allow all surfaces to respond to interference or only the selected subset.

We utilize these primitives and modes extensively in our examples, and refer to them in describing our results in Section 5.6.

5.5.1 Modeling subspace

Meshes representing complex shapes have many degrees of freedom and editing vertices directly is often impractical. Most modeling techniques reduces this space in various ways: the geometry is defined by a smaller number of degrees of freedom in a reduced subspace (subdivision surface control points or handles for variational surface editing). Restricting possible configurations of meshes often guarantees many useful properties (*e.g.*, smoothness or detail preservation). We do not wish our treatment of interference to disturb these properties, and thus we perform response in the modeling subspace.

Note in Line 3 of Algorithm 1, response is applied by directly modifying $\mathbf{p}^{(n)}$, the handle or control of the mesh. Appendix D derives the gradients with respect to the fine degrees

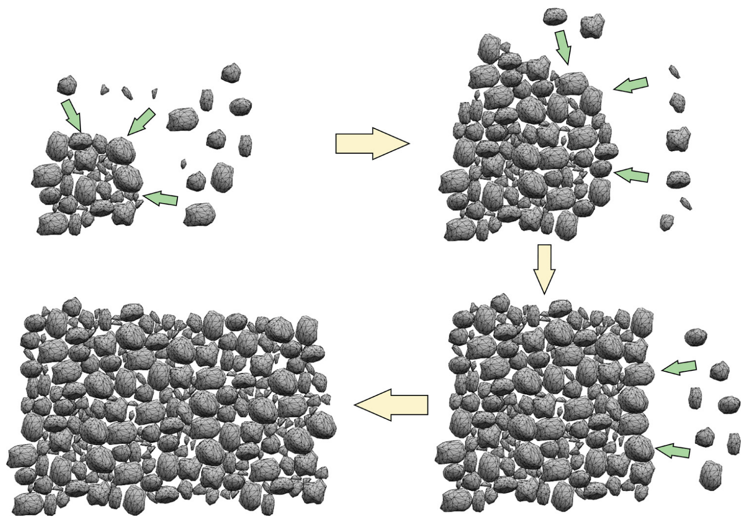


Figure 5.6: *This stone wall is constructed by moving each individual stone rigidly. As the stones interact, they slide and rotate around each other, facilitating construction of a plausible structure.*

of freedom $\mathbf{q}^{(n)}$. Using the chain rule, we can easily express these gradients as

$$\frac{\nabla V}{\partial \mathbf{p}^{(n)}} = \frac{\nabla V}{\partial \mathbf{q}^{(n)}} \frac{\partial \mathbf{q}^{(n)}}{\partial \mathbf{p}^{(n)}}.$$

As $\mathbf{q}^{(n)} = f(\mathbf{p}^{(n)})$, $\partial \mathbf{q}^{(n)} / \partial \mathbf{p}^{(n)} = \partial f / \partial \mathbf{p}^{(n)}$. The gradients for several common modeling techniques can be found in Appendix D.

5.5.2 Controlling the behavior of response

Due to the variety of needs in different modeling systems, it is impossible to present a single solution that is "one size fits all." However, we present a variety of options and "add-ons" that increase the utility of interference-aware geometric modeling. Some are purely geometric, other are motivated by physical intuition, but are always formulated in purely geometric terms.

Weighted handles. The gradients ∇V describe how to modify the DOFs in $\mathbf{p}^{(n)}$ to avoid intersection. By appropriately weighing ∇V, we can capture various inertial effects.

Let $\mathbf{W}$ be the $3M \times 3M$ identity matrix, and substitute $\mathbf{W}\nabla V^T$ for ∇V^T in Line 3. By modifying the (i,i)-th entry of $\mathbf{W}$, we can control the relative effect of response on the i-th DOF. This is most useful by inserting a 0, which removes the DOF from the system, ensuring that the corresponding vertex remains stationary. This allows us to

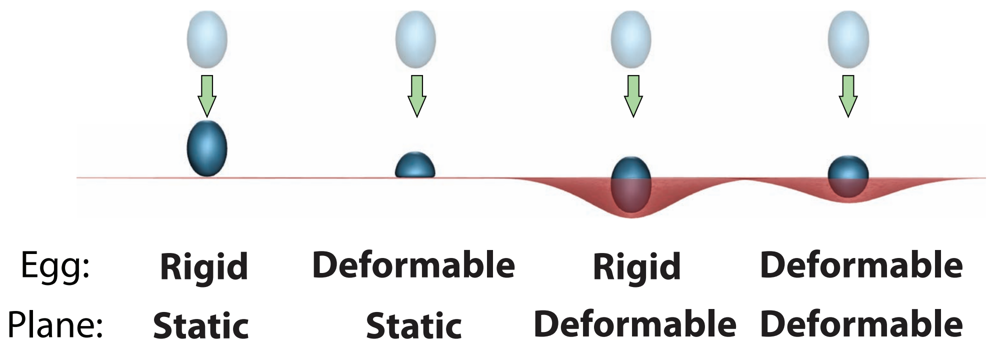


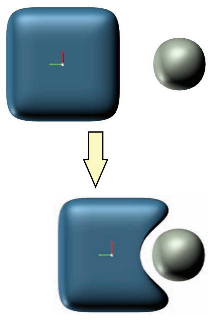
Figure 5.7: *We weigh the response per handle vertex to achieve many different effects, each useful under different circumstances.*

restrict deformation due to response to only those vertices selected, only those not selected, or allow all to deform.

Minimum separation. In Equation 5.3, t_i is the parametric value at which an intersection occurs. We can modify the computation of t_i to return the time when two surfaces enter within some *proximity*, rather than exactly touching. This is useful for simulation pipelines, where a minimum distance between all surfaces is needed.

We can express this with the following degree six polynomial:

$$(\mathbf{x}_3(t) - \mathbf{x}_0(t)) \cdot [(\mathbf{x}_1(t) - \mathbf{x}_0(t)) \times (\mathbf{x}_2(t) - \mathbf{x}_0(t))]^2 - \\ h^2 \| [(\mathbf{x}_1(t) - \mathbf{x}_0(t)) \times (\mathbf{x}_2(t) - \mathbf{x}_0(t))] \|^2.$$



Finding the roots of this polynomial gives the points along the trajectory where the vertex is exactly a distance h apart from the plane spanned by the triangle. By projecting onto the plane, we can confirm that it lies within the region of the triangle. A similar polynomial is constructed for edge-edge intersections.

We employ a series of optimizations, not unlike those in the cubic case, that vastly reduce the number of polynomials that must actually be solved. Nevertheless, the increased proximity increases the amount of primitives that make it to low-level intersection tests. We are limited, however, by the minimum separation distance supported. If set too high, two-ring neighbors will be flagged as intersecting. Avoiding this requires checking for these special cases.

Shape preservation. By default, deformations due to collisions in our algorithm are *plastic*, that is, the surface does not attempt to reform its original shape after the interfer-

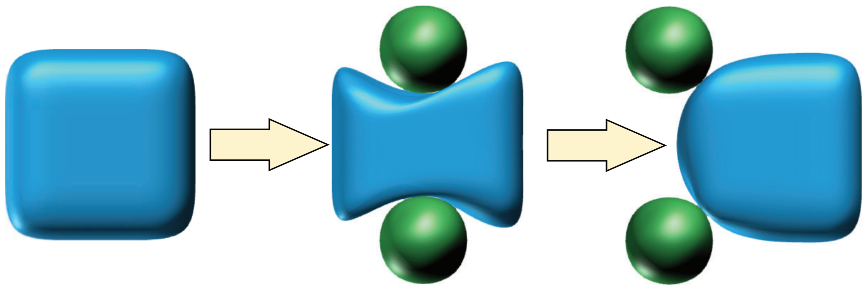


Figure 5.8: *Implementing shape preservation in our algorithm allows surfaces deformed by collisions to regain their original configuration.*

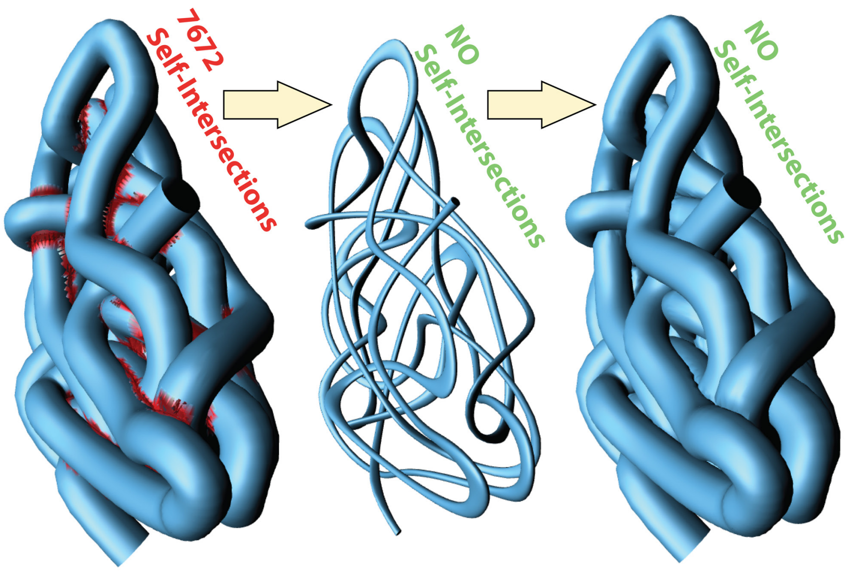


Figure 5.9: *With a simple extension, our algorithm is able to untangle complicated intersections within meshes.*

ence has desisted.

By contrast, we implemented a form of *shape preservation* in our algorithm, which allows surfaces deformed by collisions to regain their original configuration (Figure 5.8). To do this, we store an object's undeformed configuration. During each iteration, we to restore the object to this shape, modulo a rigid transformation. This is a naive approach intended solely to demonstrate the flexibility of our algorithm.

Self-intersecting meshes. Utilizing Algorithm 1, we are able to resolve self-intersections that *already* exist in many meshes. Our formulation requires $\mathbf{q}^{(n-1)}$ for detection, and is thus *history dependent*. In particular, our algorithm requires a previous configuration that is intersection-free. This is fine for modeling from scratch or simple initial shapes, but

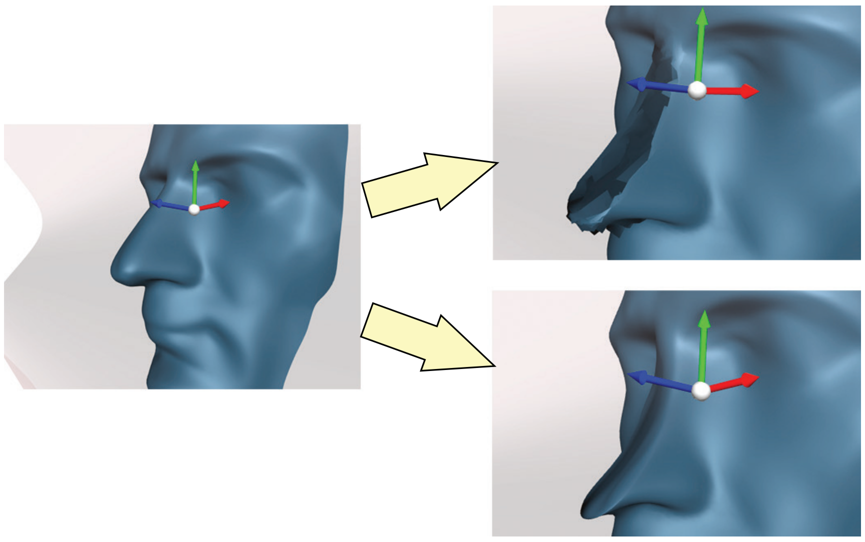


Figure 5.10: *The most viable response candidate from physical simulation, Bridson* et. al. *[BFA02] (top), has no consideration for the underlying geometric model, so while it robustly resolves the interference, in the process it ruins the continuity of this subdivision face colliding with a curvy plane. In contrast, our response (bottom) modifies the control mesh, preserving the smoothness of the subdivided surface.*

poses a challenge for meshes with pre-existing intersections.

Our algorithm for removing existing intersections is based on the observation that if we have *any* non-self intersecting shape $\mathbf{q}^0$, as long as there is one-to-one correspondence to the final shape, we can run our algorithm between $\mathbf{q}^{(0)}$ and a desired configuration $\tilde{\mathbf{q}}^{(1)}$, with intersections repaired automatically.

While the task of constructing such shape in full generality is formidable, a natural candidate in many cases is a shape closer to the skeleton of the mesh. Intermediate steps of the method of Au *et. al.* [ATC+08], based on constrained Laplacian smoothing, yield exactly such meshes. Using this method, we contract the self-intersecting mesh until it is free of intersections. When none are detected, we have found our intersection-free configuration $\mathbf{q}^{(0)}$. We then run an unmodified Algorithm 1 to obtain $\mathbf{q}^{(1)}$, the mesh closest to the original but free of intersections. Clearly, this method is not guaranteed to succeed, but for many models we have tried (*e.g.*, Figure 5.9), it resolves self-intersections successfully.

5.6 Results

We tested our algorithm on a variety of scenarios that stress different parts of the system. The results demonstrate the robustness, speed, generality, and controllability of our

Model	Vertices	Triangles	Collisions	Regions	Iterations	Total (ms, per iter.)	Iterations (single)	Total (single) (ms, per iter.)
Plant	13759	26782	36.2054	4.07143	1.55556	48.5476	2.19753	40.6433
Bunny	38045	75943	46.3333	1.41667	1.07692	630.2633	1.75000	523.9560
Tree	32937	18745	16.1604	2.49057	2.20833	56.5912	2.64286	56.8141
Knot	5808	11616	23.2549	1.83333	2.29213	65.4325	2.16129	57.3880
Ogre	13318	26060	50.1433	2.60000	8.58974	36.8397	11.8438	27.5214

Table 5.1: *For five examples we give, from left to right, the number of mesh vertices, the number of mesh triangles, the average number of collisions between primitives, the average number of disjoint regions, the average number of iterations with one constraint per region, the average time per iteration (ms) with one constraint per region, the average number of iterations when using a single STIV, and the average time per iteration when using a single STIV constraint. Contributions to the average are only taken when the number of collisions is non-zero.*

method, as seen here and in the accompanying video. All modeling sessions were performed single-threaded on a 3.6GHz Intel Core i5.

The following examples range from a few thousand triangles to over 75K, with vertex counts ranging from a few hundred to 38K. Table 5.1 contains full example data, including various measures for evaluating performance, both for the case of a single STIV constraint and one per disjoint region of the interference surface. The quantitative measures typically used to evaluate performance of contact algorithms for physically-based simulation are not entirely appropriate for our task (interactive modeling). For this task, the frame rate profiles (see Figure 5.16) capture a more practically relevant performance measure. During all operations, we maintain an interactive frame rate on average, with occasional momentary dips only during the most stressful of operations.

Plant in vase. In this example we wish to place a tight bunch of grass-like shoots into a vase (Figure 5.11). The vase is immovable, while the shoots are allowed to deform through a simple tri-cubic FFD lattice. We translate the entire plant down into the vase. As it first intersects, the bottom of the lattice pinches inward, allowing the shoots to slide freely into the vase. Once we have reached the end of the vase, we scale the lattice to increase its size, allowing the bunch to freely expand outwards, while the bottom remains constricted inside the vase.

Intestines (85K faces). Frequently, modeling sessions begin with previously modeled geometry that unfortunately may contain intersections. This is the case with the small intestines of Figure 5.9. Because of the tight coiling of the long small intestines, intersections are far too numerous to repair by hand ($7,672$ intersections between pairs of triangles). Instead, we contract the mesh towards its automatically computed centerline

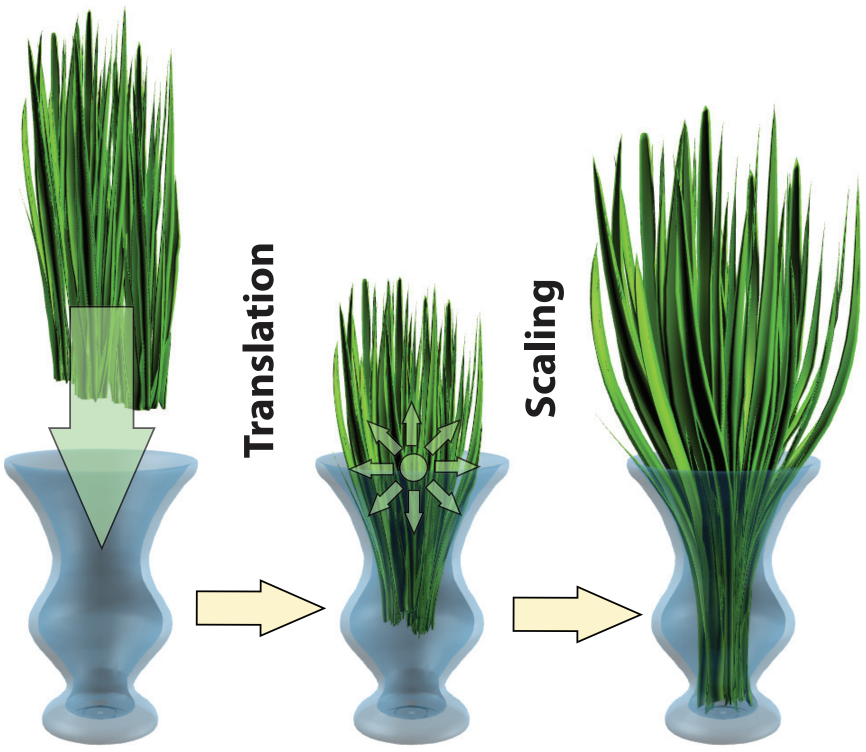


Figure 5.11: *STIVs do not introduce any artificial friction, so these plant shoots freely slide into the vase, without intersecting it.*

skeleton until it is intersection-free. Then, we expand the mesh to its original configuration, with interference-awareness enabled. This results in an intersection-free mesh as close as possible to the original input mesh.

Stone wall (18K faces). Rigid bodies are common in simulations, yet their placement into initial configurations is a modeling problem. We stack a large number of stone models into a wall (Figure 5.6), moving them rigidly to preserve their shape. This allows us to freely stack and manipulate the stones into the wall structure.

Bunny in teapot. This stress-test was designed to push our system to its limits. The teapot is fixed, while the high-resolution bunny is enclosed in a tri-cubic FFD lattice. We select the entire bunny and scale up, continuing even after intersections occur. Eventually the bunny is tightly pressed against the teapot interior (Figure 5.12). The entire meshes, both bunny and teapot, are in contact throughout. This stresses timings because there are fewer false positive intersections, and thus fewer options for culling candidate intersections; all low-levels tests must be done. Despite these set-backs, we maintain steady frame-rates and the user receives consistent feedback, even when the frame-rate eventually drops to

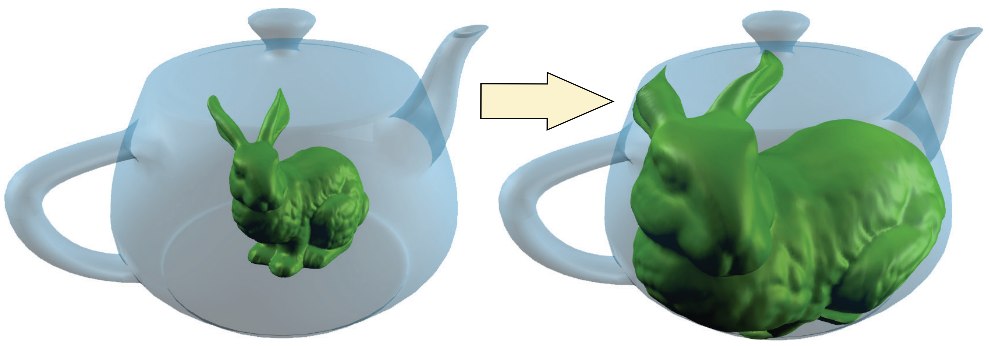


Figure 5.12: *We enlarge the bunny trapped in the teapot until it is tightly pressed against the sides. Such large regions of consistent intersection are challenging for collision detection.*

below 5 frames per second.

Tree in corner. Similar to the bunny, this example grows a tree mesh in a contained region. It also uses a tri-cubic FFD lattice, however the mesh almost entirely consists of "triangle soup". As the tree grows, it pushes against the wall, eventually creeping over the edges.

Face (12K faces). We take a subdivision surface of a face and press against a wavy plane, with only the face deforming, then only the plane deforming. The opposite mesh acts as a stamp tool, permanently branding its shape onto the intersecting mesh.

Knot. This tangled knot is deformed using Laplacian surface editing, with the bottom region fixed, and the top arch manipulated as a handle. It requires little movement to instigate intersections and tighten the knot configuration. By allowing the non-handle mesh to move, the knot becomes further entangled.

Ogre. This examples combines many operations and editing modes into a single, practical example (Figure 5.1). The task is to dress the ogre using off-the-shelf models. The shirt, for example, is a woman's shirt that clearly will not fit easily, and intersects the ogre in many places. All meshes are Loop subdivision surfaces.

We begin by eliminating intersections with the shirt, by contracting the ogre until intersection-free, then expanding her with interference enabled. This gives an intersection-free shirt, but it clearly was not made for this model. We finish the shirt by scaling the entire mesh down until it intersects the ogre, this "shrink-wrapping" removes the feminine cut of the cloth and gives a more ogre-ish shape. We move the glasses rigidly against the ogre's head

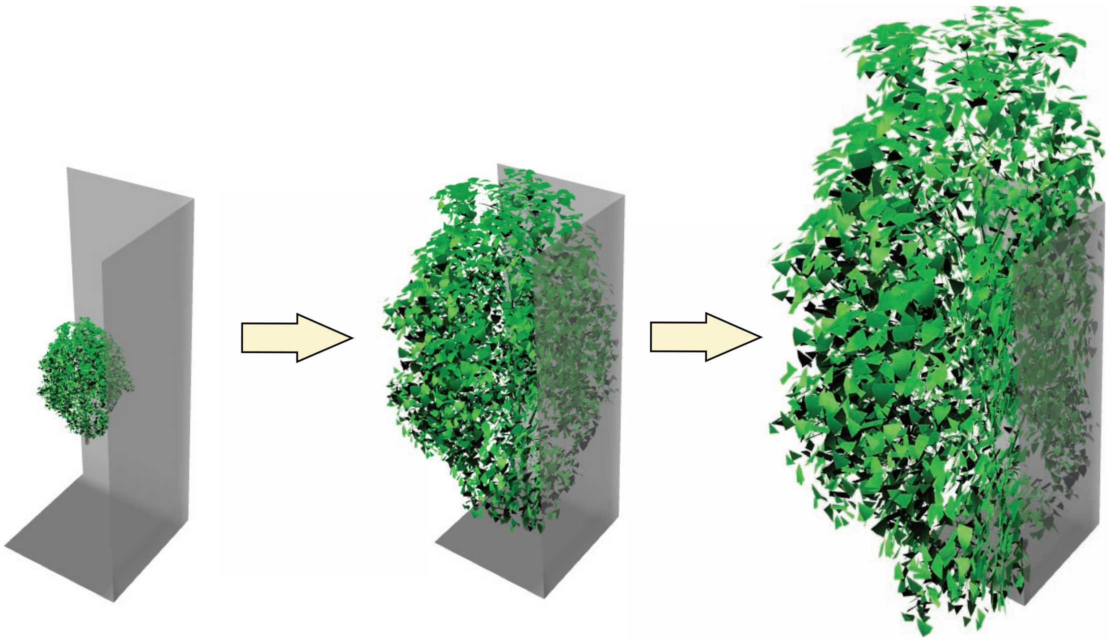


Figure 5.13: *This tree "grows" in the corner, firmly pressed against the walls without penetration.*

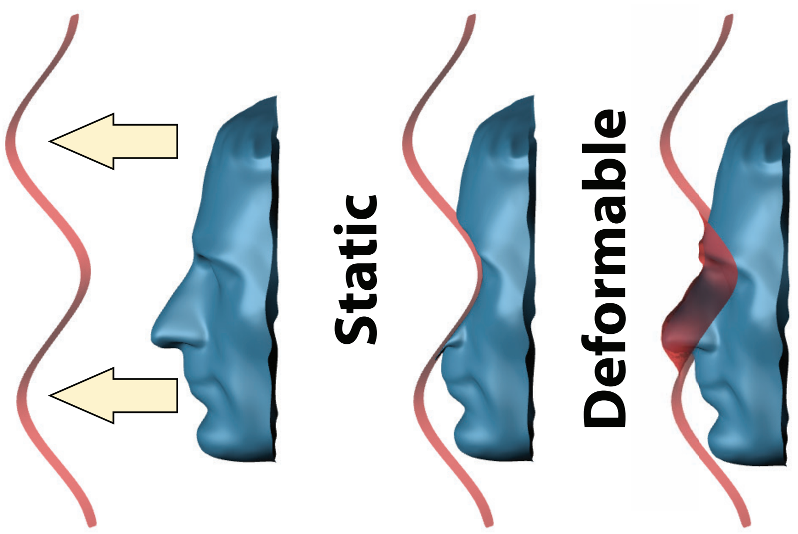


Figure 5.14: *Intersecting objects can take the shape of the mesh they interact with, inspiring interesting uses for the algorithm.*

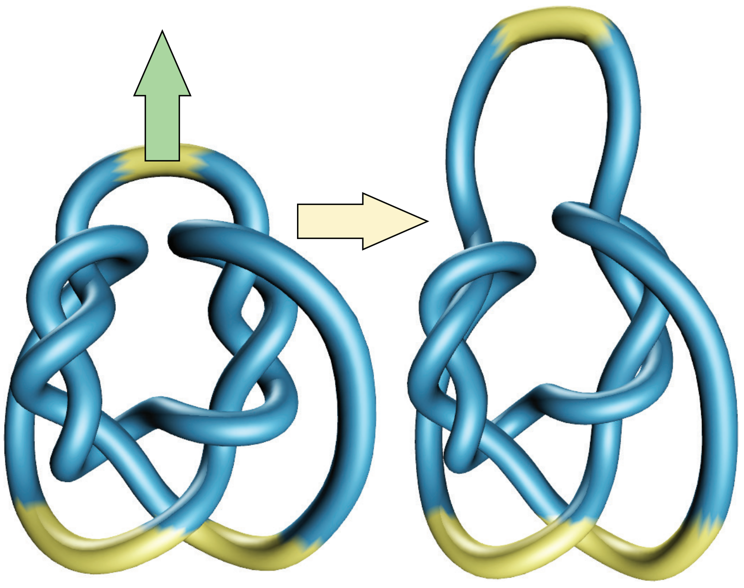


Figure 5.15: *This tangled knot has no hope of becoming untangled with interference-aware geometric modeling preventing intersections, even for large deformations.*

until they are properly positioned. We then enter deformable mode and press the glasses on its head, allowing it to freely expand to the proper size. We repeat these operations with the hat to conclude the dressing of the ogre. This is a non-trivial editing task that is made painless, even for non-experts, by interference-aware geometric modeling.

Timings. We give the frames per second values over time for the ogre shirt-fitting and the plant in vase examples. We experience dips in frame-rate during intersection, with more severe intersections causing more significant dips. The important thing to note is these dips are momentary, and quickly recover to a steady state. Overall time is spent in a few operations. An average of 15% of runtime is spent processing UI events, 15% in geometric modeling, 51% is spent in interference detection, and 16% in response. Integrating STIVs and computing gradients takes a negligible amount of time. Table 5.1 gives more detailed timing, with total costs per iteration of our algorithm.

5.7 Concluding remarks

We presented a method for responding to interference in geometric modeling sessions. This method is fast, enabling interactive editing sessions, it is general, not limited by geometry or surface representation, and it is controllable, equipping the user with a wide array of expressive ability. Interference-aware geometric modeling can fit into practically

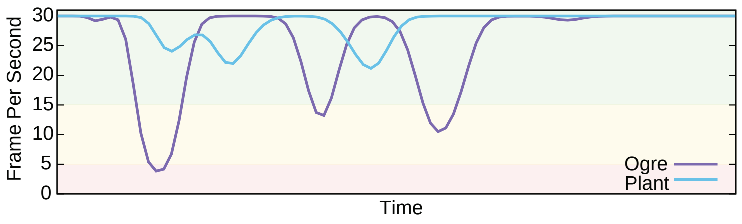


Figure 5.16: *The frames per second for two examples during an editing session. Dips are momentary, and are little interruption to our algorithm's responsive feedback.*

any existing modeling system. It can easily be enabled and disabled, allowing artists to guarantee absence of intersections during critical editing moments, while editing freely during other times.

Implementing specialized methods to handle every case is tedious and increases chance of error. Our aim was to build an algorithm that was general enough, yet had the performance capabilities to work with any scenario encountered in the domain of geometric modeling. Our resulting algorithm follows from physical principles, enough to guide intuitive behavior, but is relaxed enough to edit in real-time.

Limitations. Unfortunately, our method still has strict requirements on the input geometry. In particular, it does not handle degeneracies well. Zero area triangles and zero length edges disrupt the interference detection. Along these same lines, our algorithm also requires input meshes be free of intersection. Using our skeleton-based approach we are able to repair self-intersections in many meshes, but a few are impossible to repair in this way.

Future work. Currently, interference detection is a major bottleneck during processing. This is an active research area, and continued development in continuous detection methods will directly improve our results. There is, in particular, opportunity for utilizing parallel processing to improve interference-aware geometric modeling.

In addition, specialized interference detection algorithms show promise for geometric modeling. There are specialized factors that could be taken advantage of in designing new detection algorithms. For example, only subsets of meshes are usually edited at one time. Contrast this with simulation, for which many detection algorithms are specifically developed, which moves everything per timestep. Furthermore, algorithms could leverage specific knowledge of a modeling paradigm, *e.g.*, subdivision surfaces, for faster processing.

While debugging we observed that the response was fairly insensitive to errors in the STIV computation. This motivates inquiry in fast algorithms for approximating STIVs without performing expensive low-level interference detection.

Chapter 6

Concluding remarks

We have presented novel contributions at the various stages of the geometry processing pipeline, which all aim at providing structured and more manageable representations of 3D shapes, starting from unstructured meshes. Our objective is the generation of coarse control meshes with a quality comparable to those manually produced by artists. A high quality control mesh has a small number of quadrilateral patches that respect eventual symmetries of the object, its edges are aligned with the object features and it contains a small number of singularities.

In Section 2.1, we focused on generating coarse meshes. The decimation algorithm is based on simple and efficient local operations. Since every operation only affects a small area of the surface, it is impossible to optimize global requirements like cones placement or features alignment. Section 3.2 has shown that it is possible to generate coarse meshes with good alignment to features, but the graph simplification algorithm is unable to control the generated singularities. In this case, the optimization of the separatrices graph is purely topological and it uses global operations to simplify the graph and the corresponding cross-field. The singularities are moved to improve the quality of the quadrangulation, but it is not possible to change their number. In Section 3.3, we generated symmetric meshes, with good alignment and singularities placement. We proposed two novel symmetry detection algorithms and used them to produce symmetric cross-fields, quad meshes and non-photorealistic renderings. The quality of the results is superior to other automatic or semi-automatic state-of-the-art methods and the process is also more robust, since our formulation add coherent constraints to the optimization problem, making it simpler and easier to solve numerically. The minimal size of a quad is directly related with the layout of the singularities, and it is thus impossible to generate a coarse mesh for a model with a complex geometry.

We are currently focusing our research on finding a way to combine all these ideas in an algorithm that is able to robustly produce high quality structured meshes that are suitable for the more demanding practical applications. The quality of the results heavily depends on the choice of the number and position of singularities and the current automatic methods do not always produce satisfactory results. User input might be required to tackle the problem in a robust way; for instance, in human faces, the quad edges must be aligned with the underlying muscles of the character to increase the quality of animation. A description of the muscle structure is not present in a 3D mesh and it is extremely hard to estimate automatically. An interesting direction that we plan to explore is the automatic extraction of this information from an animation defined on the surface itself, generated using motion-capture techniques. The analysis of motion will be the basis to a new family of quadrangulation algorithms aimed at the generation of quad meshes specifically optimized for animation purposes. Another possible approach is to incorporate minimal user interaction, in the form of two-dimensional sketches, to support the automatic process. While this feature is already present in commercial tools, it requires a lot of user interaction and it forces the user to manually paint quads on complex parts of the surface.

The symmetrization algorithm we used for producing symmetric parametrizations can also be used for a variety of other applications. In particular, we plan to investigate its use on the curvature tensor field, where we hope to be able to provide a robust curvature estimation algorithm that takes symmetry into account.

At the later stages of the modeling pipeline, we consider the problem of adaptively subdividing a coarse mesh computed with the previously presented algorithms. Our implicit subdivision scheme provides a practical and theoretically sound way of subdividing meshes adaptively. The subdivision process can be locally reverted, without the need to store any explicit hierarchical data structure, since the hierarchy is implicitly encoded. Our proposal extends the well-known half-edge data structure, adding a negligible space overhead. The traversal of the hierarchy is also possible using topological angles, that allow an efficient navigation on all levels. Applications of this new subdivision scheme include adaptive subdivision surfaces and adaptive remeshing.

Finally, we extend interactive modeling algorithms to avoid intersections. With our algorithm enabled, any modeling task is guaranteed to produce intersection-free meshes that can be used without any further processing for simulations purposes. Applications like cloth modeling greatly benefits from our contribution, since physical simulations are hard or impossible to perform on self-intersecting pieces of cloth. With our algorithm, it is possible to directly model the cloth on the character, saving processing and artist's time, since the produced model will always be free from intersection and thus directly suitable for further processing. Our formulation of space-time volumes is robust and greatly reduces the number of variables in the expensive response computation, thus allowing interactive response with meshes with over 75k faces.

By applying parametrization techniques in a simplified 2D setting, our contribution on image resizing enables the design of real-time image and video retargeting algorithms, capable of producing HD quality images in a fraction of the time required by previous methods. Our contribution is the observation that the space of axis-aligned deformation is powerful enough to express most of the deformation required in content-aware media retargeting. As we have shown, this enables efficient and fast retargeting of images with a clean and simple mathematical formulation that can be easily extended for other energies. We are especially interested in taking into account the temporal coherence in our optimization process, that would enable us to retarget videos. Specifically, the combination of efficiency of our approach and an easy to use sketch-based user interface to draw the importance map could lead to a commercial solution for retargeting web videos, mainly aimed to mainstream websites like YouTube.

6.1 Future work

All algorithms presented in this thesis are fully automatic or require minimal user input. Research in geometry processing is usually focused on fully automatic algorithms and user input is used only if strictly required. Interestingly, in some practical applications, the automatically generated result requires heavy manual tuning to satisfy requirements that are hard or impossible to enforce automatically.

A classical example is the generation of a texture atlas for texture mapping, where the automatic result is only a starting point that is then edited manually to generate a layout where every patch has a semantic meaning. The layout is usually optimized to make the painting in parameter space easy; this is clearly very hard to optimize automatically. Other good examples are the generation of skeletons for deformation, and the definition of skinning weights; while automatic methods exists [BP07] [JBPS11] [JS11], these tasks are usually executed manually to achieve higher quality results. It would be interesting to consider a different class of algorithms, that rely on moderate user-input to generate directly results that do not require any additional manual tweaking. The requirements are now different, since these algorithms must run at interactive rates and provide immediate feedback. In this section, we outline possible venues for future research in this direction, with applications in parametrization, cross-parametrization, symmetry detection, volumetric remeshing, surface reconstruction and form finding.

6.1.1 Interactive quad mesh painting

The current state of art in the generation of coarse quadrilateral meshes in the movie industry is represented by the commercial tool 3D Coat [3dc] and the Blender's plugin B-surfaces [bsu]. Both software package provide tools to paint points on surfaces and connect them with edges to form a quad mesh. Surprisingly, the automatic algorithms presented in the literature are not considered for these applications since they do not provide enough direct control on the placement of singularities and on the shape of quads. An interesting venue for future work is the design of an intermediate solution; it should not require to manually place every single vertex of the quad mesh and also not be completely automatic. Since the crucial parts of the generation of an high quality quad mesh is the placement of singularities, it is reasonable to assume that the user will provide them manually, and that the system will provide an initial guess of the topology of a cross-field with those singularities and use it to lay out a quad mesh. Successive user sketches can then affect both the cross-fields and the quad density, allowing complete control on the final mesh.

6.1.2 Interactive cross-parametrization

Another interesting problem that has never been studied in this interactive setting, is the generation of a dense correspondence between surfaces. The problem can be stated as the generation of a dense, possibly bijective, map between two different surfaces with the same genus. This problem is usually called cross-parametrization and various fully automatic methods have been proposed [SAPH04] [KS04]. As with the generation of quad meshes, the automatic results are not controllable enough and these algorithms are never being used in practical applications. In this case, speed is also an issue; the computation is expensive and the user must waits for tens of seconds every time a parameter is tweaked to see its effect. Recently, a partial solution to this problem has been proposed in [Rus10]. The author proposes an algorithm to discretize the Fréchet mean on piecewise-linear surfaces. Given a small subset of the vertices of a surface called anchors, [Rus10] is able to compute a set of weights for every point of the surface such that average of the anchors with these weights is the point itself. This algorithm, in combination with its inverse (given anchors and weights, compute the point that is the weighted average of the anchors), allows to define dense correspondences between two surfaces given corresponding anchors. To the best of our knowledge, the inverse problem has never been studied before. The timings reported in [Rus10] are in the order of milliseconds, and it is thus realistic to expect this approach to compute dense correspondences in real-time as the user moves the anchors. Possible applications of such an algorithm would be texture transfer, skinning weights transfer, local parametrization and splines on surfaces. The major challenge is the definition of a smooth and efficient inverse of [Rus10] or to provide an entirely different discretization of the Fréchet mean that has a simple inverse.

6.1.3 Interactive volumetric parametrization and hex-meshing

Hexaedral remeshing, i.e. filling a volume with cubes of approximately the same size, is a hard problem with various practical applications. Hex meshes are preferred with respect to tetrahedral meshes as a way of discretizing volumes for FEM simulations since they enable simpler and more stable computations. Hex meshing a volume induces a quad mesh on its boundary; quad meshing can thus be considered as a subproblem of the more general hexahedral meshing. Recently, the cross-field based field generation used in Section 3.3, has been extended to the volumetric case and used to generate hex meshes in [NRP11]. No automatic algorithm is known to automatically compute a volumetric cross-fields, and the problem is much harder that constructing a cross-field on surfaces, since the singularities are not just vertices but also 1D simplexes. The authors of [NRP11] generate the cross-field manually and then use their algorithm to fill the volume with cubes. Another semi-automatic algorithm [MCK08] computes a parametrization with a fixed topology starting from minimal manual input. The design of a semi-automatic algorithm that generates the topology of an arbitrary 3D cross-fields is a challenging and interesting problem that has not been studied before. Since it is really hard to draw inside a volume, the user input should be restricted to sketches on the boundary on the volume, or eventually sketches on a small subset of 2D slices.

6.1.4 Interactive surface reconstruction

Surface reconstruction is a well-known problem with a plethora of efficient algorithms that are able to deal even with noisy or missing data. They all focus on the generation of triangular meshes, and at the best of our knowledge the only proposal that is able to automatically generate structured quad-meshes is [PTSZ11].

A novel interesting problem is the copy of the topology of an existing mesh to the geometry of a point cloud, without knowing dense correspondences between the two. A cross-parametrization approach could be attempted but it will be really challenging to robustly handle the noise that is present in point-clouds acquired with 3D scanners.

Another possible approach is to press the template mesh on the point cloud, while keeping enabled a variant of the algorithm proposed in Chapter 5. The user can have complete control of the process and can tweak the deformation in real-time. While this paradigm is not suitable for arbitrary meshes, it will be really useful for special cases like face reconstruction.

Adapting interference-aware modeling to work with point clouds is not trivial, since a new definition of STIV must be carefully designed and the system should be able to properly handle point clouds with non-uniform density. Another challenging task is to increase the efficiency of the entire system to handle in real-time point clouds with millions of vertices, that are commonly acquired by recent 3D scanners.

6.1.5 Interactive form finding for self-supporting surfaces

Self-supporting surfaces represent shell structures that, in case of physical realization, are able to stand without any support in the interior. All the internal forces are compression-only. Usual examples are masonry vaults. The automatic design of these surfaces is a challenging task that has only recently been addressed with computational approaches ([Blo09],[VHWP12]). The TNA framework [Blo09] decouples horizontal and vertical equilibrium, enabling to compute self-supporting surfaces efficiently once the horizontal force distribution is known. Designing an algorithm that allows to have full control on the forces and on the 3D shape of the surface is a highly challenging task entirely unexplored. Possible avenues for future research are the use of quad tessellation algorithm to generate suitable force diagrams and the use of customized solver for the non-linear optimization usually involved in the surface computation.

6.1.6 Hyperbolic tessellation for symmetry detection in high genus surfaces

The algorithm presented in Section 3.3 relies on correspondences provided by the user to extract a dense symmetry map of a genus 0 surface. The basic idea of the algorithm is to conformally map the surface to the complex plane and then look for involutions there where they have a simple closed form. The extension to higher genus is not trivial. However, higher genus surfaces can be conformally mapped to the hyperbolic plane. By defining appropriate conformal reflections in hyperbolic space, the entire Poincaré disk can be tessellated with copies of the surface [Thu97]. Involutions still have a simple form in this setting and it might be possible to compute a dense symmetry map extending the algorithm of Section 3.3.

Another interesting avenue for research is the use of mesh uniformization to define cross-parametrizations. While this has been already studied for genus 0 surfaces [LF09], it cannot be trivially extended to higher genus meshes.

Appendix A

Tri-to-Quad mesh conversion

Many datasets, e.g., those from range scanning, come originally as triangle meshes. Such datasets must be converted to quad meshes prior to applying our simplification method. Note that any other polygonal meshes can be trivially reduced to triangle meshes first. Solving this task will also covers the case of meshes featuring any other polygon, as it is trivial to first break them up into triangles.

Some authors obtain the initial quad mesh by performing one step of Catmull-Clark subdivision [DSSC08, DSC09a]. The obtained quad mesh has the desirable property of preserving all original edges, but it has also several drawbacks: in terms of complexity, is has three times more quads than triangles of the initial mesh; in terms of quality, no more than 50% of its vertices are regular.

In [VZ01], a hybrid tri-quad mesh is built first, similarly to what we do in Step 1 below, which is converted next into a purely quad mesh. Each face of the tri-quad mesh is split into triangles by barycentric subdivision; next, pairs of such triangles are merged to form quads by deleting all edges that existed prior to subdivision. This is better than doing a Catmull-Clark on the input triangle mesh, both in terms of quality and in terms of complexity, but it still produces a non-negligible increase in the total number of quads.

Since our simplification method is able to enforce feature lines even if they are not present as edges in the input mesh, we find it more convenient to develop another method, which does not preserve original edges, but produces a smaller number of faces and a mesh of better quality.

No complexity increase scheme

Our scheme always produces a quad quad mesh featuring half as many quads as triangles in the starting mesh. It requires, as input, a (connected) mesh with an even number of triangles. If the mesh is closed (and two-manifold), this condition is guaranteed; otherwise, any border edge can be split in half, increasing the number of triangles by one unit.

Step 1: making a quad-dominant mesh. First, most triangles are merged pairwise into quads, dissolving their shared edge. Edges of the triangle mesh can be flagged as dissolved with any heuristics, with the constraint that no triangle is allowed to have more than one flagged edge. The objectives are to maximize the number of flagged edges and to prioritize creation of quads with (nearly) right angles. We adopted a simple, linear time approach. First, for each triangle we select its best candidate edge to be dissolved, scoring each edge by the "squareness" of the corresponding quad, measured as the sum of pairwise dot products of the four normalized edges. Next, any selected edge is flagged only if this does not invalidate the selection of edges with a better score.

Step 2: making a pure quad-mesh. A few triangles (still an even number) remain after Step 1. These triangles are made to "crawl" over the mesh toward each other until they can be merged into quads. Iteratively, a triangle t_i is selected and quads in a region around it are marked, with a breadth first visit, with cross-edge distance from t_i until another triangle t_j is reached. The triangle t_j is moved toward t_i by means of a sequence of edge flip operations: specifically an edge between t_j and the neighboring quad on the path to t_i is dissolved, forming a pentagonal face, which is split back into a quad and a triangle t'_j (there are four other possible ways to do this), thus making the triangle "crawl" over the surface. The split that moves t_j faster toward t_i is selected, breaking ties in favor of the alternative maximizing squaredness.

Comparison to other conversion strategies

In terms of complexity of the resulting quad mesh, this method improves by a factor of 6 over direct use of Catmull-Clark subdivision. This is a significant gain, considering that quad simplification is still a relatively time consuming process. Quality is also drastically improved, as more regular vertices are produced (see Fig. A.1).

The most striking gain in term of quality occurs when the tri-mesh contains regions tesselated with square isosceles triangles (Fig. A.1, bottom). Tri-meshes falling in this category are not uncommon in practice. They occur, for example, with models obtained by March-

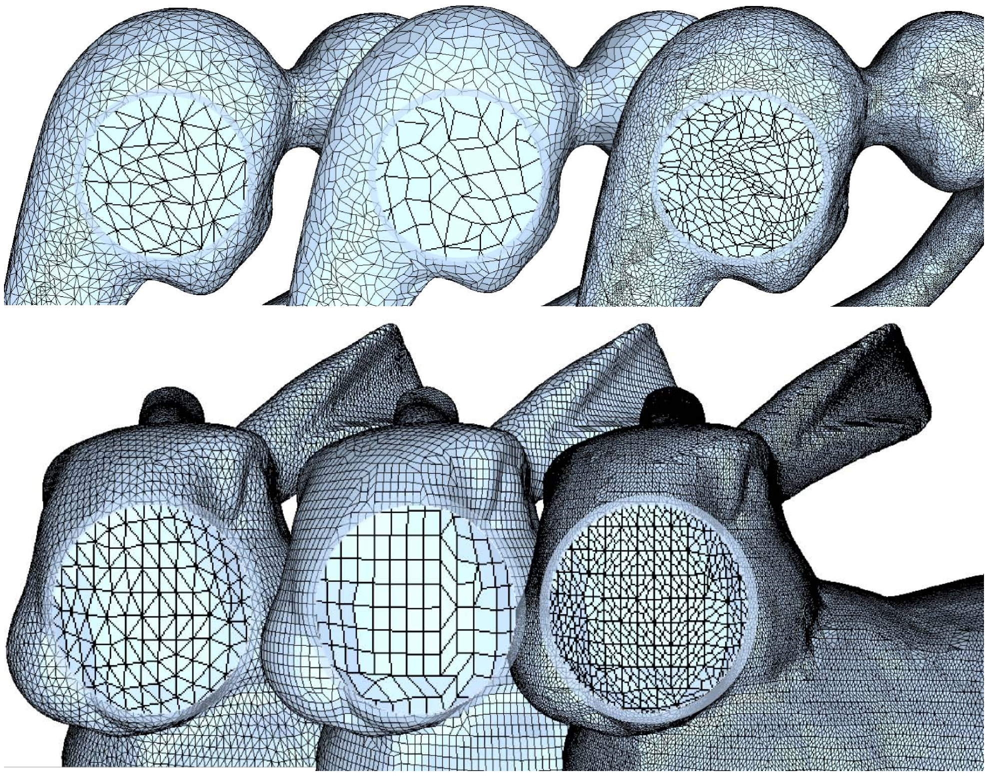


Figure A.1: *Comparison of Tri-to-Quad conversion methods. Left: input triangle mesh. Middle: quad mesh built with no complexity increase scheme. Right: quad mesh built with one step of Catmull-Clark subdivision.*

ing Cubes-like algorithms (in presence of flat regions with almost any orientation), with models obtained by zippering together re-triangulated depth scans or height fields, with procedural meshes, with CAD models, etc. For example, at least two among the most common tri-meshes used as benchmarks, Stanford bunny and Fandisk, feature this kind of structure.

Appendix B

Image Retargeting User Study

We conducted a user study with 305 participants, following the paired-comparisons protocol of [RGSS10]. Eight methods have been compared: manual crop (CR), nonhomogeneous warping (WARP) [WGCO07], Scale-and- Stretch (SNS) [WTSL08], MULTIOP [RSA09], shift-maps (SM) [PKVP09], streaming video (SV) [KLHG09], energy-based deformation (LG) [KFG09] and our algorithm (AA). All datasets in the study have been created by the authors of the respective methods, manually tweaking parameter values and sometimes the saliency to show the strengths of their retargeting algorithm and produce the best possible result.

The benchmark is made of 37 images and every image is tagged with one or more of the following attributes: people and faces, lines and/or clear edges, evident foreground objects, texture elements or repeating patterns, specific geometric structures, and symmetry. To better understand the strength and weakness of every retargeting algorithm, all the statistics of this study are grouped according to these attributes, and we also give the aggregate results for the entire dataset.

We note that the study participants had no reason to prefer a retargeted image over a (manually) cropped one since the study did not place the images in any semantic context. This biases the study in favor of manual cropping as it does not introduce any distortion. For this reason, cropping should be considered as a reference, not as a proper retargeting algorithm (for more details see the original paper [RGSS10]).

The gathered data is attached to the submission in the form of a MySQL database that uses the RetargetMe schema. Scripts to automatically analyze the data can be found at the RetargetMe website.

	Total	Lines/ Edges	Faces/ People	Texture	Foreground Objects	Geometric Structure	Symmetry
CR	2106	1376	973	308	1119	895	310
SV	1926	1274	745	287	908	**850**	340
MULTIOP	1826	1189	761	**314**	879	746	336
AA	**2000**	**1320**	**911**	295	**1055**	790	**350**
SCL	1019	751	323	192	383	512	214
SM	1429	985	569	250	698	650	185
SNS	1226	829	408	212	590	543	162
WARP	900	676	350	158	416	390	119

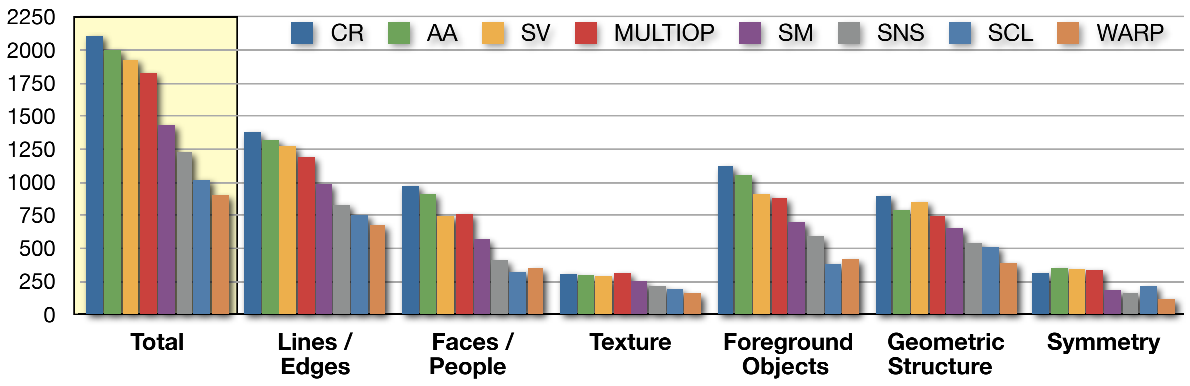


Figure B.1: *The number of votes for the eight methods considered in our user-study. A method gets a vote if a user picked a result by that method in a single paired-comparison question.*

Votes and Ranking Figure B.1 provides the study statistics, showing that our deformation subspace is a good choice for content-aware retargeting. Our results have ranked higher than the other six state-of-the-art methods. In particular, they achieved a quality slightly superior to SV [KLHG09], while being simpler to implement, faster and not requiring a GPU implementation to obtain interactive frame rates. Our study is in accordance with the original [RGSS10], providing further validation of the consistency of users' preferences.

Table B.1 shows the ranking according to the rank product method [RGSS10] (the smaller the number, the better).

CR	AA	SV	MULTIOP	SM	SNS	SCL	WARP
1.41	2.04	2.88	2.88	5.15	6.32	6.92	7.65

Table B.1: *Rank product of all methods.*

Our method consistently scores higher than the other retargeting methods with the exception of manual cropping.

Agreement and Statistical Significance We computed the Kendall coefficients of agreement u [KB40] to study the similarity of choices between participants. All participants would be in complete agreement if they voted the same way, and then $u = 1$. The minimum value of u is attained by an even distribution of answers and is given by $u = -\frac{1}{8}$ in our case.

The coefficients of agreement are shown in Table B.2, and they indicate that users have more agreement when people or strong symmetries are present in a scene. It is interesting to note that the participants reacted inconsistently to distortion in textures and geometric structures.

Lines / Edges	Faces / People	Texture	Foreground Objects	Geometric Structures	Symmetry	Total
0.113	0.261	0.090	0.216	0.113	0.220	0.148

Table B.2: *Coefficients of agreement.*

The statistical significance of the coefficients can be determined by testing the null hypothesis that the comparisons are assigned randomly (no agreement amongst users). A χ^2 test shows that the coefficients of agreement are statistically significant at the significance level of 0.01 in all seven categories.

Grouping Following [RGSS10], we group the 8 methods in statistically equivalent groups. Two methods in the same group are considered indistinguishable, since the difference in the votes they received is not sufficient to elect a clear winner.

Figure B.2 shows the groups computed over the entire survey (marked as "Aggregate") and for each attribute.

Note that our method is considered indistinguishable from CR, showing that the participants' preference towards it is strong. The grouping for the other methods is similar to the previous study. The main competitor of our method is SV, that always ranks lower than our method or is statistically indistinguishable from it.

Summary Our study shows that the space of axis-aligned deformations is a good candidate for content-aware image retargeting. Furthermore, our study validates the study of [RGSS10], since our results are very similar to theirs. We release all the gathered data to the public in the hope that it will be used by other researchers to validate their results.

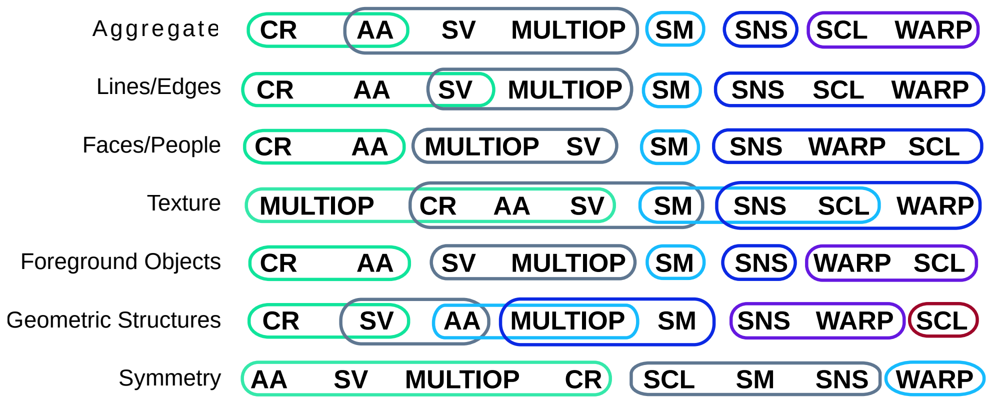

Aggregate
CR AA SV MULTIOP SM SNS SCL WARP
Lines/Edges
CR AA SV MULTIOP SM SNS SCL WARP
Faces/People
CR AA MULTIOP SV SM SNS WARP SCL
Texture
MULTIOP CR AA SV SM SNS SCL WARP
Foreground Objects
CR AA SV MULTIOP SM SNS WARP SCL
Geometric Structures
CR SV AA MULTIOP SM SNS WARP SCL
Symmetry
AA SV MULTIOP CR SCL SM SNS WARP

Figure B.2: *Grouping of the methods in statistically indistinguishable groups.*

Appendix C

Discrete Laplacian Operator

The continuos Laplace-Beltrami operator can be discretized at a mesh vertex v_i by a linear combination of the function values at the center vertex v_i and its one-ring neighbors v_j (see [BKP$^+$10]):

$$\nabla f(v_i) = w_i \sum_{v_j \in N_1(v_i)} w_{ij}(f(v_j) - f(v_i))$$

with $w_i = \frac{1}{2A_i}$ and $w_{ij} = (cot\ \alpha_{ij} + cot\ \beta_{ij})$. α_{ij} and β_{ij} are the two angles opposite to the edge incident on vertices i and j, as shown in Figure C.1. $N_1(v_i)$ is the unordered set of vertices in the 1-ring of v_i, i.e. the vertices connected by an edge to v_i. A_i is the voronoi area (or an approximation of it) of the vertex v_i.

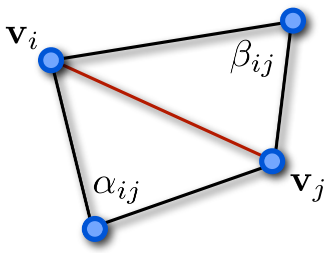


Figure C.1: *The two angles opposed to the edge incident on v_i and v_j*

In matrix notation, we can write the discrete laplacian operator as follows:

$$\begin{pmatrix} \nabla f(v_1) \\ \vdots \\ \nabla f(v_n) \end{pmatrix} = \mathbf{DM} \begin{pmatrix} f(v_1) \\ \vdots \\ f(v_n) \end{pmatrix}$$

with $\mathbf{D} = diag(w_1, \ldots, w_n)$ is a diagonal matrix with the weights w_i, and $\mathbf{M}$ is a symmetric matrix of the form:

$$m_{ij} = \begin{cases} -\sum_{v_k \in N_1(v_i)} w_{ik}, & \text{if i = j,} \\ w_{ij}, & \text{if } v_j \in N_1(v_i), \\ 0, & \text{otherwise.} \end{cases}$$

The matrix $\mathbf{L} = \mathbf{DM}$ is usually called *Laplacian matrix* or *Cotangent matrix*.

Since the computation of the cotangent of angles can introduce numerical errors, it is better to avoid to explictly compute the cotan weights using trigonometric functions. The following pseudo-code computes the cotangent of the three internal angles of a triangle in a robust way, using only products and additions. The angles α, β and γ are the angles incident on the vertices v_1,v_2 and v_3, respectively (see Figure C.2).

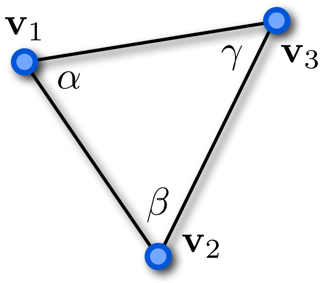


Figure C.2: *The notation used in Algorithm 2*

Algorithm 2 Compute cotangent of the angles of a triangle with vertices v_1, v_2 and v_3

```
1: $v_{12} = v_2 - v_1$
2: $v_{13} = v_3 - v_1$
3: $v_{23} = v_3 - v_2$
4: $A = ||v_{12} \times v_{13}||_2$
5: $\cot \alpha = (v_{12} \cdot v_{13})/A$
6: $\cot \beta = (v_{23} \cdot -v_{12})/A$
7: $\cot \gamma = (-v_{23} \cdot -v_{13})/A$
```

Appendix D

STIV and subspace gradients

STIV gradients

We present the gradient of the inner operation of Equation 5.3 with respect to a single vertex. Directly taking the derivative with respect to $\mathbf{q}_j^{(n)}$ gives

$$\frac{\partial V}{\partial \mathbf{q}_j^{(n)}} = \frac{\partial w_j}{\partial \mathbf{q}_j^{(n)}}(1-t_i)\Delta \mathbf{r}_i^{(n)} \cdot \hat{\mathbf{n}} +$$

$$w_i \frac{-\partial t_i}{\partial \mathbf{q}_j^{(n)}} \Delta \mathbf{r}_i^{(n)} \cdot \hat{\mathbf{n}} + w_i(1-t_i)\delta_{ij}\mathbf{I}_3\hat{\mathbf{n}} + w_i(1-t_i)\Delta \mathbf{r}_i^{(n)} \frac{\partial \hat{\mathbf{n}}}{\mathbf{q}_j^{(n)}},$$

with $\partial t_i / \partial \mathbf{q}_j^{(n)} = w_j \mathbf{n}$. Note that $\mathbf{n}$ is the un-normalized cross product of the two edges comprising the triangle.

$$\frac{\partial \hat{\mathbf{n}}_i}{\partial \mathbf{q}_j^{(n)}} = \sum_{k=0,1,2} \left(\frac{\mathbf{e}_k(t_i) \times \hat{\mathbf{n}}}{\|\mathbf{n}\|} \otimes \hat{\mathbf{n}} \right) \left(\frac{\partial t_i}{\partial \mathbf{q}_j^{(n)}} \otimes \Delta \mathbf{q}_j^{(n)} \right),$$

where $\mathbf{e}_k$ is the edge opposite vertex k, and $\partial \alpha_i / \partial \mathbf{q}_j^{(n)}$ are derivatives for the barycentric coordinates of a point in a triangle.

Subspace gradients

Subdivision surfaces. With subdivision surfaces, the control vertices $\mathbf{p}^{(n)}$ are updated, followed by re-computation using the subdivision matrix

$$\mathbf{q}^{(n)} = \mathbf{S}\mathbf{p}^{(n)}.$$

In this case, the linear operator $\mathbf{S}$ is the function f, so $\partial f/\partial \mathbf{p}^{(n)}$ is simply $\mathbf{S}$. Left multiplying by the original gradients gives the subspace gradients.

Laplacian editing. Laplacian surface editing intrinsically captures surface shape using differential coordinates, $\delta^{(0)} = \mathbf{L}\mathbf{q}^{(0)}$. Then, for a particular edit of the handle $\mathbf{p}^{(n)}$, we solve for the new positions using the formula

$$\tilde{\mathbf{L}}\mathbf{q}^{(n)} = \begin{pmatrix} \delta^{(0)} \\ \mathbf{p}^{(n)} \end{pmatrix},$$

where $\tilde{\mathbf{L}}$ is the augmented Laplacian matrix, as described in Sorkine *et. al.* [SCOL+04]. Ignoring numerical instability for a moment, we can re-write this as

$$\mathbf{q}^{(n)} = (\tilde{\mathbf{L}}^T\tilde{\mathbf{L}})^{-1}\tilde{\mathbf{L}}^T\mathbf{S}_L \begin{pmatrix} \delta^{(0)} \\ \mathbf{p}^{(n)} \end{pmatrix}. \tag{D.1}$$

In this form, the derivative is $(\tilde{\mathbf{L}}^T\tilde{\mathbf{L}})^{-1}\tilde{\mathbf{L}}^T\mathbf{S}_L$, where $\mathbf{S}_L$ is a "selector" matrix containing the identity matrix in the subset corresponding to $\mathbf{p}^{(n)}$, and zeros elsewhere.

Rigid motion. We can write the gradients in terms of the six degrees of freedom representing the center of mass of an object to obtain rigid motion, implementing the *rigid* editing mode. Let $\mathbf{p}^{(n)} = (\mathbf{x}_{cm}\ \theta_{cm})^T$, represent these six degrees of freedom.

Any point i on a rigid body has its location in *body* coordinates specified by $\mathbf{r}_i = \mathbf{q}_i^{(0)} - \mathbf{x}_{\mathrm{cm}}^{(0)}$. and the coordinates at any given point are given by $\mathbf{q}_i^{(n)} = \mathbf{x}_{\mathrm{cm}} + \mathbf{R}_{\mathrm{cm}}\mathbf{r}_i$, where $\mathbf{R}_{\mathrm{cm}}$ is the 3 DOF rigid body rotation coordinates expressed in matrix form.

The partial derivative of a vertex i with respect to the center of mass, is then given by the 3×6 matrix

$$\frac{\partial \mathbf{q}^{(n)}}{\partial \mathbf{p}^{(n)}} = \left(\mathbf{I}_3 \quad -\mathbf{r}^*\right),$$

where $\mathbf{I}_3$ is the 3×3 identity matrix, and $-\mathbf{r}^*$ is the skew-symmetric cross product matrix.

Free-form deformations. Free-form deformations express every point of $\mathbf{q}^{(n)}$ as a convex combinations of the points in $\mathbf{p}^{(n)}$, that is:

$$\mathbf{q}^{(n)} = \mathbf{M}\mathbf{p}^{(n)}.$$

The linear operator $\mathbf{M}$ can be build using different basis functions, in our case we used tricubic B-splines. The subspace gradients are computed in the same way as for subdivision surfaces.

Bibliography

[3dc] 3D Coat. `http://3d-coat.com/`.

[ACSD+03] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22(3):485–493, July 2003.

[ACWK06] A. Angelidis, M.P. Cani, G. Wyvill, and S. King. Swirling-sweepers: Constant-volume modeling. *Graphical Models*, 68(4):324–332, 2006.

[Ado10] Adobe Systems Inc. Photoshop CS5, July 2010. http://www.adobe.com/photoshop/.

[AFC+10] Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez, and Paul G. Kry. Volume contact constraints at arbitrary resolution. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 82:1–82:10, New York, NY, USA, 2010. ACM.

[AP97] M. Anitescu and F.A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14(3):231–247, 1997.

[APH11] G. Aldrich, D. Pinskiy, and B. Hamann. Collision-driven volumetric deformation on the gpu. *Eurographics 2011*, 2011.

[ASK+05] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. *ACM Trans. Graph.*, 24:408–416, July 2005.

[ATC+08] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 44:1–44:10, New York, NY, USA, 2008. ACM.

[Bar89] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Proc. SIGGRAPH*, pages 223–232, 1989.

[Bar94] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proc. SIGGRAPH*, pages 23–34, 1994.

[BBK06] Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching. *Proc. Natl. Acad. Sci. USA*, 103(5):1168–1172, 2006.

[BBS02] M.J. Borden, S.E. Benzley, and J.F. Shepherd. Hexahedral sheet extraction. In *Proc. 11th Int. Meshing Roundt.*, pages 147–152, 2002.

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.*, 21(3):594–603, 2002.

[BJ08] J. Barbič and D.L. James. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics*, pages 39–52, 2008.

[BKP$^+$10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Levy. *Polygon Mesh Processing.* AK Peters, 2010.

[Ble] Blender. `http://www.blender.org/`.

[BLK11] David Bommes, T. Lempfer, and Leif Kobbelt. Global structure optimization of quadrilateral meshes. *Computer Graphics Forum*, 30(2):375?–384, 2011.

[Blo09] Philippe Block. *Thrust Network Analysis: Exploring Three-dimensional Equilibrium.* Dissertation, Massachusetts Institute of Technology, 2009.

[BMRJ04] I. Boier-Martin, H. Rushmeier, and J Jin. Parametrization of triangle meshes over quadrilateral domains. In *Proceedings Symposium on Geometry Processing*, 2004.

[BP07] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH ’07, New York, NY, USA, 2007. ACM.

[BPGK06] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt. Primo: coupled prisms for intuitive surface modeling. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 11–20. Eurographics Association, 2006.

[Bri93] E. Brisson. Representing geometric structures in d dimensions: Topology and order. *Discrete and Computational Geometry*, 9:387–426, 1993.

[BS07] M. Botsch and O. Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, pages 213–230, 2007.

[BSFG09] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. In *Proc. SIGGRAPH*, 2009.

[bsu] B Surfaces. `http://www.bsurfaces.info/`.

[BSW83] R.E. Bank, A.H. Sherman, and A. Weiser. Refinement algorithms and data structures for regular local mesh refinement. In R. Stepleman, editor, *Scientific Computing*, pages 3–17. IMACS/North Holland, 1983.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[BZK09] D. Bommes, H. Zimmer, and L. Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):1–10, 2009.

[Cam90] Stephen Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation*, 6:291–302, 1990.

[Cas08] I. Castaño. Next-generation hardware rendering of displaced subdivision surfaces. SIGGR2008 - Exhibitor Tech Ses., 2008.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10:350–355, 1978.

[CCR08] P. Cignoni, M. Corsini, and G. Ranzuglia. Meshlab: an open-source 3d mesh processing system. *ERCIM News (73) - `http://meshlab.sourceforge.net/`*, pages 45–46, 2008.

[CDPB08] D. Cailliere, F. Denis, D. Pele, and A. Baskurt. 3d mirror symmetry detection using hough transform. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 1772–1775. IEEE, 2008.

[CDS10] Keenan Crane, Mathieu Desbrun, and Peter Schröder. Trivial connections on discrete surfaces. *Computer Graphics Forum*, 29(5):1525–1533, July 2010.

[CFK$^+$10] Renjie Chen, Daniel Freedman, Zachi Karni, Craig Gotsman, and Ligang Liu. Content-aware image resizing by quadratic programming. In *Proc. NORDIA*, 2010.

[CMLZ08] Guoning Chen, Konstantin Mischaikow, Robert S. Laramee, and Eugene Zhang. Efficient morse decompositions of vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 14:848–862, 2008.

[CMS97] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers and Graphics*, 22:37–54, 1997.

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *ACM Trans. Graph. (SIGGRAPH)*, pages 905–914, New York, NY, USA, 2004. ACM.

[CWQ$^+$04] Kin-Shing D. Cheng, Wenping Wang, Hong Qin, Kwan-Yee K. Wong, Huaiping Yang, and Yang Liu. Fitting subdivision surfaces to unorganized point data using sdm. In *PG '04: Proc. of the Computer Graphics and Applications, 12th Pacific Conference*, pages 16–24, Washington, DC, USA, 2004. IEEE Computer Society.

[DBG$^+$06] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J.C. Hart. Spectral surface quadrangulation. *ACM Trans. Graph.*, 25(3):1057–1066, 2006.

[DEGN98] T.K. Dey, H. Edelsbrunner, S. Guha, and D.V. Nekhayev. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) (N.S*, 66:23–45, 1998.

[DH94] T. Delmarcelle and L. Hesselink. The topology of symmetric, second-order tensor fields. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 140–147, 1994.

[DKG05] S. Dong, S. Kircher, and M. Garland. Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Comput. Aided Geom. Des.*, 22(5):392–423, 2005.

[DKT98] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 85–94, New York, NY, USA, 1998. ACM.

[DLG90] N. Dyn, D. Levin, and J.A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, April 1990.

[DS78] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10:356–360, 1978.

[DSC09a] J. Daniels, C.T. Silva, and E. Cohen. Localized quadrilateral coarsening. *Comput. Graph. Forum*, 28(5):1437–1444, 2009.

[DSC09b] Joel Daniels, Cláudio T. Silva, and Elaine Cohen. Semi-regular quadrilateral-only remeshing from simplified base domains. *Comput. Graph. Forum*, 28(5):1427–1435, 2009.

[DSSC08] J. Daniels, C.T. Silva, J. Shepherd, and E. Cohen. Quadrilateral mesh simplification. *ACM Trans. Graph.*, 27(5):1–9, 2008.

[DWS$^+$97] M.A. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes.

In *Proceedings IEEE Visualization '97*, pages 81–88. IEEE, October 1997.

[Ede87] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[EGKT08] D. Eppstein, M.T. Goodrich, E. Kim, and R. Tamstorf. Motorcycle graphs: Canonical quad mesh partitioning. *Computer Graphics Forum*, 27(5):1477–1486, July 2008.

[eig] Eigen library - sparse direct solvers.

[EML09] Christian Eisenacher, Quirin Meyer, and Charles Loop. Real-time view-dependent rendering of parametric surfaces. In *I3D '09: Symposium on Interactive 3D Graphics and Games*, pages 137–143, New York, NY, USA, 2009. ACM.

[Eri04] Christer Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3D Technology).* Morgan Kaufmann, December 2004.

[Far88] Gerald Farin. *Curves and surfaces for computer aided geometric design: a practical guide.* Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[Far96] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press, Boston, 4. edition, 1996.

[FBAF08] François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou. Image-based collision detection and response between arbitrary volumetric objects. In *ACM Siggraph/Eurographics Symposium on Computer Animation, SCA 2008, July, 2008*, Dublin, Irlande, July 2008.

[Fic65] G. Fichera. Elastostatics problems with unilateral constraints: the Signorini problem with ambiguous boundary conditions. *Seminari 1962-1963 di analisi, algebra, geometria e topologia*, page 613, 1965.

[FM11] B. Farb and D. Margalit. *A primer on mapping class groups.* Princeton Univ Press, 2011.

[GAK10] D. Ghosh, N. Amenta, and M. Kazhdan. Closed-form blending of local symmetries. *Computer Graphics Forum*, 29(5):1681–1688, 2010.

[GD01] J.E. Gain and N.A. Dodgson. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics*, pages 289–298, 2001.

[Gee08] K. Gee. Direct3d 11 tessellation. GameFest - Microsoft game technology conference, 2008.

[GG06] T. D. Gatzke and C. M. Grimm. Estimating curvature on triangular meshes.

International Journal on shape Modeling, 12:1–29, 2006.

[GGH02] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. In Tom Appolloni, editor, *SIGGRAPH*, pages 355–361. ACM, 2002.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *ACM Trans. Graph. (SIGGRAPH)*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[GPF09] A. Golovinskiy, J. Podolak, and T. Funkhouser. Symmetry-aware mesh processing. *Mathematics of Surfaces XIII*, pages 170–188, 2009.

[GSCO06] Ran Gal, Olga Sorkine, and Daniel Cohen-Or. Feature-aware texturing. In *Proc. EGSR*, pages 297–303, 2006.

[GSCO07] R. Gal, A. Shamir, and D. Cohen-Or. Pose-oblivious shape signature. *IEEE Trans. on Visualization and Computer Graphics*, 13(2):261–271, 2007.

[GST09] M. Gissler, R. Schmedding, and M. Teschner. Time-critical collision handling for deformable modeling. *Computer Animation and Virtual Worlds*, 20(2-3):355–364, 2009.

[HDD$^+$93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proc. SIGGRAPH '93*, pages 19–26, New York, NY, USA, 1993. ACM.

[HDD$^+$94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *ACM Trans. Graph. (SIGGRAPH)*, pages 295–302, New York, NY, USA, 1994. ACM.

[Her82] H. Hertz. Ueber die Berührung fester elastischer Körper. *Journal für die reine und angewandte Mathematik (Crelle's Journal)*, 1882(92):156–171, 1882.

[HL93] Josef Hoschek and Dieter Lasser. *Fundamentals of computer aided geometric design.* A. K. Peters, Ltd., Natick, MA, USA, 1993. Translator-Schumaker, Larry L.

[Hop96] H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996.

[HPS08] K. Hormann, K. Polthier, and A. Sheffer. Mesh parameterization: Theory and practice. In *SIGGRAPH Asia 2008 Course Notes*, number 11, pages v+81, Singapore, December 2008. ACM Press.

[HPSZ11] David Harmon, Daniele Panozzo, Olga Sorkine, and Denis Zorin. Interference-aware geometric modeling. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 137:1–137:10, New York, NY, USA, 2011. ACM.

[HPW05] K. Hildebrandt, K. Polthier, and M. Wardetzky. Smooth feature lines on surface meshes. In *Proc. 3rd Eurographics Symp. on Geom. Proc.*, page 85, 2005.

[HTG04] B. Heidelberger, M. Teschner, and M. Gross. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG*, 12(3):145–152, 2004.

[HVS+09] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. Asynchronous contact mechanics. *ACM Trans. Graph.*, 28:87:1–87:12, 2009.

[HVTG08] David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. Robust treatment of simultaneous collisions. *ACM Trans. Graph.*, 27(3):23:1–23:4, 2008.

[HZ00] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

[HZM+08a] J. Huang, M. Zhang, J. Ma, X. Liu, L. Kobbelt, and H. Bao. Spectral quadrangulation with orientation and alignment control. *ACM Trans. Graph.*, 27(5):1–9, 2008.

[HZM+08b] Jin Huang, Muyang Zhang, Jin Ma, Xinguo Liu, Leif Kobbelt, and Hujun Bao. Spectral quadrangulation with orientation and alignment control. *ACM Trans. Graph.*, 27(5):147, 2008.

[IGG01] Martin Isenburg, Stefan Gumhold, and Craig Gotsman. Connectivity shapes. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 135–142, Washington, DC, USA, 2001. IEEE Computer Society.

[IKN98] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20:1254–1259, 1998.

[JBPS11] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 30(4):78:1–78:8, 2011.

[JS11] Alec Jacobson and Olga Sorkine. Stretchable and twistable bones for skeletal shape deformation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA)*, 30(6):165:1–165:8, 2011.

[JSW05] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 561–566, New York, NY, USA, 2005. ACM.

[JT70] MA Jenkins and JF Traub. A three-stage algorithm for real polynomials using quadratic iteration. *SIAM Journal on Numerical Analysis*, 7(4):545–566, 1970.

[KAG$^+$09] M. Kazhdan, N. Amenta, S. Gu, D.F. Wiley, and B. Hamann. Symmetry restoration by stretching. In *Canadian Conference on Computational Geometry*. Citeseer, 2009.

[Kan01] Takashi Kanai. Meshtoss: Converting subdivision surfaces from dense meshes. In *Proc. of the Vision Modeling and Visualization Conference*, pages 325–332. Aka GmbH, 2001.

[KB40] M G Kendall and B Babington-Smith. On the method of paired comparisons. *Biometrica*, 31:324–345, 1940.

[KCB09] Pierre Kraemer, David Cazier, and Dominique Bechmann. Extension of half-edges for the representation of multiresolution subdivision surfaces. *Vis. Comput.*, 25(2):149–163, 2009.

[KEP05] Danny M. Kaufman, Timothy Edmunds, and Dinesh K. Pai. Fast frictional dynamics for rigid bodies. *ACM Trans. Graph.*, 24:946–956, 2005.

[KFG09] Zachi Karni, D. Freedman, and Craig Gotsman. Energy-based image deformation. *Comput. Graph. Forum*, 28(5), 2009.

[Kin97] P. Kinney. Cleanup: Improving quadrilateral finite element meshes. In *6th Int. Meshing Roundt.*, pages 449–461, 1997.

[KKL02] S.J. Kim, C.H. Kim, and D. Levin. Surface simplification using a discrete curvature norm. *Computers & Graphics*, 26(5):657–663, 2002.

[KL03] J. Kim and S. Lee. Transitive mesh space of a progressive mesh. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):463–480, 2003.

[KLCF10] Vladimir G. Kim, Yaron Lipman, Xiaobai Chen, and Thomas Funkhouser. Mbius Transformations For Global Intrinsic Symmetry Analysis. *Computer Graphics Forum*, 29(5):1689–1700, 2010.

[KLF11] Vladimir G. Kim, Yaron Lipman, and Thomas Funkhouser. Blended intrinsic maps. *ACM Trans. Graph.*, 30:79:1–79:12, August 2011.

[KLHG09] Philipp Krähenbühl, Manuel Lang, Alexander Hornung, and Markus Gross. A system for retargeting of streaming video. *ACM Trans. Graph.*, 28(5), 2009.

[KLS03] Andrei Khodakovsky, Nathan Litke, and Peter Schröder. Globally smooth parameterizations with low distortion. *ACM Trans. Graph.*, 22(3):350–357, 2003.

[KMDZ09] Denis Kovacs, Jason Mitchell, Shanon Drone, and Denis Zorin. Real-time creased approximate subdivision surfaces. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 155–160, New York, NY, USA, 2009. ACM.

[KNP07] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover surface parameterization using branched coverings. *Computer Graphics Forum*, 26(3):375–384, 2007.

[Kob00] L. Kobbelt. $\sqrt{3}$ subdivision. In *Proceedings ACM SIGGRAPH 2000*, pages 103–112, 2000.

[Kos65] J.L. Koszul. *Lectures on groups of transformations*, volume 32 of *Lectures on Mathematics*. Tata Institute of Fundamental Research, Bombay, India, 1965.

[KS04] V. Kraevoy and A. Sheffer. Cross-parameterization and compatible remeshing of 3d models. *ACM Transactions on Graphics (Proc. SIGGRAPH 2004)*, 2004.

[KSJP08] Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K. Pai. Staggered projections for frictional contact in multibody systems. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–11, New York, NY, USA, 2008. ACM.

[KSNS07] E. Kalogerakis, P. Simari, D. Nowrouzezahrai, and K. Singh. Robust statistical estimation of curvature on discretized surfaces. In *Symposium on Geometry Processing*, pages 13–22, 2007.

[LCDF10] Y. Lipman, X. Chen, I. Daubechies, and T. Funkhouser. Symmetry factored embedding and distance. In *ACM SIGGRAPH 2010 papers*, pages 1–12. ACM, 2010.

[LD09] Guillaume Lavoué and Florent Dupont. Technical section: Semi-sharp subdivision surface fitting based on feature lines approximation. *Comput. Graph.*, 33(2):151–161, 2009.

[LF09] Yaron Lipman and Thomas Funkhouser. Mobius voting for surface correspondence. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.

[LJFW08] J. Lin, X. Jin, Z. Fan, and C.C.L. Wang. Automatic polycube-maps. In *Proceedings of the 5th international conference on Advances in geometric modeling and processing*, pages 3–16. Springer-Verlag, 2008.

[LJW10] Ligang Liu, Yong Jin, and Qingbiao Wu. Realtime aesthetic image retargeting. In *Proc. Eurographics Workshop on Computational Aesthetic in Graphics, Visualization, and Imaging*, pages 1–8, 2010.

[LJX$^+$10] Y.K. Lai, M. Jin, X. Xie, Y. He, J. Palacios, E. Zhang, S.M. Hu, and X. Gu. Metric-driven rosy field design and remeshing. *Visualization and Computer Graphics, IEEE Transactions on*, 16(1):95–108, 2010.

[LKH08] Y.-K. Lai, L. Kobbelt, and S.-M. Hu. An incremental approach to feature aligned quad dominant remeshing. In *Proc. 2008 ACM Symp. on Sol. and Phys. Mod.*, pages 137–145, New York, NY, USA, 2008. ACM.

[LLCO08] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27:78:1–78:10, August 2008.

[LMH00] Aaron Lee, Henry Moreton, and Hugues Hoppe. Displaced subdivision surfaces. In *ACM Trans. Graph. (SIGGRAPH)*, pages 85–94, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Loo87] C. Loop. Smooth subdivision surfaces based on triangles. Master thesis, University of Utah, Dept. of Mathematics, 1987.

[Löt84] P. Lötstedt. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM Journal on Scientific and Statistical Computing*, 5:370–384, 1984.

[LRC$^+$02] D. Lübke, M. Reddy, J.D. Cohen, A. Varshney, B. Watson, and R. Hübner. *Level Of Detail for 3D Graphics.* Morgan Kaufmann, 2002.

[LRL06] Wan-Chiu Li, Nicolas Ray, and Bruno Lévy. Automatic and interactive mesh to t-spline conversion. In *SGP '06: Symposium on Geometry Processing*, pages 191–200. EG Association, 2006.

[LS00] Michael Lee and Hanan Samet. Navigating through triangle meshes implemented as linear quadtrees. *ACM Transactions on Graphics*, 19(2):79–121, 2000.

[LS08] Charles Loop and Scott Schaefer. Approximating catmull-clark subdivision surfaces with bicubic patches. *ACM Trans. Graph.*, 27(1):1–11, 2008.

[LSS$^+$98] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Comp. Graph. Proc.*, pages 95–104, 1998.

[LVJ05] Chang Ha Lee, Amitabh Varshney, and David W. Jacobs. Mesh saliency. In *ACM Trans. Graph. (SIGGRAPH)*, pages 659–666, New York, NY, USA, 2005. ACM.

[May] Autodesk maya. `http://usa.autodesk.com/`.

[MB10] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization, 2010. Manuscript.

[MC95] Brian Mirtich and John Canny. Impulse-based dynamic simulation. In *WAFR: Proceedings of the workshop on Algorithmic foundations of robotics*, pages 407–418, Natick, MA, USA, 1995. A. K. Peters, Ltd.

[MCK08] Tobias Martin, Elaine Cohen, and Mike Kirby. Volumetric parameterization and trivariate b-spline fitting using harmonic functions. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, SPM '08, pages 269–280, New York, NY, USA, 2008. ACM.

[MGP06] N.J. Mitra, L.J. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM Transactions on Graphics (TOG)*, 25(3):560–568, 2006.

[MHHR07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, 2007.

[Mit07] M. Pauly Mitra, N. L. Guibas L. Symmetrization. *ACM Transactions on Graphics*, 26(3), 2007.

[MJ98] H. Müller and R. Jaeschke. Adaptive subdivision curves and surfaces. In *Proceedings of Computer Graphics International '98*, pages 48–58, 1998.

[MK04] M. Marinov and L. Kobbelt. Optimization techniques for approximation with subdivision surfaces. In *SM '04: Proceedings of the ninth ACM symposium on Solid modeling and applications*, pages 113–122, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[MK05] M. Marinov and L. Kobbelt. Automatic generation of structure-preserving multi-resolution models. *CG Forum*, 24(3):479–486, 2005.

[MMTP04] W. Ma, X. Ma, S.-K. Tso, and Z. Pan. A direct approach for subdivision surface fitting from a dense triangle mesh. *Computer Aided Geometric Design*, 36(16):525–536, 2004.

[MNP08] A. Myles, T. Ni, and J. Peters. Fast parallel construction of smooth surfaces from meshes with tri/quad/pent facets. *Computer Graphics Forum*, 27(5):1365–1372, July 2008.

[MO06] C. Mendoza and C. O'Sullivan. Interruptible collision detection for deformable objects. *Computers & Graphics*, 30(3):432–438, 2006.

[Mod] Modo 301. `http://www.luxology.com`.

[MPKZ10] Ashish Myles, Nico Pietroni, Denis Kovacs, and Denis Zorin. Feature-aligned t-meshes. *ACM Trans. Graph.*, 29(4):1–11, 2010.

[MPT99] W.A. McNeely, K.D. Puterbaugh, and J.J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *Proc. SIGGRAPH*, pages 401–408, 1999.

[MS01a] Jérôme Maillot and Jos Stam. A unified subdivision scheme for polygonal modeling. *Computer Graphics Forum*, 20(3):471–479, 2001.

[MS01b] Victor J. Milenkovic and Harald Schmidl. Optimization-based animation. In *Proc. SIGGRAPH*, pages 37–46, 2001.

[MS04] Facundo Mémoli and Guillermo Sapiro. Comparing Point Clouds . In *Proceedings Symposium on Geometry Processing 2004*, pages 33–42. Eurographics, 2004.

[MW88] M. Moore and J. Wilhelms. Collision detection and response for computer animation. In *Proc. SIGGRAPH*, pages 289–298, 1988.

[MZ55] D. Montgomery and L. Zippin. *Topological transformation groups*, volume 1. Interscience Publishers New York, 1955.

[NRP11] M. Nieser, U. Reitebuch, and K. Polthier. CubeCover - parameterization of 3d volumes. *Computer Graphics Forum*, 30(5):1397–1406, 2011.

[OBS04] Y. Ohtake, A. Belyaev, and H.P. Seidel. Ridge-valley lines on meshes via implicit surface fitting. In *International Conference on Computer Graphics and Interactive Techniques*, pages 609–612. ACM New York, NY, USA, 2004.

[OD99] C. OSullivan and J. Dingliana. Real-time collision detection and response using sphere-trees. In *15th Spring Conference on Computer Graphics*, pages 83–92, 1999.

[OFCD02] Robert Osada, Thomas Funkhouser, Bernard Chazelle, and David Dobkin. Shape distributions. *ACM Trans. Graph.*, 21:807–832, October 2002.

[OMMG10] Maks Ovsjanikov, Quentin Mérigot, Facundo Mémoli, and Leonidas Guibas. One Point Isometric Matching with the Heat Kernel. *Computer Graphics Forum*, 29(5):1555–1564, 2010.

[OSCS99] S.J. Owen, M.L. Staten, S.A. Canann, and S. Saigal. Q-morph: An indirect approach to advancing front quad meshing. *International Journal for Numerical Methods in Engineering*, 44(9):1317–1340, March 1999.

[OSG08] M. Ovsjanikov, J. Sun, and L. Guibas. Global intrinsic symmetries of shapes. *Computer graphics forum*, 27(5):1341–1348, 2008.

[PAH06] P.-O. Persson, M.J. Aftosmis, and R. Haimes. On the use of loop subdivision surfaces for surrogate geometry. In *Proceedings 15th International Meshing Roundtable*, pages 375–392, Birmingham (AL), USA, September 17-20 2006.

[PGR07] J. Podolak, A. Golovinskiy, and S. Rusinkiewicz. Symmetry-enhanced remeshing of surfaces. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 235–242. Eurographics Association, 2007.

[PKS10] S. Pabst, A. Koch, and W. Straßer. Fast and scalable CPU/GPU collision detection for rigid and deformable surfaces. In *Proc. Symposium on Geometry Processing*, pages 1605–1612, 2010.

[PKVP09] Y. Pritch, E. Kav-Venaki, and S. Peleg. Shift-map image editing. In *Proc. ICCV*, Sep-Oct 2009.

[PLH02] Helmut Pottmann, Stefan Leopoldseder, and Michael Hofer. Approximation with active b-spline curves and surfaces. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 8, Washington, DC, USA, 2002. IEEE Computer Society.

[PLPZ12] Daniele Panozzo, Yaron Lipman, Enrico Puppo, and Denis Zorin. Fields on symmetric surfaces. In *Proceedings of the 2012 SIGGRAPH Conference*, New York, NY, USA, 2012. ACM.

[PP09a] D. Panozzo and E. Puppo. Interpolatory adaptive subdivision for mesh lod editing. In *Proceedings GRAPP 2009 - International Conference on Computer Graphics Theory and Applications*, pages 70–75, Lisboa, Portugal, February 5–8 2009.

[PP09b] E. Puppo and D. Panozzo. RGB subdivision. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):295–310, 2009.

[PP10] Daniele Panozzo and Enrico Puppo. Adaptive lod editing of quad meshes. In *Afrigraph*, pages 7–16, 2010.

[PP11] Daniele Panozzo and Enrico Puppo. Implicit hierarchical quad-dominant meshes. *Comput. Graph. Forum*, 30(6):1617–1629, 2011.

[PPG04] Mark Pauly, Dinesh K. Pai, and Leonidas J. Guibas. Quasi-rigid objects in contact. In *Proc. SCA*, pages 109–119, 2004.

[PPT$^+$11] Daniele Panozzo, Enrico Puppo, Marco Tarini, Nico Pietroni, and Paolo Cignoni. Automatic construction of adaptive quad-based subdivision surfaces using fitmaps. *IEEE Transaction on Visualization and Computer Graphics*, 2011. http://doi.ieeecomputersociety.org/10.1109/TVCG.2011.28.

[Pro97] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *Proc. Computer Animation and Simulation*, pages

177–189. Springer Verlag, 1997.

[PS07] H.R. Pakdel and F.F. Samavati. Incremental subdivision for triangle meshes. *International Journal of Computational Science and Engineering*, 3(1):80–92, 2007.

[PSG$^+$06] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser. A planar-reflective symmetry transform for 3d shapes. *ACM Transactions on Graphics*, 25(3):549–559, 2006.

[PSS01] E. Praun, W. Sweldens, and P. Schröder. Consistent mesh parameterizations. *Proc. of SIGGRAPH 2001*, 2001.

[PTC09] N. Pietroni, M. Tarini, and P. Cignoni. Almost isometric mesh parameterization through abstract domains. *IEEE Trans. on Vis. and Comp. Graph.*, 2009.

[PTC10] Nico Pietroni, Marco Tarini, and Paolo Cignoni. Almost isometric mesh parameterization through abstract domains. *IEEE Transaction on Visualization and Computer Graphics*, 16(4):621–635, July/August 2010.

[PTSZ11] Nico Pietroni, Marco Tarini, Olga Sorkine, and Denis Zorin. Global parametrization of range image sets. *ACM Transactions on Graphics, Proceedings of SIGGRAPH Asia 2011*, 30(6), 2011.

[Pup98] E. Puppo. Variable resolution triangulations. *Computational Geometry*, 11(3-4):219–238, 1998.

[Pup07] E. Puppo. Dynamic adaptive subdivision meshes. In *2007 Israel-Italy Bi-National Conference on Shape Modeling and Reasoning for Industrial and Biomedical Application*, pages 60–64, Technion Haifa, Israel, May 2007.

[PWS12] Daniele Panozzo, Ofir Weber, and Olga Sorkine. Robust image retargeting via axis-aligned deformation. *Computer Graphics Forum (proceedings of EUROGRAPHICS)*, 31(2), 2012.

[PZ07] J. Palacios and E. Zhang. Rotational symmetry field design on surfaces. *ACM Trans. Graph.*, 26(3):55, 2007.

[RBBK07] D. Raviv, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Symmetries of non-rigid shapes. In *Proc. Non-rigid Registration and Tracking (NRTL) workshop. See Proc. of International Conference on Computer Vision (ICCV)*, October 2007.

[RBBK10] D. Raviv, A.M. Bronstein, M.M. Bronstein, and R. Kimmel. Full and partial symmetries of non-rigid shapes. *International journal of computer vision*, 89(1):18–39, 2010.

[RGPP11] Luigi Rocca, Nikolas De Giorgis, Daniele Panozzo, and Enrico Puppo. Fast neighborhood search on polygonal meshes. In Andrea F. Abate, Michele Nappi, and Genny Tortora, editors, *Eurographics Italian Chapter Conference*, pages 15–21, 2011.

[RGSS10] Michael Rubinstein, Diego Gutierrez, Olga Sorkine, and Ariel Shamir. A comparative study of image retargeting. *ACM Trans. Graph.*, 29(5), 2010.

[RLL$^+$06] N. Ray, W.-C. Li, B. Lévy, P. Alliez, and A. Sheffer. Periodic global parameterization. *ACM Trans. Graph.*, 2006.

[RSA08] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved seam carving for video retargeting. *ACM Trans. Graph.*, 27(3), 2008.

[RSA09] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Multi-operator media retargeting. *ACM Trans. Graph.*, 28(3):23, 2009.

[Rus10] R.M. Rustamov. Barycentric coordinates on surfaces. *Comput. Graph. Forum*, 29(5):1507–1516, 2010.

[RVAL09] Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. Geometry-aware direction field processing. *ACM Trans. Graph.*, 29(1):1–11, 2009.

[RVLL08] N. Ray, B. Vallet, W.C. Li, and B. Lévy. N-Symmetry Direction Field Design. *ACM Trans. Graph.*, 27:2, 2008.

[SA07] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Proc. Symposium on Geometry Processing*, pages 109–116, 2007.

[Sab04] M. Sabin. Recent progress in subdivision: a survey. In N.A. Dogdson, M.S. Floater, and M.A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 203–230. Springer-Verlag, 2004.

[Sam05] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[SAPH04] John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. Inter-surface mapping. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 2004.

[SB99] F.F. Samavati and R.H. Bartels. Multiresolution curve and surface representation by reversing subdivision rules. *Computer Graphics Forum*, 18(2):97–120, 1999.

[Sch79] H. Schwerdtfeger. *Geometry of complex numbers: circle geometry, Moebius transformation, non-euclidean geometry.* Dover Books on Mathematics Series. Dover, 1979.

[SCOL$^+$04] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.P. Seidel. Laplacian surface editing. In *Proc. Symposium on Geometry processing*, pages 175–184, 2004.

[SCSI08] Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani. Summarizing visual data using bidirectional similarity. In *Proc. CVPR*, 2008.

[SDW$^+$09] J.F. Shepherd, M.W. Dewey, A.C. Woodbury, S.E. Benzley, M.L. Staten, and S.J. Owen. Adaptive mesh coarsening for quadrilateral and hexahedral meshes. *Finite Elements in Analysis and Design*, 46(1-2):17 – 32, 2009.

[She02] Jonathan Richard Shewchuk. What is a good linear finite element? - interpolation, conditioning, anisotropy, and quality measures. In *Proc. of the 11th International Meshing Roundtable*, 2002. Unpublished extended version available at http://www.cs.berkeley.edu/ jrs/papers/elemj.pdf.

[SHHG01] S. Seeger, K. Hormann, G. Häusler, and G. Greiner. A sub-atomic subdivision approach. In B. Girod, H. Niemann, and H.-P. Seidel, editors, *Proceedings of Vision, Modeling and Visualization 2001*, pages 77–85, Berlin, 2001. Akademische Verlag.

[Sil] Silo 2. `http://www.nevercenter.com/`.

[SL03] J. Stam and C. Loop. Quad/triangle subdivision. *Computer Graphics Forum*, 22(1):79–85, 2003.

[SLF08] A. Selle, M. Lentine, and R. Fedkiw. A mass spring model for hair simulation. *ACM Trans. Graph.*, 27(3):64–64, 2008.

[SMAB02] F.F. Samavati, N. Mahdavi-Amiri, and R.H. Bartels. Multiresolution surfces having arbitrary topologies by a reverse doo subdivision method. *Computer Graphics Forum*, 21(2):121–136, 2002.

[Sny95] John M. Snyder. An interactive tool for placing curved surfaces without interpenetration. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 209–218, New York, NY, USA, 1995. ACM.

[Sor06] Olga Sorkine. Differential representations for mesh processing. *Computer Graphics Forum*, 25(4):789–807, 2006.

[SP86] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. In *Proc. SIGGRAPH*, pages 151–160, 1986.

[SS09] Ariel Shamir and Olga Sorkine. Visual media retargeting. In *ACM SIGGRAPH Asia Courses*, 2009.

[ST96] D. Stewart and J.C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *Intl. Journal for Numerical Methods in Engineering*, 39:2673–2691, 1996.

[Sta98a] J. Stam. Evaluation of Loop subdivision surfaces. In *ACM Trans. Graph. (SIGGRAPH)*, 1998.

[Sta98b] J. Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *ACM Trans. Graph. (SIGGRAPH)*, pages 395–404, 1998.

[STKK99] Hiromasa Suzuki, Shingo Takeuchi, Fumihiko Kimura, and Takashi Kanai. Subdivision surface fitting to a range of points. In *PG '99: Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, page 158, Washington, DC, USA, 1999. IEEE Computer Society.

[SW07] Scott Schaefer and Joe Warren. Exact evaluation of non-polynomial subdivision schemes at rational parameter values. In *Proc. 15th Pacific Conference on Computer Graphics and Applications*, pages 321–330, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[SWG$^+$03] Pedro V. Sander, Zoë J. Wood, Steven J. Gortler, John Snyder, and Hugues Hoppe. Multi-chart geometry images. In Leif Kobbelt, Peter Schröder, and Hugues Hoppe, editors, *Symposium on Geometry Processing*, volume 43 of *ACM International Conference Proceeding Series*, pages 146–155. Eurographics Association, 2003.

[TACSD06] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing quadrangulations with discrete harmonic forms. In *Proc. 4th Eurographics Symp. on Geom. Proc.*, pages 201–210, 2006.

[THCM04] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. Polycube-maps. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 853–860, New York, NY, USA, 2004. ACM.

[THM$^+$03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. VMV*, pages 47–54, 2003.

[Thu97] William P. Thurston. *Three-dimensional geometry and topology. Vol. 1*, volume 35 of *Princeton Mathematical Series.* Princeton University Press, Princeton, NJ, USA, 1997. Edited by Silvio Levy.

[TKZ$^+$04] Matthias Teschner, Stefan Kimmerle, Gabriel Zachmann, Bruno Heidelberger, Laks Raghupathi, Arnulph Fuhrmann, Marie-Paule Cani, François Faure, Nadia Magnenat-Thalmann, and Wolfgang Strasser. State-of-the-art report: Collision detection for deformable objects. In *Proc. Eurographics*, pages 119–139, 2004.

[TMLT11] Min Tang, Dinesh Manocha, Jiang Lin, and Ruofeng Tong. Collision-streams: Fast GPU-based collision detection for deformable models. In *I3D '11: Proceedings of the 2011 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 63–70, 2011.

[TMT10] Min Tang, Dinesh Manocha, and Ruofeng Tong. Fast continuous collision detection using deforming non-penetration filters. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 7–13, New York, NY, USA, 2010. ACM.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proc. SIGGRAPH*, pages 205–214, 1987.

[TPC$^+$10] M. Tarini, N. Pietroni, P. Cignoni, D. Panozzo, and Puppo E. Practical quad mesh simplification. *CG Forum (Eurographics 2010)*, 29(2):407–418, 2010.

[TPP$^+$11] Marco Tarini, Enrico Puppo, Daniele Panozzo, Nico Pietroni, and Paolo Cignoni. Simple quad domains for field aligned mesh parametrization. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 142:1–142:12, New York, NY, USA, 2011. ACM.

[TSH01] Xavier Tricoche, Gerik Scheuermann, and Hans Hagen. Continuous topology simplification of planar vector fields. In *IEEE Visualization*, 2001.

[UNC10] UNC. Self-ccd: Continuous collision detection for deforming objects, 2010.

[Vel03] L. Velho. Stellar subdivision grammars. In *Proceedings Eurographics Symposium on Geometry Processing*, 2003.

[VFTS06] W. Von Funck, H. Theisel, and H.P. Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.

[VG00] L. Velho and J. Gomes. Variable resolution 4-k meshes: Concepts and applications. *Computer Graphics Forum*, 19(4):195–214, 2000.

[VHWP12] Etienne Vouga, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. Design of self-supporting surfaces. *ACM Trans. Graphics*, 31, 2012. Proc. SIGGRAPH.

[VZ01] L. Velho and D. Zorin. 4-8 subdivision. *Computer-Aided Geometric Design*, 18:397–427, 2001.

[WB01a] K. Watanabe and A.G. Belyaev. Detection of salient curvature features on polygonal surfaces. In *CG Forum*, volume 20, pages 385–392. Citeseer, 2001.

[WB01b] Andrew Witkin and David Baraff. Physically based modeling: Course notes.

In *SIGGRAPH Courses*, 2001.

[WG09] T. Weinkauf and D. Günther. Separatrix Persistence: Extraction of Salient Edges on Surfaces Using Topological Methods. In *Computer Graphics Forum*, volume 28, pages 1519–1528. Blackwell Publishing Ltd, 2009.

[WGCO07] Lior Wolf, Moshe Guttmann, and Daniel Cohen-Or. Non-homogeneous content-driven video-retargeting. In *Proc. ICCV*, 2007.

[WL07] P Wriggers and Tod A Laursen. *Computational contact mechanics*, volume 498 of *CISM courses and lectures*. Springer, 2007.

[WLSL10] Yu-Shuen Wang, Hui-Chih Lin, Olga Sorkine, and Tong-Yee Lee. Motion-based video retargeting with optimized crop-and-warp. *ACM Trans. Graph.*, 29(4):article no. 90, 2010.

[Won05] S.K. Wong. *High performance virtual clothing dynamics*. PhD thesis, Hong Kong University of Science and Technology (People's Republic of China), 2005.

[WTSL08] Yu-Shuen Wang, Chiew-Lan Tai, Olga Sorkine, and Tong-Yee Lee. Optimized scale-and-stretch for image resizing. *ACM Trans. Graph.*, 27(5):118, 2008.

[WW02] J. Warren and H. Weimer. *Subdivision Methods for Geometric Design*. Morgan Kaufmann, 2002.

[XK99] Z. Xu and K. Kondo. Adaptive refinements in subdivision surfaces. In *Eurographics '99, Short papers and demos*, pages 239–242, 1999.

[XZT$^+$09] K. Xu, H. Zhang, A. Tagliasacchi, L. Liu, G. Li, M. Meng, and Y. Xiong. Partial intrinsic reflectional symmetry of 3d shapes. *ACM Transactions on Graphics (TOG)*, 28(5):138, 2009.

[ZCHM09] Guo-Xin Zhang, Ming-Ming Cheng, Shi-Min Hu, and Ralph R. Martin. A shape-preserving approach to image resizing. *Comput. Graph. Forum*, 28(7):1897–1906, 2009.

[ZHLB10] Muyang Zhang, Jin Huang, Xinguo Liu, and Hujun Bao. A wave-based anisotropic quadrangulation method. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 118:1–118:8, New York, NY, USA, 2010. ACM.

[ZS00] D. Zorin and P. Schröder, editors. *Subdivision for Modeling and Animation (SIGGRAPH 2000 Tutorial N.23 - Course notes)*. ACM Press, 2000.

[ZSS97] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 97), ACM Press*, 1997. 259-268.