Ibero-American Symposium on Computer Graphics - SIACG (2006)
P. Brunet, N. Correia, and G. Baranoski (Editors)

Adaptive trimming of cubic triangular Bézier patches

Á.L. García, J. Ruiz de Miras, F.R. Feito[†]

Department of Computer Science
University of Jaén
Campus Las Lagunillas
23071 Jaén, SPAIN

Abstract

We present a method to handle cubic trimmed triangular Bézier patches. This scheme makes use of levels of detail and surface subdivision to achieve a fast and flexible hierarchical data structure that is specially useful to compute surface intersections in a robust and efficient way. The accuracy of the results can be adjusted by adding or subtracting elements to the levels of detail hierarchy, and it is also easy to obtain a decomposition of a trimmed patch into single triangular Bézier patches.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

1. Introduction

The surface trimming problem is one of the basic issues in the field of Solid Modeling. It can be stated as: given a surface patch that intersects with one or more geometric elements (polygons, surface patches, etc.), determine the intersection curves on the patch, divide the patch into two regions, and remove one of them according to a trimming criterion.

It is necessary to develop efficient and robust algorithms to solve this problem in order to increase the capabilities of CAD/CAM systems. Many strategies have been studied to solve this problem [KGMM97, Key00, BKZ01, LLS01, FHHS04], but none of them can be considered the best solution for all kinds of surfaces and modelling applications.

The first step of a trimming algorithm is the computation of the trimming curves on the surface. Typically, these curves are obtained by intersecting the surface with another geometric element (curve, plane, surface, object). There are lots of papers concerning this problem [AMY96, CKKK97, GK97, LJCW04]; some of them use analytical methods to obtain an exact solution, while others turn to numerical methods to compute a more or less precise approximation using different strategies. Once the intersection curves are known, the trimmed surface is built by merging them with the original boundaries and cutting off those regions of the surface that are outside the new boundaries.

Our approach uses surface subdivision as Aziz et al. [AB90], but their work deals only with computing the intersection of patches; to get appropriate trimming curves it is necessary to detect connected components, loops and self-intersections, and this processing is not considered by them.

Another interesting work on trimmed surfaces is the one by Stürzlinger [Stü98], but that paper is centered on ray-tracing algorithms; it does not compute the intersection curve, nor obtains the trimmed surface as a geometric entity that can be used as a primitive for modeling.

The work of Krishnan [Kri97] and Keyser [Key00] is very interesting, dealing with trimming of tensor product Bézier patches as an intermediate step for B-rep generation of CSG models. Nevertheless, to our knowledge there is no paper discussing specifically the trimming of triangular Bézier patches.

We present a somewhat different method to represent and trim triangular Bézier patches (TBPs) [Far86]. The main ad-

[†] e-mail: {algarcia, demiras, ffeito}@ujaen.es

vantage of these patches is their simplicity, that has made them be used for mesh interpolation (see [HB03], for example); scattered data fitting is another research field where TBPs have proven to be useful [Zei02]. Particularly interesting is the work from Nürnberger et al. [NKZ03], where this kind of patches is used to construct a Lagrange interpolation of a detailed triangle mesh; the initial mesh is approximated by a quite small number of patches, making easier the storage, transmission and visualization of the surface represented by the initial mesh. TBPs are also used in hardware based visualization, both by specific hardware implementation [ATI01] and use of programmable GPUs [BS05], and are the chosen surface patches in the mathematical model to represent free-form solids called the extended simplicial chain (ESC) model [GRF03, RF99].

This work presents the next step in the development of the ESC model since, as it is said before, the capability of trimming surfaces is essential in any geometric modeling system. To achieve this purpose, we have developed an adaptive hierarchical data structure, using subdivision techniques to compute intersection curves on a patch and then trim the patch portions that lie outside the desired result.

In the following sections, we present a brief summary of the basics of TBPs, a description of the data structure, and how we use it to compute the patch-patch intersection and patch trimming operation. Finally, we will comment on the results we have obtained and the future work.

2. Triangular Bézier patches

Triangular Bézier Patches (TBPs) are cited [Far86, BFK84] as the first extension of the Bézier curves to surfaces, as a more natural generalization than are tensor product patches. As with polygons, it is easier to model a surface with triangular elements than with square ones. Even the well-known GPU manufacturer ATI has developed the TRUFORM technology that uses this kind of patches (renamed as "Curved PN-Triangles") as a simple and efficient way to display smoother and more natural images [ATI01, VPBM01].

TBPs are defined by a triangular control point net with $\frac{1}{2}\cdot(n+1)\cdot(n+2)$ points (n being the degree of the patch). Each control point is named b_{ijk}, with $i \geq 0,\ j \geq 0,\ k \geq 0$, and $i+j+k=n$.

The parametric domain is triangular, having three parameters, u, v and w, that take value from the continuous interval [0, 1]; the sum u+v+w always equals 1.0.

All this work has been carried out using cubic TBPs as reference surface patches. Figure 1 shows the control net from a cubic patch, and its parametric domain.

The surface is calculated using the Bernstein polynomials; for this kind of patch, these polynomials are given by:

$$B^n_{i,j,k}(u,v,w) = \frac{n!}{i!\cdot j!\cdot k!}\cdot u^i\cdot v^j\cdot w^k;$$

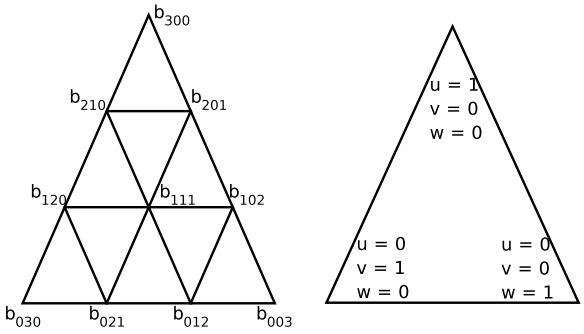

Figure 1: *Control point net of a cubic TBP and parametric domain*

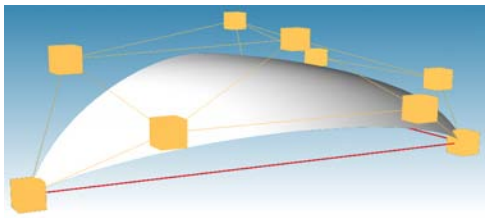

Figure 2: *cubic triangular Bézier patch*

Calling λ the 3-tuple (i,j,k) and τ the 3-tuple (u,v,w), we obtain the expression for a n-degree TBP as:

$$b^n(\tau) = \sum_{|\lambda|=n} b_\lambda \cdot B^n_\lambda(\tau);$$

$|\lambda| = n$ stands for all the 3-tuples (i,j,k) which satisfy that $i+j+k=n$. Figure 2 shows a cubic TBP with its associated control point net and its base triangle (drawn in red).

Two interesting properties can be inferred from the formulae above [PBP02]. One is every TBP is included in the convex hull of its control points, and interpolates b_{n00}, b_{0n0} *and* b_{00n}; the second one is that the edges of a patch are Bézier curves whose control points are those on the edges of the control point mesh. These properties will be useful when using TBPs in the next sections.

Subdivision of TBPs is another important feature that is intensively used in our work. Seidel [Sei89] describes how to use the de Casteljau algorithm to compute the control point net of a subpatch for a given parametric subdomain in a simple and fast way. Another interesting paper on subdivision of TBPs is the one by Li et al. [LKL*02], where a method to compute in advance how many times a TBP should be subdivided is presented.

3. The levels of detail hierarchy

To develop the intersection and trimming algorithms for TBPs, we decided to construct a hierarchical data structure

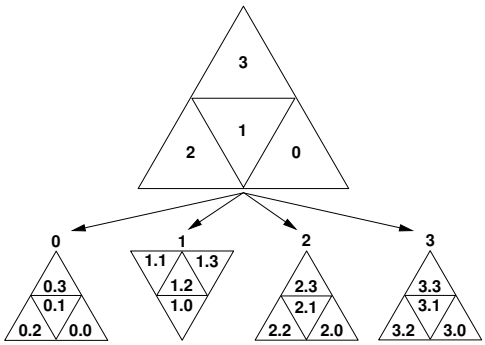

Figure 3: *Parent node and child nodes.* Lod = *1*

based on the previous work with these patches in the ESC model [GRF03, GRF04]. Our algorithm to test the inclusion of a point in a free-form solid represented with the ESC model uses subdivision of TBPs to compute the intersection of a ray with a patch, so we decided to store the subdivision results and use them for inclusion testing, computing intersection curves and patch trimming.

Basically, we consider a tree-like structure, with n levels and m nodes stored in each level. Each node at level 1 consist of a subpatch obtained from the original TBP using subdivision, and the child nodes store subpatches obtained from the subpatch stored in its respective parent node. Figure 3 schematically shows this process. Following Vlachos et al. [VPBM01] notation, we will consider the subdivision level of detail (*lod*) as the number of evaluation points on one edge of the domain minus two.

From this point, we will call *lod* the subdivision level of detail, and *level* each level of the structure.

Figure 4 shows a TBP and the triangles obtained by joining the vertices of the subpatches stored in our structure at levels 1, 2 and 3 using *lod*=1. As it can be seen, the amount of subpatches grows very quickly, making unnecesary the use of many levels or high values of *lod*. To be precise, the number of subpatches obtained with n levels and *lod*=l, noted n_{SP} is:

$$n_{SP} = (l+1)^{2n};$$

For example, the leaf nodes of a tree with 3 levels and *lod*=1 store 64 subpatches.

The data kept in this tree can be used in several frequently used tasks in solid modeling:

- Visualization: the base triangles of the subpatches gives us an approximation of the patch. Therefore, drawing the base triangles at a desired level of the tree allows us to get a more or less acceptable (depending on the chosen level) view of the patch. Choosing an upper level of the tree will produce a less-detailed view that can be drawn

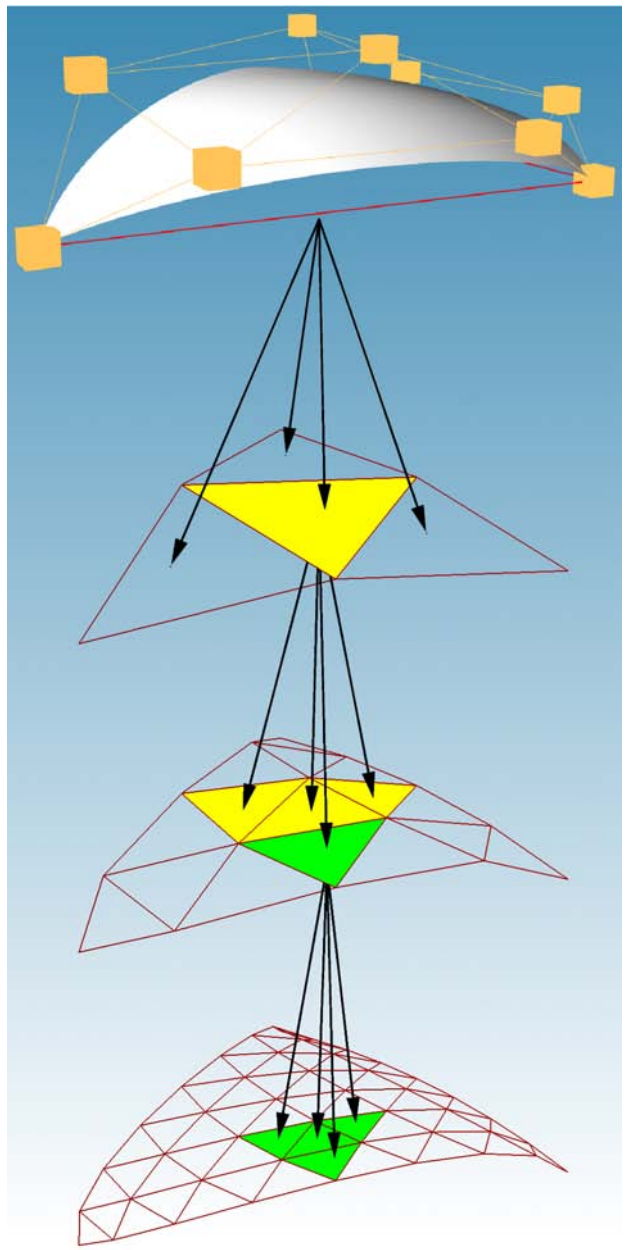

Figure 4: *TBP and subdivisions stored at levels 1, 2 and 3 (*lod = *1)*

faster (useful when dealing with many TBPs or drawing models that are placed far away from the camera).
- Polygonization: a polyhedral approximation of a TBP can be obtained by considering the base triangles of the subpatches at one level of the tree. The lower the level, the more precise the result.
- Point in solid test: our algorithm to test the inclusion of a point in a solid computes the intersections of a ray with the surface of the solid. The intersections with a TBP are computed in an iterative way, dealing first with the subpatches at the upper levels of the tree, discarding those branches whose parents are not intersected by the ray, and refining the found points of intersection through the lower levels.
- Surface intersection and trimming: this will be explained in the next sections

Note that the tree stores not only the triangle data, but also all the subpatch related information (coordinates of the parametric subdomain, control points, bounding box, etc.). This allows us to precompute the data when loading the model, therefore speeding up significantly all subsequent operations without loss of accuracy.

4. Patch-Patch intersection

As stated before, the first step of a surface trimming algorithm is the computation of the trimming curves on the surface. Therefore, we will describe here the steps we have followed to determine the intersection of two TBPs.

Algorithm 4.1: TBP_INT($P1, P2, max\_level$)

comment: Computes the intersection of TBPs P1 and P2

comment: max_level is the level of the trees to be used

if $P1.BBox$ *intersects* $P2.BBox$
then
- **if** $(P1.level, P2.level = max\_level)$
  - **then** $i \leftarrow$ TR_INT$(P1.base, P2.base)$; ADD_SEGMENTS(i)
- **else**
  - **if** $(P1.level, P2.level \neq max\_level)$
    - **then** $Pm \leftarrow$ *TBP with smallest BBox*; $PM \leftarrow$ *TBP with biggest BBox*; **for each** $SP \in PM.children$ **do** TBP_INT(SP, Pm)
  - **else**
    - **if** $P1.level = max\_level$
      - **then** **for each** $SP \in P2.children$ **do** TBP_INT$(P1, SP)$
      - **else** **for each** $SP \in P1.children$ **do** TBP_INT$(SP, P2)$

else return $(No\ intersection)$

Basically, what we do is propagate the intersection computation from the upper levels of the tree to the lower levels, checking at each level if it is necessary to continue the process by testing the intersection of the bounding boxes of the subpatches. The intersection is computed at the leaf nodes by intersecting the base triangles of the subpatches and adding the resulting segments to a data structure that keeps the segments order and detects possible loops and self-intersections. Once all the segments have been added, a post-process is carried out on the segments to group them in curve components that can be used as input by the trimming algorithm.

Algorithm 4.1 describes with more detail the intersection computation process. The algorithm TR_INT consist of a classic triangle-triangle intersection algorithm [Möl97], and Algorithm 4.2 describes ADD_SEGMENTS.

Algorithm 4.2 describes how a segment from the TBP_INT algorithm is processed. The *Reorder segments* instruction is called only if the new segment connects two separated sets of already linked segments in the structure.

Algorithm 4.2: ADD_SEGMENTS(i)

comment: Inserts segment i into the segments structure

if *no segments stored*
then *Add segment i to segments*

else
- **if** *i intersects any* $s \in segments$
  - **then** *Delete s from segments*; $p \leftarrow$ INTERSECTION(i, s); $\{i1, i2\} \leftarrow$ DIVIDE(i, p); $\{s1, s2\} \leftarrow$ DIVIDE(s, p); ADD_SEGMENTS$(i1, i2, s1, s2)$
- **else**
  - **if** $\exists$ *s such that links with i*
    - **then**
      - **if** *there are potential loops* **then** *Save i separatedly*
      - **else** *Add segment i linked with s*; *Reorder segments*

A study of the special cases that may appear when adding a new segment to the structure shows two different situations:

- The new segment forms a closed loop with a (sub)set of previously added segments.
- The new segment forms a Y-like bifurcation with other two segments (see corners of the trim in figure 7).

Both cases are treated by storing the new segment separately in a *potential_loop_segments* structure; the segments from this structure will be used later to obtain the connected components.

Once all the intersection segments have been computed, the algorithm BUILD_COMPONENTS is run on them to obtain the components of the result. See Algorithm 4.3.

Algorithm 4.3: BUILD_COMPONENTS(*segments*)

comment: Creates connected components from *segments*

$c \leftarrow$ NEW_COMPONENT(*segments.first*)

while *segments is not empty*
do
- $s \leftarrow$ *segments.next*
- **if** *s links with c*
  - **then** *Add s to c*
  - **else**
    - *Save component c*
    - $c \leftarrow$ NEW_COMPONENT(s)

while *potential_loop_segments is not empty*
do
- $s \leftarrow$ *potential_loop_segments.next*
- **for each** $c \in$ *components*
- **do**
  - **if** *s links with c*
  - **then**
    - *Add a copy of s to c*
    - **if** *a loop has appeared*
    - **then**
      - *Cut the new loop*
      - *Save it as closed component*

Check components for connectivity and loops

The last step of Algorithm 4.3 consist of checking if there exist any pair of components that can be linked together. If so, they are joined as a new component, and if it is closed, then it is saved. This process is repeated until there is no more components that can be linked, and then the remaining components are saved as open components together with the closed ones saved before.

Note that the BUILD_COMPONENTS algorithm takes advantage from the fact that the TBPs used are cubic, and therefore it can be expected that the intersection curves will never be specially complex. This algorithm has not been tested with other kind of surfaces so far.

As the intersection is computed through the levels of the tree, many calculations will be avoided, as there will be no intersection between the bounding boxes of many pairs of nodes. Moreover, the choice of the patch with the biggest bounding box for subdivision maximizes the number of discarded subpatches. Figure 5 shows two TBPs and their computed intersection (in white in the upper picture, and in blue in the others), drawing only the base triangles of the chosen subpatches at each level. As it can be seen, the number of discarded subpatches is significant. The pictures have been taken using a tree with 4 levels and *lod*=1.

Figure 6 shows two TBPs intersecting at two different regions. The intersection curve therefore has two separated components (drawn in white). The figures have been obtained using a tree with 2 levels and *lod*=3. Our algorithm correctly detects this situation.

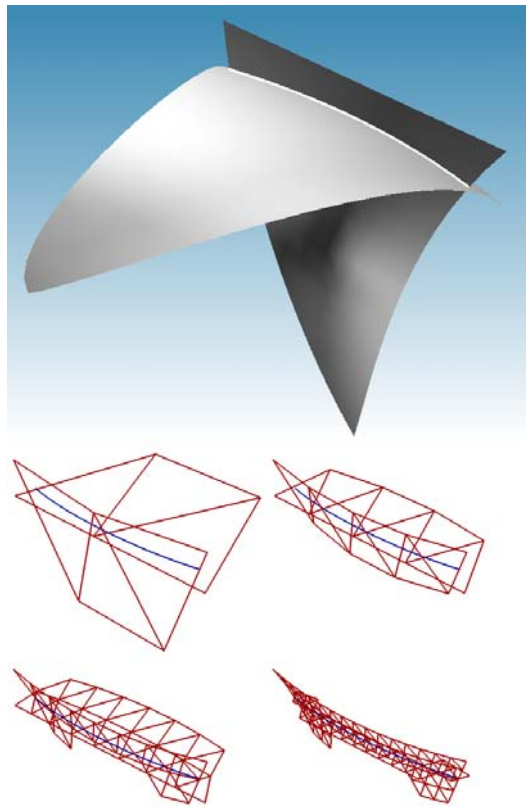

Figure 5: *Intersection curve of two TBPs and subdivisions at levels 1, 2, 3 and 4 used to compute it (*lod=*1)*

5. Patch trimming

The patch-patch intersection algorithm presented before is a basic element in the patch trimming algorithm that is going to be presented now. However, it is just the first step of a more complex process that also implies things like tesselating triangles or checking point inclusion with respect to a solid.

It is necessary to establish a binary partition of the space in order to trim the patch, therefore we have considered the use of a free-form solid bounded by TBPs. The region trimmed from the patch in this section is the one placed inside of the solid.

Roughly speaking, the trimming process has the following phases (see figure 7):

a) Compute the intersection curves between the TBP to be trimmed and each TBP that bounds the free-form solid.

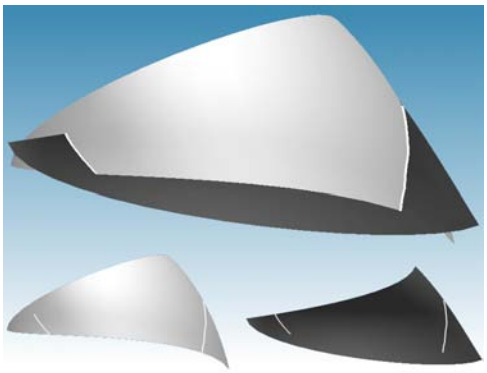

Figure 6: *Up: two TBPs with a two component intersection (in white). Down: the components of the intersection are drawn on the individual patches*

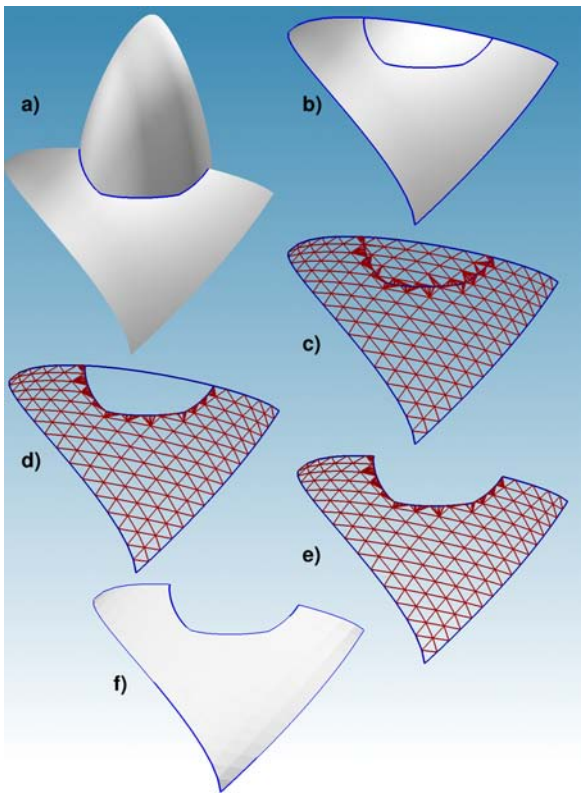

Figure 7: *The phases of the patch trimming algorithm*

This includes computing the connected components for each patch-patch intersection. Figure 7.a

b) Combine the components of all the intersections that have been found. In this phase, all the intersection points among components are computed, and the components are cut up into pieces such that no piece contains intersection points different from its starting and ending points. The original boundaries of the TBP are also included in this process, and as a result of this, the set of boundaries for both inside and outside regions of the patch are obtained. Figure 7.b
c) Tesselate the parametric domain of all the subpatches stored at the leaf nodes of the levels of detail hierarchy that are intersected by the pieces computed in phase 2.. As a result of this, all the elements stored at the hierarchy are completely inside or completely outside of the trimming solid. Figure 7.c
d) Delete those subpatches stored at the leaf nodes that are inside of the solid, and propagate the deletion process up in the hierarchy, so that the nodes whose children have been completely deleted are deleted too. To determine which subpatches are inside of the solid, a point in solid test has to be used [GRF04]. Figure 7.d
e) Erase the boundaries of the inside region of the patch. Figure 7.e
f) Establish the new boundaries for the trimmed TBP. Figure 7.f

Pictures at figure 7 have been taken using a tree with 2 levels and *lod*=3.

The first step of the process in phase b) is carried out by computing the intersection points among the intersection curves and the original patch boundaries. As these curves are piecewise linear, the intersections can be computed easily. These intersection points are used to cut up into pieces all the curves, obtaining as a result simple and continuous pieces that bound the inside and the outside region of the TBP.

To tesselate the parametric domain of the subpatches intersected by the trimming curves in phase c), we compute the barycentric coordinates of the intersection points among the base triangle of each subpatch and the trimming curves, and use these points as tesselation vertices together with the vertices of the subdomain. The tesselation algorithm does not differ from any of the classic ones that can be found in the references [O'R98, Las96].

The deletion process of phase d) makes intensive use of the hierarchy. First, the process is started from the root node. At any level, each node tests if it is completely inside or completely outside of the trimming solid by testing the inclusion of its bounding box (except the leaf nodes; these nodes use for testing the barycenter of the base triangle of the subpatches). If the box is completely inside, the whole branch that grows from that node is deleted, and the state of the node changes to "to be deleted". If some children of a node

are deleted, that node changes its state to "trimmed". This behaviour is partially inspired by the mechanism used with octrees in polihedral solid modeling, and allows a simple and fast trimming through all the hierarchy. Algorithm 5.1 describes with more detail this process.

Finally, to erase the boundaries of the inside region of the patch in phase e), we choose a point on each trimming curve and test its inclusion in the trimming solid. If the point is inside, the trimming curve is discarded, otherwise it is accepted as boundary of the final trimmed patch in phase f).

Algorithm 5.1: TRIM(p, obj, max_level)

```
comment: Trim the subpatches of patch p that are inside obj
comment: max_level is the level of the trees to be used
if p.level ≠ max_level
then { inclusion_value ← INCLUSION(p.BBox, obj)
       if inclusion_value = INSIDE
       then { for each sp ∈ p.children
              do DELETE_SUBPATCH_BRANCH(sp)
            p.state ← to be deleted
     { if inclusion_value ≠ OUTSIDE
       else { then { for each sp ∈ patch.children
                     do { TRIM(sp, obj)
                          if sp.state = to be deleted
                          then { delete sp
                                 p.state ← trimmed
                   if p.children.amount = 0
                     then p.state ← to be deleted
            }
else { inclusion_value ← INCLUSION(p.tr.barycenter, obj)
       if inclusion_value = INSIDE
         then patch.state ← to be deleted
```

6. Results

Images obtained in our tests have been shown all over this paper. We have tested the algorithm with different levels in the tree and several values of *lod*, proving to be a robust and efficient trimming algorithm.

The use of this kind of data structure allows us to select the detail at which we want the intersection and trimming operations be done, making it useful for applications like fast prototyping and visualization of free-form solids, and allowing us to adapt the accuracy of the results to the desired modeling operation.

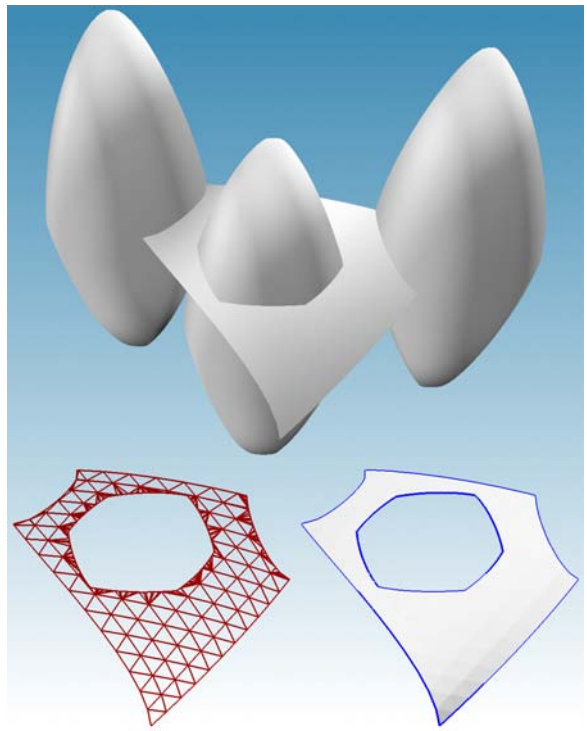

Figure 8: *A complex trimmed TBP. Up: initial configuration. Down: the trimmed patch*

Figure 8 shows a more complex configuration of patch trimming, and figure 9 shows another example where there exist intersections among the trimming curves and the original boundaries of the patch. Figure 10 shows the steps of the trimming process for a "happy patch" sample, where solids with concavities and convexities are used to perform four trims on the same TBP. In all these cases, our algorithm has produced correct results. The three figures have been created using a tree containing 2 levels and *lod*=3 for each TBP.

Table 1 shows computation information from figures 8, 9 and 10 as obtained on a Pentium IV 2.4 GHz PC. It shows the number of TBPs involved in the trimming operation, the number of calls to the triangle-triangle intersection algorithm, the number of segments bounding the resulting patch and the time spent in the whole trimming operations; time spent comprises all the phases indicated in section 5. Figure 8 was the faster one to be computed, even when figure 9 uses less TBPs; this is due to the fact that there is no intersection between the trimming curves in figure 8.

Although the number of triangle-triangle intersection tests may seem significant, the use of the hierarchy has reduced it significantly; figure 10, for example, implies more than 30 million tests and more than 70 seconds of computation time if computed without the benefits of the detail tree.

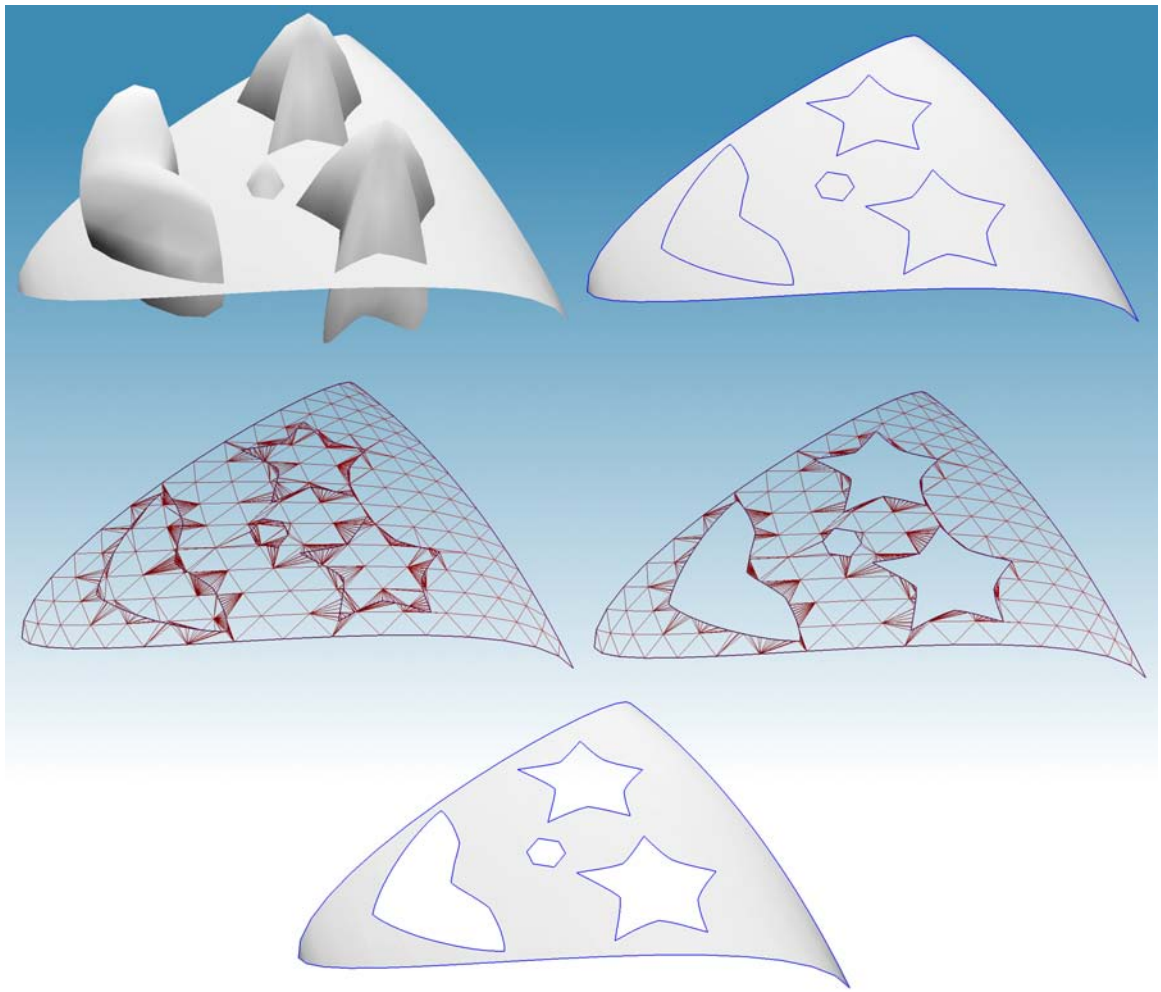

Figure 10: *"Happy patch" example. Upper left: initial configuration. Upper right: boundaries of the trimmed patch. Mid-left: result of the tesselation phase. Mid-right: trimmed tesselation. Down: final result*

Obviously, the time to compute a trimmed patch depends directly on the number of levels of the tree and the subdivision *lod* used at each one, so the user can choose between getting a more accurate result (by adding more levels and/or increasing the value of *lod*) or a faster approach (using less levels and/or a smaller value of *lod*).

7. Conclusions

A hierarchical data structure to handle triangular Bézier patches (TBPs) have been presented, together with algorithms to compute both the patch-patch intersection curves and patch trimming operation using a free-form solid bounded by TBPs. The algorithms have been explained in detail and several pictures of the results obtained with different resolutions have been shown, proving the effectiveness and robustness of the algorithms.

These algorithms are based on the propagation of the operations through the nodes of the hierarchy, avoiding the evaluation of many operations at the leaf nodes by means of checking the bounding boxes of their ancestor nodes.

This work is related with previous works by Aziz et al. [AB90] and Stürzlinger [Stü98], but including new features, as it provides algorithms to obtain valid trimming curves from the result of patch-patch intersection, and an effective way to compute the trimmed surface patch defined by those curves.

The posibility of choosing the resolution parameters both to create the data structure and to run the algorithms gives us flexibility in the modeling operations, making it easy to han-

Figure	Intersected TBPs	Triangle-Triangle intersections	Segments	Time
8	31	502624	303	0.44 sec.
9	13	821392	344	0.67 sec.
10	43	1305920	836	1.20 sec.

Table 1: *Computation information from figures 8, 9 and 10 (using a tree with 2 levels and* lod=*3) for each TBP*

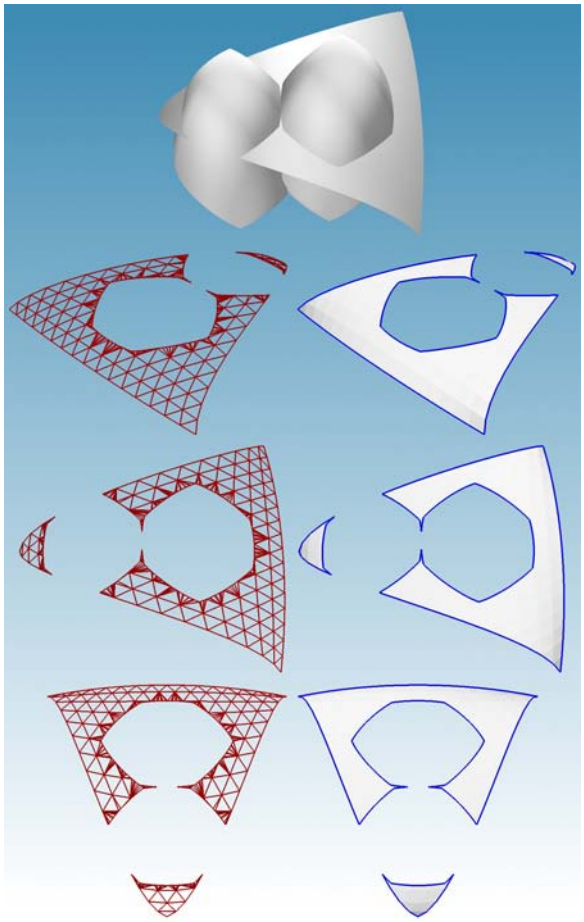

Figure 9: *A trimmed TBP whith complex trimming curves. Top: initial configuration. Down: several views of the result*

dle TBPs with different accuracy levels, although keeping coherence and topology of the trimmed patches.

8. Acknowledgements

This work has been partially supported by the Spanish Ministry of Education and Science and the European Union (via ERDF funds) through research project TIN2004-06326-C03-03

References

[AB90] AZIZ N. M., BATA R.: Bezier surface/surface intersection. *Computer Graphics and Applications 10*, 1 (January 1990), 50–58.

[AMY96] ABDEL-MALEK K., YEH H.-J.: Determining intersection curves between surfaces of two solids. *Computer Aided Design 28*, 6–7 (June–July 1996), 539–549.

[ATI01] ATI TECHNOLOGIES, INC.: Truform white paper, 2001.

[BFK84] BÖHM W., FARIN G., KAHMANN J.: A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design 1* (1984), 1–60.

[BKZ01] BIERMANN H., KRISTJANSSON D., ZORIN D.: Approximate boolean operations on free-form solids. In *ACM Siggraph* (Los Angeles, USA, 2001), ACM, pp. 185–194.

[BS05] BOUBEKEUR T., SCHLICK C.: Generic mesh refinement on gpu. In *Graphics Hardware 2005* (Los Angeles, USA, 2005), SIGGRAPH/Eurographics, pp. 99–104.

[CKKK97] CHO N., KIM N., KIM Y., KANG S.-H.: An evolutionary method for general surface–surface intersection problems. *Computers & Industrial Engineering 33*, 3–4 (December 1997), 573–576.

[Far86] FARIN G.: Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design 3* (1986), 83–127.

[FHHS04] FAROUKI R., HAN C., HAS J., SEDERBERG T.: Topologically consistent trimmed surface approximations based on triangular patches. *Computer Aided Geometric Design 21*, 5 (May 2004), 459–478.

[GK97] GRANDINE T., KLEIN F.: A new approach to the surface intersection problem. *Computer Aided Geometric Design 14*, 2 (1997), 111–134.

[GRF03] GARCÍA Á., RUIZ J., FEITO F.: Free-form solid modelling based on extended simplicial chains using triangular bézier patches. *Computers & Graphics 27*, 1 (February 2003), 27–39.

[GRF04] GARCÍA Á., RUIZ J., FEITO F.: Point in solid test for free-form solids defined with triangular Bézier patches. *The Visual Computer 20*, 5 (July 2004), 298–313.

[HB03] HAHMANN S., BONNEAU G.: Polynomial surfaces interpolating arbitrary triangulations. *Transactions on Visualization and Computer Graphics 9*, 1 (2003), 99–109.

[Key00] KEYSER J.: *Exact Boundary Evaluation for Curved Solids*. PhD thesis, University of North Carolina at Chapel Hill, 2000.

[KGMM97] KRISHNAN S., GOPI M., MANOCHA D., MINE M.: Interactive boundary computation of boolean combinations of sculptured solids. *Computer Graphics Forum 16*, 3 (1997), 67–78. Proceedings of Eurographics'97.

[Kri97] KRISHNAN S.: *Efficient and accurate boundary evaluation algorithms for boolean combinations of sculptured solids*. PhD thesis, University of North Carolina at Chapel Hill, 1997.

[Las96] LASZLO M. J.: *Computational Geometry and Computer Graphics in C++*, 1 ed. Prentice Hall, 1996.

[LJCW04] LI X., JIANG H., CHEN S., WANG X.: An efficient surface-surface intersection algorithm based on geometry characteristics. *Computers & Graphics 28*, 4 (August 2004), 527–537.

[LKL*02] LI Y.-Q., KE Y.-L., LI W.-S., PENG Q.-S., TAN J.-R.: Termination criterion for subdivision of triangular Bézier patch. *Computers & Graphics 26*, 1 (February 2002), 67–74.

[LLS01] LITKE N., LEVIN A., SCHRÖDER P.: Trimming for subdivision surfaces. *Computer Aided Geometric Design 18*, 5 (June 2001), 463–481.

[Möl97] MÖLLER T.: A fast triangle-triangle intersection test. *Journal of Graphics Tools 2*, 2 (1997), 25–30.

[NKZ03] NÜRNBERGER G., KOHLMÜLLER N., ZEILFELDER F.: Construction of cubic 3d spline surfaces by Lagrange interpolation at selected points. In *Curve and Surface Design. Saint Malo 2002* (2003), Lyche T., Mazure M.-L., Schumaker L., (Eds.), Nashboro Press, pp. 235–245.

[O'R98] O'ROURKE J.: *Computational Geometry in C*, 2 ed. Cambridge University Press, 1998.

[PBP02] PRAUTZSCH H., BOEHM W., PALUSZNY M.: *Bézier and B-Spline Techniques*. Springer Verlag, 2002.

[RF99] RUIZ J., FEITO F.: Mathematical free-form solid modeling based on extended simplicial chains. In *WSCG '99: VII Conference on Computer Graphics, Visualization and Interactive Digital Media* (Plzen-Bory, Czech Republic, 1999), pp. 241–248.

[Sei89] SEIDEL H.-P.: A general subdivision theorem for Bézier triangles. In *Mathematical Methods in Computer Aided Geometric Design*, Lyche T., Schumaker L., (Eds.). Academic Press, Inc., 1989, pp. 573–581.

[Stü98] STÜRZLINGER W.: Ray-tracing triangular trimmed free-form surfaces. *Transactions on Visualization and Computer Graphics 4*, 3 (Jul–Sep 1998), 202–214.

[VPBM01] VLACHOS A., PETERS J., BOYD C., MITCHELL J.: Curved PN triangles. In *Symposium on Interactive 3D Graphics* (New York, USA, 2001), ACM, ACM Press, pp. 159–166.

[Zei02] ZEILFELDER F.: *Tutorials on Multiresolution in Geometric Modelling*. Mathematics and Visualization. Springer Verlag, 2002, ch. Scattered data fitting with bivariate splines, pp. 243–286.