DOI: 10.1111/j.1467-8659.2010.01746.x
Eurographics Symposium on Rendering 2010
Jason Lawrence and Marc Stamminger
(Guest Editors)
Volume 29 (2010), Number 4

Invisible Seams

Nicolas Ray[1] Vincent Nivoliers[1,3] Sylvain Lefebvre[1,2] Bruno Lévy[1]

[1] ALICE / INRIA Nancy Grand Est, LORIA [2] REVES / INRIA Sophia-Antipolis [3] Institut National Polytechnique de Lorraine

Abstract

Surface materials are commonly described by attributes stored in textures (for instance, color, normal, or displacement). Interpolation during texture lookup provides a continuous value field everywhere on the surface, except at the chart boundaries where visible discontinuities appear. We propose a solution to make these seams invisible, while still outputting a standard texture atlas. Our method relies on recent advances in quad remeshing using global parameterization to produce a set of texture coordinates aligning texel grids across chart boundaries. This property makes it possible to ensure that the interpolated value fields on both sides of a chart boundary precisely match, making all seams invisible. However, this requirement on the uv coordinates needs to be complemented by a set of constraints on the colors stored in the texels. We propose an algorithm solving for all the necessary constraints between texel values, including through different magnification modes (nearest, bilinear, biquadratic and bicubic), and across facets using different texture resolutions. In the typical case of bilinear magnification and uniform resolution, none of the texels appearing on the surface are constrained. Our approach also ensures perfect continuity across several MIP-mapping levels.

Categories and Subject Descriptors (according to ACM CCS): Computing Methodologies [I.3.7]: COMPUTER GRAPHICS—Three-Dimensional Graphics and Realism

1. Introduction

A major drawback of texture atlases is the seam artifacts: Chart boundaries produce visible discontinuities, even through simple bilinear magnification (see Figure 2, left). These visible seams are the result of discontinuities in the parameterization: Two neighboring points on the surface are not neighboring in the texture (see Figure 1). The interpolator – usually the graphics hardware – is unaware of this and interpolates in the texture, bleeding in incorrect information from texels outside the charts.

In this paper, our goal is to produce atlases without this drawback. While still being standard texture atlases, we guarantee that interpolating into the texture produces a continuous result along the surface. Our approach works across various magnification filters, from bilinear to bicubic, supports adaptive resolution – specified independently per–triangle – and MIP-mapping while still ensuring continuity of interpolation. A major advantage of producing a standard atlas is simplicity: Available accelerated hardware texture lookups can be used, complex shaders are not burdened by added complexity to critical texture accesses, and our approach is backward compatible with any renderer capable of texture mapping.

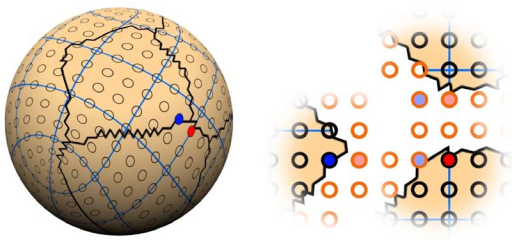


Figure 1: *Neighboring texels on the surface (left, red/blue dots) are not neighbors in the atlas (right). To make the seam invisible, texels have to align across chart boundaries and their colors have to be duplicated (light colored texels).*

Hiding the seams requires to reproduce in the texture atlas the color neighborhoods observed on the surface. This property cannot be enforced easily if the texel grids do not match across a chart boundary (see Figure 2). We thus build our atlas from a *grid-preserving parameterization* (GPP) of the surface (see Figure 5 right). These parameterizations map a $2D$ grid onto a semi-regular grid embedded in the surface, providing the necessary alignments across triangle edges.

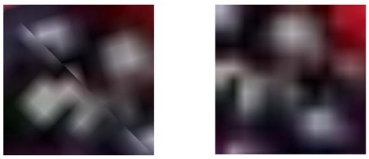


Figure 2: Left: *If texels grids are not aligned discontinuities are inevitable.* Right: *Aligned grids can provide continuity if the color of some texels in the texture are constrained.*

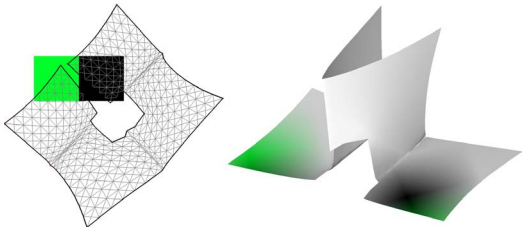


Figure 3: *The surface (right) is parameterized in a single chart (left). However, because the chart was formed before solving for texel value constraints, the green and black texels bleed onto independent parts of the surface.*

While this is a natural idea, building a proper texture atlas from a GPP is not straight-forward.

The standard approach for building a texture atlas is to cluster the surface in independent charts. The charts are individually parameterized and then packed in a rectangle image so as to use as many pixels as possible (see [FH05] for a survey). A GPP is quite different: It is defined per–triangle, with a transition function mapping triangles next to each other. It is thus not possible to form charts *prior* to parameterization. In addition, forming charts while ignoring interpolation constraints would lead to artifacts, as illustrated Figure 3.

It is also important to realize that even if texels align across chart boundaries, simply copying colors on both sides of a chart will not solve the problem in general. There are several specific cases, such as charts which are thin wrt. texel size (see Figure 4) or singularities of the parameterization (see Figure 8) which would break a naive approach. The number of failure cases significantly increases with more complex interpolation kernels and adaptive resolution.

Our main contribution is to rethink the parameterization pipeline to solve simultaneously for the color constraints and the chart formation, from an initial GPP. Through the use of equivalence classes for color constraints and by careful definition of the influence of each texel, our approach supports generic interpolation kernels (we demonstrate bilinear and bicubic), MIP-mapping and adaptive resolution texture maps in the same algorithm. The result of our approach is a regular texture atlas and a set of rules to enforce color constraints. After the user paints on the surface, these rules are applied. The atlas is then ready to be used as a regular texture.

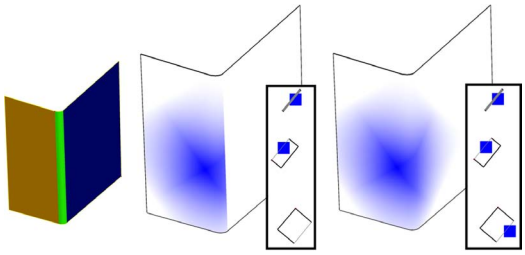


Figure 4: *Matching texel colors across chart boundaries does not ensure continuity. Here, a texel lying on the brown chart propagates its value to the texel on the green chart, but cannot reach the blue chart (middle). Our texel class approach resolve such problems (right).*

1.1. Previous work

Removing seam discontinuities Several approaches aim at reducing the perception of seams. For instance, Lefebvre and Hoppe [LH06] synthesize textures having matching colors across seams. Sheffer and Hart [SH02] hide the visible artifacts by optimizing for the seam positions. To ensure continuity across chart boundaries Carr *et al.* [CH04, CHCH06] and Purnomo *et al.* [PCK04] create a quad segmentation and map each chart to a square. Burley *et al.* [BL08] map a texture to each patch of a Catmull-Clark subdivision surface, reprogramming the renderer to obtain a visually seamless interpolation. Gonzalez *et al.* [GP09] use a dedicated fragment shader to match colors by triangulating the seams. Other schemes rely on non-planar domains [THCM04, YKH10] or store colors in a volume hierarchy surrounding the object [BD02, LD07]. We discuss the pros and cons of each method with respect to ours in Section 5.

Grid–preserving parameterizations To obtain an alignment of texel grids across chart boundaries, we expect a GPP as input. A GPP provides (u,v) coordinates for each facet corner. In the parametric domain facets typically overlap and are disconnected (see Figure 5, right). It is possible to navigate from one facet to another by applying, in the parametric domain, the *transition functions* bringing a facet half edge to its opposite half edge. (The opposite half edge belongs to the facet neighboring on the surface). Methods for GPP construction focus on having a restricted class of transition functions which are combination of *integer* translations and rotations of 90 degree (see Figure 5, left). These were initially used to produce quad remeshing of a surface [RLL*06, KNP07, BZK09, TACSD06], projecting a $2D$ grid on the surface. In this work, we compute GPPs using QuadCover [KNP07] steered by a smoothed estimate of the principal curvature directions [RVAL09].

2. Overview

Our approach inputs a mesh with grid–preserving texture coordinates (u,v) associated to the facet corners. Since this is

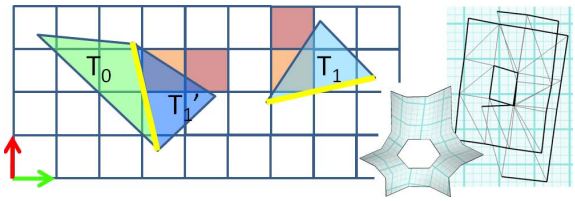


Figure 5: Left: *Matching the yellow half edge from triangle T_1 with the yellow half edge of triangle T_0 involves an integer translation and a 90 degree rotation (T_1' is T_1 through this transform). The same transform sends the colored texels overlapped by T_1 exactly to the colored texels overlapped by T_1'.* Right: *A GPP for the 3D model in the middle.*

Figure 6: *Interpolation cells (left, black square) for bilinear magnification mode are dual to the texel grid i.e. their corners are placed at texel centers. The influence area of the center texel (right, blue square) is the union of the four interpolation cells requiring the texel.*

the output of a GPP (see Section 1.1), the transition functions bringing each half edge of a facet to the half edge of its neighboring facet are composed of only integer translation and 90 degree rotations (see Figure 5). Our algorithm also inputs the target texture resolution specified by the user. The algorithm outputs new texture coordinates defining a one to one mapping together with a **stitch map** describing how to enforce the constraints on texel values. In the standard case, the stitch map is simply an indirection map [LH06].

Our method can be decomposed into two main steps: A first step (Section 3) considers each triangle – more generally *facet* – independently. We then detect texel value constraints across all facets. The second step (Section 4) produces larger charts by iteratively coalescing non–overlapping charts. The charts are finally packed together to obtain a texture atlas, and the stitch map is computed. After this point, any texture atlas using our uv's and having the corresponding resolution can be turned into a seamless atlas by applying the stitch map to it. This operation is usually done just after authoring, as a post-processing step. For dynamic textures that change over time, it is also possible to apply the stitch map at rendering time.

2.1. Terminology

Charts The base object manipulated by our algorithm is a chart. As usual, we define a chart as a set of facets and a parameterization of these facets. We number each chart and write chart q as C_q.

Texels We define a regular grid of texels in the parametric domain of each chart. We note $t_{q,i}$ the i-th texel in C_q and $p(t_{q,i})$ the position of $t_{q,i}$ in the parametric domain of C_q. Depending on the magnification mode, the texels are either aligned with integer coordinates (e.g. bilinear magnification) or exactly at the center of each integer cell (e.g. nearest–mode magnification).

We call an **interpolation cell** the area where the same set of texels is required to evaluate the magnification filter (see Figure 6). Conversely, we name **influence area** of a texel the area defined by the interpolation cells involving the texel. For instance, in the case of bilinear interpolation this is a square of 2×2 interpolation cells centered on the texel, but the square border is not included.

We call a **required texel** a texel whose influence area intersects the chart facets, and a **border texel** a required texel whose influence area crosses the chart boundary in the parametric domain. We call an **inner texel** a texel which is inside a facet, i.e. it corresponds to a point on the surface.

Equivalence classes Texels required to share the same value are grouped in equivalence classes, each described by its integer **class id**. An equivalence class captures texel value constraints: All texels in a same class must share the same value. We note $Id(t_{q,i})$ the class id of texel $t_{q,i}$.

3. Enforcing continuity across seams

The input GPP does not define a proper texture atlas. However, it does define a valid parametric domain for each individual facet. Hence, in this first step of our algorithm we consider each facet as an individual chart. We initialize the facet texture coordinates with those given by the GPP, multiplied by a scaling factor $k \in \mathbb{N}, k \geq 1$. The factor k directly controls the texel-grid resolution. Our goal is now to determine which texels across all the (one–facet) charts must be in a same equivalence class, and thus receive a same value.

For the sake of clarity we first present the most common case – bilinear magnification and uniform resolution (Section 3.1) – and later generalize to other magnification modes (Section 3.2) and adaptive resolutions (Section 3.3). Both Section 3.1 and Section 3.3 first describe construction of the equivalence classes (Sections 3.1.1 and 3.3.1), and then describe how to update the constrained texel values after the user paints the surface (Sections 3.1.2 and 3.3.2). This later operation, that we name *stitching*, ensures that all constraints are enforced and that all interpolation seams disappear.

3.1. Standard case

The standard case corresponds to usual texturing situations, using a uniform resolution and bilinear magnification.

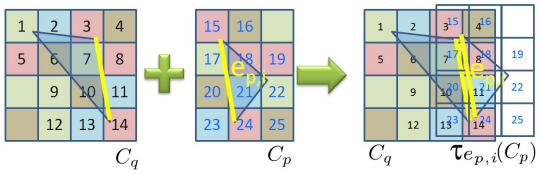


Figure 7: *Continuity conditions across the yellow edge $e_{p,i}$.*

3.1.1. Construction of the equivalence classes

The initialization step creates a chart for each facet. It initializes the texture coordinates to be k times the input grid–preserving texture coordinates. It then determines which texels are required texels. At first, each required texel is allocated its own class id. These will be merged when value constraints are discovered.

Discovering texel equivalence classes To ensure that the resulting texture atlas will be seamless, it is necessary to constrain the values of texels crossing chart boundaries. Intuitively, an inner texel spanning color on several charts requires the corresponding texels – one in each of these charts – to be in the same class.

We detect texel correspondences efficiently by relying on the transition functions. We consider in turn each boundary edge $e_{p,j}$ of a chart C_p. The edge is intersected by the influence areas of several border texels. For each such texel $t_{p,i}$, we find the texel $t_{q,k}$ in the adjacent chart C_q so that $p(t_{q,k}) = \tau_{e_{p,j}}(p(t_{p,i}))$, where $\tau_{e_{p,j}}$ is the transition function of $e_{p,j}$. If $e_{p,j}$ is not a surface boundary we are guaranteed $t_{q,k}$ exists thanks to the grid preserving property. We then merge class ids together, enforcing $Id(t_{q,k}) = Id(t_{p,i})$. For instance in Figure 7 class ids 3 and 15 must be merged, as well as 4 and 16, 7 and 17, etc.

Texel classes can be efficiently generated from these constraints using the *disjoint set* data-structure [CLRS01]. It is important to realize that non trivial relationships will appear, as illustrated Figure 8. These are essentially due to texels having their influence area containing a singularity.

Choosing a value for an equivalence class Each equivalence class tells us which texels must receive the same value. However, the actual value is not decided yet.

Any arbitrary value would provide a continuous result, as long as all texels in the class receive the same one. However our goal is to let the user paint the textures. We thus relate the equivalence class to a position in a facet. This will let us read the value of the constrained texels *after* the user paints on the surface. We name this position the *class value position*. It is expressed in the frame of a facet, to make it invariant to all subsequent operations on charts (coalescing, resizing and packing, see Section 4).

Most equivalence classes contain only one texel, and read their value from its position. When the class contains multiple texels, we choose whenever possible the position of an inner texel. Indeed, such texels can be manipulated directly by a surface painting tool. We thus select the texel having smallest signed distance to a facet in its chart (the distance being negative inside). The class value position is then the position of this texel re-expressed in this facet frame.

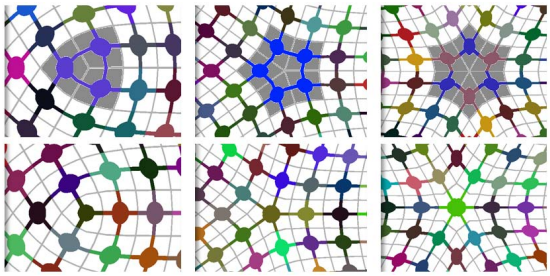


Figure 8: *Color constraints around different singularities. The texel grids projected on the surface have some texels (gray background) constrained to have the same value as other inner texel(s). Singularities lie within an interpolation cell in the top row, and on a texel in the bottom row.*

The obtained class value position will determine where the color of its texels must be read every time the user gives a new input texture.

If a class contains more than two inner texels, only one of them can receive an independent color – implying the user cannot paint the other texels freely. The situation where two or more inner texels appear in a same class occurs near singularities (see Figure 8). As noted in [KNP07] scaling grid preserving texture coordinates by 2 ensures that the image of a regular 2D grid is a quad mesh on the surface (Figure 8, bottom row). In this case the image of the texel grid has singularities that do *not* constrain texels lying on the surface. Thus, simply restricting k to be even guarantees no class contains two or more inner texels, and that all inner texels are independent. Note that these arguments do not hold when adaptive resolution, MIP–mapping or other magnification modes are used (Sections 3.2, 4.4, 3.3), in which case few texels on the surface may be constrained.

3.1.2. Stitching for the standard case

In the standard case, stitching after user input is a very simple operation: We visit each texel in turn, and read its value from its class value position. This position is either the texel itself (hence the texel value does not change), or the texel chosen for the class.

3.2. Other magnification modes

In the previous section, constraints on texel values were defined for bilinear interpolation. This extends naturally to

other magnification modes (nearest, biquadratic, and bicubic) by changing the definition of the texel influence area, which changes the set of required texels. The alignment of the texel grid is set so that each integer cell corresponds to one interpolation cell. As a result, texels are on integer positions with bilinear and bicubic modes, while they are at the center of integer cells in nearest and biquadratic modes. The influence area of each texel is defined as a square centered on the texel with different sizes: 1 for nearest, 2 for bilinear, 3 for biquadratic and 4 for bicubic. The rest of the algorithm is unchanged.

3.3. Adaptive resolution

We now describe how to let the user increase the resolution allocated to each facet independently. Increasing the resolution of a facet makes it larger in texture space, using more texels. On screen finer details appear on the facet. We ensure proper interpolation across seams even in case of resolution mismatch between both sides (see Figure 9). The only requirement is on the magnification function: It must be possible to re–express the value field obtained from texels at a coarse resolution using texels at a finer resolution. All the usual magnification modes have this property, since they can be formulated as polynomial wavelets exactly decomposing coarse resolution functions into finer resolution ones.

3.3.1. Construction of the equivalence classes

Iterative discovery of equivalence classes Each facet is now associated with a power of two resolution multiplier 2^r with $r \in \mathbb{N}$, directly controlled by the user.

Texel value constraints are propagated between different resolutions iteratively: We first consider all facets at the same coarsest resolution, building one–facet charts and equivalence classes as previously described. We then iteratively scale all the charts required to have higher resolution by a factor of two – simply multiplying the parametric coordinates. The set of texels of the chart is cleared and filled with new, higher resolution texels. At each iteration we build equivalence classes for the new texels, as explained in the rest of this section. We proceed until all charts have reached the resolution desired by the user.

Equivalence classes between charts of same resolution Border texels at the interface of charts of same resolution are treated as in the uniform resolution case. Equivalence classes are built, and a value position is chosen for each. It is possible, however, that the class value position no longer aligns with a texel after the iterative process: The texel whose position was chosen at the time of construction may have been deleted by chart upscale. This does not create any particular issue. The class value is still read from the same position (see Section 3.3.2, case $c < p$).

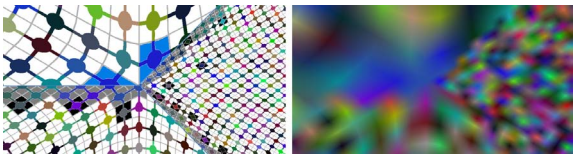


Figure 9: *Color constraints around a singularity with adaptive resolution. Black background texels are used to define texels of a coarser resolution. Grey background texels are interpolated from a coarser resolution. Blue texels are constrained due to the singularity.*

Equivalence classes across resolutions After each scaling, new, finer resolution texels neighboring a chart with lower resolution require a special treatment. The value field along the edge in both charts has to match. The fine interpolation cells thus have to reproduce the coarse ones. This implies a hard constraint fixing the values of all the fine texels in function of the value of the coarse texels.

For such a texel $t_{p,i}$ crossing a half edge $e_{p,j}$, we apply the transition function of $e_{p,j}$ to find the corresponding position $\tau_{e_{p,j}}(p(t_{p,i}))$ in the coarser resolution chart. This position lies within a coarse interpolation cell. The grid preserving property ensures that it is aligned with a regular power of two subdivision of the interpolation cell. The number of subdivisions is given by the resolution ratio between the charts. Thanks to the multi–resolution property of the magnification function (see the introduction of section 3.3), we are guaranteed that we can compute a value that will match both interpolated value fields. We thus record this position as value position for the equivalence class of $t_{p,i}$. We explain in the next section how to compute the value ensuring continuity.

3.3.2. Stitching for adaptive resolution

Stitching is slightly more complex when adaptive resolution is used. By construction each equivalence class contains texels at a same resolution factor 2^c. The position from which to compute the value may however lie in a facet at a different resolution 2^p (same, finer or coarser).

Due to inter–resolution dependencies, we visit texels by increasing resolution factor c (coarse first). For each visited texel, we determine its equivalence class and compute its value from the class value position. This process is illustrated Figure 10. We detail below the three cases that occur:

1. $c = p$ If the resolution is the same, the position corresponds to another texel, its value is picked.
2. $c < p$ The position is at a finer resolution, implying the texel under the position was deleted during the iterative adaptive resolution process (see Section 3.3). No special treatment is required: The texel value is interpolated from finer texels using the selected magnification mode.
3. $c > p$ The position is at a coarser resolution: The finer resolution texel is constrained to match the interpolation field from coarser texels. The value is computed as a

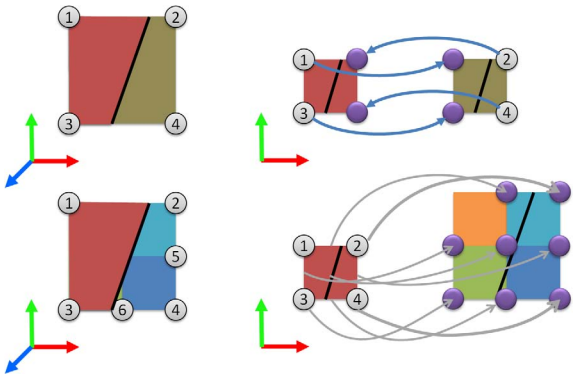


Figure 10: *A seam between charts of different resolutions represented on the surface (left) and in texture space (right). Squares correspond to bilinear interpolation cells. Texel values are computed from coarse (top) to fine resolutions (bottom). Purple texels take their values from numbered texels.*

function of the texel values surrounding the class value position, the magnification mode, and the respective resolutions c and p:

- **Nearest and bilinear (interpolating):** The value is directly computed using the magnification filter on the coarser texels.
- **Biquadratic and bicubic (approximating):** The magnification function is a spline which does not pass through the texel values. We apply subdivision rules $c-p$ times to obtain the value that ensures the same spline is produced by the higher resolution texels.

4. Building the seamless texture atlas

We now have resolved all the constraints between texels of the one–facet charts. This section explains how to produce a texture atlas by coalescing and packing charts, how the final stitch map is computed, and how to support MIP–mapping.

4.1. Coalescing charts

Coalescing the charts is not required to produce an atlas providing continuous interpolation. However, it is strongly recommended for optimizing texture space usage. The algorithm creates larger charts by iteratively coalescing two existing charts.

Given an edge e shared between two charts of the same resolution, we verify whether we can put the charts side by side. In particular, we check that texels that would overlap are compatible. This is done through the transition function of the edge e. Each required texel $t_{q,i}$ in chart C_q mapping on a required texel $t_{p,j}$ in the other chart C_p must be so that $Id(t_{q,i}) = Id(t_{p,j})$. If compatible, one chart is transformed into the frame of the other and both charts are coalesced into a single one. Since we express class value positions in facet frames we do not need to update them.

The simplest global coalescing strategy would be to iteratively consider each edge, coalescing adjacent charts whenever possible. This however results in irregularly shaped charts difficult to pack. We improve the shape of the charts by pre-computing 'firewalls' i.e. edges that cannot be used to coalesce charts. We obtain these firewalls as boundaries of the surface segmentation algorithm proposed by [EDD*95].

4.2. Packing charts

After the creation of larger charts, we pack them into a single texture atlas. We use the Tetris packing algorithm [LPRM02] modified to preserve the grid preserving property of the texture coordinates: We only allow integer translations of the charts. At the end of the process, textures coordinates are normalized by scaling them by $1/(texture\_size)$. Graphics hardware samplers consider texels at the center of the integer cells, thus all texture coordinates must be translated by half a texel size if the bilinear or bicubic modes are used.

4.3. Computing the stitch map

The final step of our algorithm is to compute the stitch map, describing how to obtain the value of constrained texels. The stitch map encodes in a convenient way the equivalence classes and their value positions. It is a $2D$ array of uv coordinates and scale ratios covering the image. For each texel, the stored uv coordinates are its equivalence class value position, re-expressed in the final image rather than relative to a facet frame. The scale ratios are directly the c and p resolutions of, respectively, the texel's chart and the chart enclosing the texel's equivalence class value position.

This lets us apply the computations described Sections 3.1.2 and Section 3.3.2 in each texel. In particular, we can efficiently perform these operations from a GPU fragment program when the texture content is dynamic. With adaptive resolution we have to ensure the order of computations from coarse to fine.

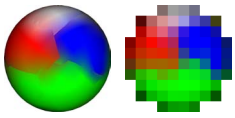
4.4. MIP–mapping support

To define a texture atlas which is seamless through MIP–mapping, it is sufficient to ensure continuity at each level of the MIP–map pyramid (Figure 11). The main difficulty is the dual alignment of successive levels of the pyramid: Interpolation cells do not align across levels since finer resolution texels lie in-between coarser resolution ones. We are nevertheless able to define a texture atlas providing continuous interpolation across several MIP–map levels – typically the three or four finest levels, depending on the initial texture resolution (see Figure 11).

We first produce the coarsest level of detail that must be

Figure 11: *Our approach ensures perfect continuity across the finer MIP–mapping levels.*

seamless using our algorithm, but we skip the texture coordinate normalization. Then, increasingly finer levels are produced one by one. Each new level is obtained by multiplying the previous level texture coordinates by 2. We then consider that all facets belong to a same unique chart and determine the texel value constraints as usual (Section 3.1). We finally normalize the texture coordinates. This achieves perfect continuity on these MIP-map levels. Coarser levels are of course MIP-mapped but perfect continuity is no longer guaranteed. It is important to realize though that coarse levels correspond to far away viewpoints: The object is small on screen and MIP-map levels are strongly filtered – color differences across charts are difficult to perceive. The inset shows an example of the first level where continuity breaks, in a extreme setting where each chart has a different color at the finest resolution. At this level of detail the object covers approximatively a 10×10 pixels area on screen.



Note that this approach is limited to uniform resolution. Indeed, adaptive resolution is achieved via subdivision of interpolation cells which is incompatible with the dual alignment of MIP–mapping. The specialized texture access of [PCK04] would solve this issue by enforcing proper alignment of texels across levels.

5. Results

Construction time Our method is a pre–processing step that does not affect rendering performance. In our implementation, only border texels are stored (in sparse matrices). We produce our result 1.7 seconds for the frog model (7K triangles, final texture is 1024×280) with adaptive resolution and 48 seconds for the statue model (100k triangles, final texture size 256×512). The hand model (25K triangles, final texture is 512×512) has many different resolutions that need to be inspected and takes 8.3 seconds to be solved. In all our examples, the generation of the GPP always takes less than one minute.

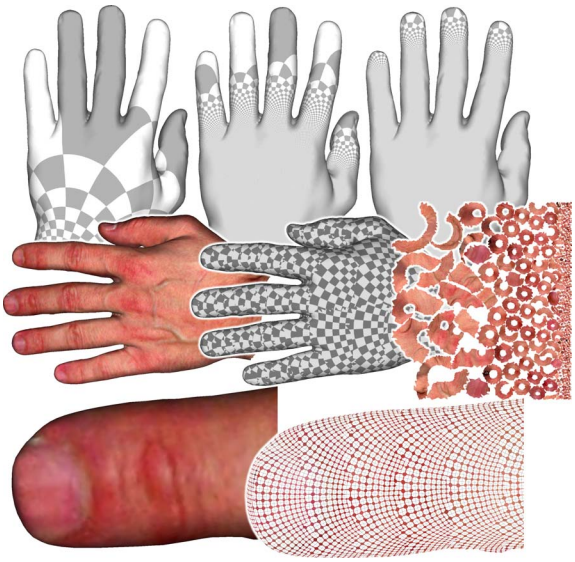


Figure 12: *Scalability of our per facet resolution : The upper row is the original LSCM parameterization at scale 1, 100 and 10000. The middle row shows a* 512×512 *texture applied with the original map scaling countered by per facet resolution. Bottom row is a close-up view of a finger, with explicit rendering of the texel grid (Right).*

Figure 13: *Adaptive resolution can provide more resolution to important features like the eye of the frog.*

Packing efficiency The filling ratio – sum of 2D facet area over global texture area – of our texture atlas (hand 41%, adaptive resolution frog 47% and the statue 49%) competes with automatically generated texture atlas using LSCM, variational shape approximation, and the Tetris packer (hand 42%, adaptive resolution frog 44% and the statue 49%).

Applications The texture atlas can be used during authoring and rendering as any other texture atlases, but also brings interesting features:

- The local texture resolution of a facet can be increased on–the–fly within a 3*D* paint system. This requires constructing a new texture atlas from the same GPP, however the content of most facets is directly copied from the previous atlas. For the facets changing resolution, new higher-resolution texels are computed from the existing lower resolution texels.

- Complex shaders often perform computations in a local neighborhood around a texel (e.g. filtering, computing normals, gradients, etc.). As long as the extent of this neighborhood is bound and known, setting the corresponding texel influence area guarantees that all accessed samples will be properly defined in the texture atlas.
- Besides color and normal mapping, our atlases can be used to displace vertices from a vertex shader texture lookup without introducing cracks.

Discussion Our approach provides a continuous result through a single standard texture lookup (contrary to [BD02, PCK04, CHCH06, LD07, YKH10, GP09]), it does not constrain charts to be axis aligned in uv space (contrary to [PCK04, CHCH06]), which can lead to excessive distortion requiring to modify the input mesh [CHCH06]. In addition we support adaptive resolution and several magnification modes. The drawbacks of our approach are that it computes a new set of uv coordinates rather than fixing existing ones (contrary to [GP09]), that it inputs a GPP which computation is still an active area of research, and that MIP-mapping support is limited. On this last point, please note that at worst MIP-mapping gives results equivalent to standard MIP-mapping of texture atlases. Finally, some approaches enable other effects such as geometric simplification [PCK04]. We did not explore such capabilities.

6. Conclusion

Our approach generates standard texture atlases making seams totally invisible, even through MIP-mapping and different magnification modes. Our textures are authored as any other texture and only require a fast post-processing step before being ready for rendering. Preserving the hardware texture access makes our technique affordable on low-end hardware, and avoids additional burden on complex shaders.

Since our method essentially defines similar neighborhoods – in terms of values – in the texture and along the surface, we believe it will also find applications for on-surface image processing methods such as by–example texture synthesis, smoothing, or simulation of weathering effects. We hope our approach will support further research in grid–preserving parameterizations.

Acknowledgements

We thank Bruno Vallet, Chris Wojtan and Greg Turk for early discussions on this project. This work received support from the Agence Nationale de la Recherche (SIMILAR-CITIES ANR-2008-COORD-021-01) and the European Research Council (GOODSHAPE FP7-ERC-StG-205693).

References

[BD02] BENSON D., DAVIS J.: Octree textures. *ACM Trans. on Graphics 21*, 3 (2002), 785–790. 2, 8

[BL08] BURLEY B., LACEWELL D.: Ptex: Per-face texture mapping for production rendering. In *Eurographics Symp. on Rendering 2008* (2008), pp. 1155–1164. 2

[BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quadrangulation. *ACM Trans. on Graphics 28*, 3 (2009), 1–10. 2

[CH04] CARR N. A., HART J. C.: Painting detail. In *SIGGRAPH '04* (2004), ACM, pp. 845–852. 2

[CHCH06] CARR N. A., HOBEROCK J., CRANE K., HART J. C.: Rectangular multi-chart geometry images. In *SGP '06: EG/ACM symp. on Geometry processing* (2006), Eurographics Association, pp. 181–190. 2, 8

[CLRS01] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to Algorithms*, 2nd revised edition ed. The MIT Press, September 2001. 4

[EDD*95] ECK M., DEROSE T., DUCHAMP T., HOPPE H., LOUNSBERY M., STUETZLE W.: Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95* (1995), ACM, pp. 173–182. 6

[FH05] FLOATER M. S., HORMANN K.: Surface parameterization: a tutorial and survey. In *Advances in multiresolution for geometric modelling*, Dodgson N. A., Floater M. S., Sabin M. A., (Eds.). Springer Verlag, 2005, pp. 157–186. 2

[GP09] GONZÁLEZ F., PATOW G.: Continuity mapping for multi-chart textures. In *SIGGRAPH Asia '09* (New York, NY, USA, 2009), ACM, pp. 1–8. 2, 8

[KNP07] KALBERER F., NIESER M., POLTHIER K.: Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3 (September 2007), 375–384. 2, 4

[LD07] LEFEBVRE S., DACHSBACHER C.: Tiletrees. In *Proceedings of I3D* (2007), ACM, pp. 25–31. 2, 8

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *SIGGRAPH '06 25*, 3 (2006), 541–548. 2, 3

[LPRM02] LÉVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH '02* (New York, NY, USA, 2002), ACM, pp. 362–371. 6

[PCK04] PURNOMO B., COHEN J. D., KUMAR S.: Seamless texture atlases. In *SGP '04: EG/ACM symp. on Geometry processing* (New York, NY, USA, 2004), ACM, pp. 65–74. 2, 7, 8

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Trans. Graph. 25*, 4 (2006), 1460–1485. 2

[RVAL09] RAY N., VALLET B., ALONSO L., LEVY B.: Geometry-aware direction field processing. *ACM Trans. Graph. 29*, 1 (2009), 1–11. 2

[SH02] SHEFFER A., HART J. C.: Seamster: inconspicuous low-distortion texture seam layout. In *VIS '02* (2002), IEEE Computer Society, pp. 291–298. 2

[TACSD06] TONG Y., ALLIEZ P., COHEN-STEINER D., DESBRUN M.: Designing quadrangulations with discrete harmonic forms. In *SGP '06* (Aire-la-Ville, Switzerland, Switzerland, 2006), Eurographics Association, pp. 201–210. 2

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 853–860. 2

[YKH10] YUKSEL C., KEYSER J., HOUSE D. H.: Mesh colors. *ACM Trans. Graph. 29*, 2 (2010), 1–11. 2, 8