In *Proceedings IEEE Visualization 2002*, pages 395–402. IEEE Computer Society Press, 2002.

QuadTIN: Quadtree based Triangulated Irregular Networks

Renato Pajarola
Computer Graphics Lab
Information & Computer Science Department
University of California Irvine
pajarola@acm.org

Marc Antonijuan
Multimedia Technology Integration Center
School of Engineering
La Salle University
marc@antonijoan.com

Roberto Lario
Dpto. Arquitectura de Computadores y Automática
Fac. de CC. Físicas
Universidad Complutense Madrid
rlario@dacya.ucm.es

ABSTRACT

Interactive visualization of large digital elevation models is of continuing interest in scientific visualization, GIS, and virtual reality applications. Taking advantage of the regular structure of grid digital elevation models, efficient hierarchical multiresolution triangulation and adaptive level-of-detail (LOD) rendering algorithms have been developed for interactive terrain visualization. Despite the higher triangle count, these approaches generally outperform mesh simplification methods that produce irregular triangulated network (TIN) based LOD representations. In this project we combine the advantage of a TIN based mesh simplification preprocess with high-performance quadtree based LOD triangulation and rendering at run-time. This approach, called QuadTIN, generates an efficient quadtree triangulation hierarchy over any irregular point set that may originate from irregular terrain sampling or from reducing oversampling in high-resolution grid digital elevation models.

CR Categories: I.3.5 Computational Geometry and Object Modeling; I.3.3 Image Generation; E.2 Data Storage Representation

Keywords: multiresolution triangulation, real-time terrain visualization, triangulated irregular networks, level-of-detail

1. INTRODUCTION

Efficient interactive visualization of very large *digital elevation models* (DEMs) is important in a number of application domains such as scientific visualization, GIS, virtual reality, flight simulation, military command & control, or interactive 3D games. Grid digital terrain data sets can easily reach several million vertices while graphics hardware accelerators may be capable of interactively render only a fraction of this at 20 frames or more per second. Due to the generally very large size of DEM terrain data sets and the limited rendering power of graphics systems, efficient *level-of-detail* (LOD) based mesh simplification is required to reduce the geometric scene complexity adaptively and without leading to an intolerable poor visual representation.

Taking advantage of the regular grid structure of common DEM terrain data sets, quadtree based hierarchical multiresolution triangulation methods [18] have proven to be very efficient in terms of LOD selection, triangulation and rendering performance. On the other hand, TIN based triangle mesh simplification methods [5, 2] are generally superior in the triangle count for a given LOD error threshold and allow arbitrary irregular input point data sets. Achieving the high performance of regular quadtree based multiresolution triangulation on irregular point sets is hard to achieve.

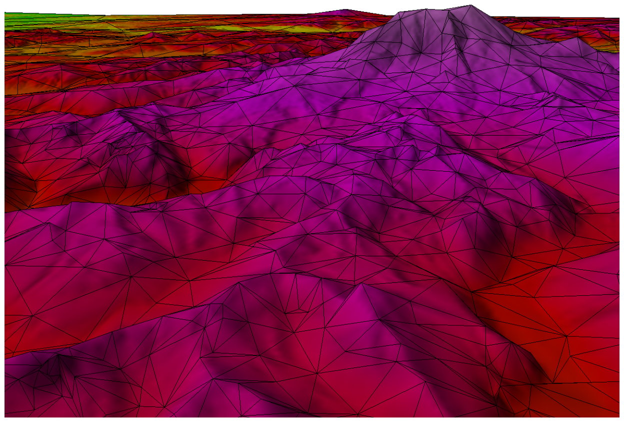


FIGURE 1. QuadTIN triangulation of irregular points of the Puget Sound elevation model. The seamingly irregular triangle mesh is represented and rendered by one single triangle strip.

Having uniform high-resolution terrain data available does not mean that this uniform resolution is desired everywhere at all. Therefore, quadtree based methods do not provide an adequate approach. However, TIN based simplification in a preprocess to eliminate unnecessary detail data (see our examples in Section 5) can optimally remove this redundancy. The remaining data set is not anymore a conforming grid that is directly usable in quadtree based methods.

Our method imposes an efficient quadtree triangulation hierarchy on any irregular TIN based input,[1] thus the name *QuadTIN*. The proposed approach is able to provide fast quadtree based adaptive LOD triangulation and real-time rendering of irregular terrain height-field data using additional Steiner points. Furthermore, due to the imposed restricted quadtree triangulation on the TIN input any arbitrary LOD can be represented as one single triangle strip. See Figure 1 for an example.

The remainder of the paper is organized as follows. Section 2 presents a short overview on related work and in Section 3 we review the quadtree based triangulation method underlying our approach. In Section 4 we describe our QuadTIN triangulation approach. Experiments are presented in Section 5 and Section 6 concludes the paper.

2. RELATED WORK

There has been extensive work on TIN based triangle mesh simplification, refinement methods and multiresolution triangulation for terrains that goes beyond the scope of this paper. We refer the interested reader to the literature for more details (see [3, 5, 6, 9, 12, 13, 19]). Because such TIN based methods work on arbitrary irregular point input data sets they tend to have higher computa-

1. irregular triangulation of elevation points in the 2D projection

tional costs associated with simplification and refinement operations compared to regular hierarchical methods. Furthermore, TIN based multiresolution hierarchies require more complex and costly data structures as well to capture irregular refinement or simplification operations and adjacency relations. Their main advantages are handling of arbitrary point distributions and superior (smaller) triangle counts for given LOD thresholds.

Adaptive quadtree based hierarchical multiresolution triangulations have been studied in the literature for adaptive triangulation of grid-digital terrain elevation models [1, 7, 8, 10, 14, 16, 23]. In [18] we discuss advantages and differences between the various approaches of this class of quadtree [16, 23] or bintree [1, 7, 8, 14] triangulations. These methods take advantage of the regular grid structure to create an efficient multiresolution triangulation hierarchy. The main advantages are simple construction of the multiresolution hierarchy, fast LOD selection and efficient rendering. The main disadvantages are the suboptimal size of an adaptive LOD mesh compared to TINs and the restricted applicability.

In [26] semi-regular, quasi-regular and irregular 4-k meshes are presented. However, this type of irregularity is quite different from what we are considering in this paper. In [26], the semi- and quasi-regular 4-8 meshes are basically *subdivision* methods [25], and the irregular 4-8 meshes are adaptive tesselations of parametric or implicit surfaces. In contrast, this paper discusses how to use any given irregular point set in 2D with additional Steiner points to create a restricted quadtree triangulation.

3. RESTRICTED QUADTREE TRIANGULATION

The *restricted quadtree triangulation* (RQT) method is an adaptive, hierarchical triangulation model [23, 27] to tessellate surfaces. Figure 2 shows the basic recursive quadtree subdivision and triangulation that introduces vertices from the grid in two steps.

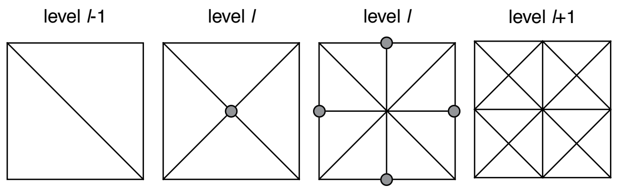


FIGURE 2. Recursive quadtree subdivision and triangulation.

To avoid cracks in the triangulated surface from unrestricted adaptive subdivision and triangulation of the quadtree as shown in Figure 3, RQT subdivision is constraint such that the levels of adjacent quadtree nodes differ by at most one. We call this the *restricted quadtree property*.

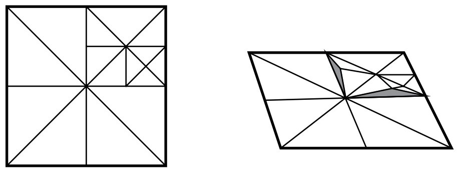


FIGURE 3. Cracks (shaded in grey) resulting from an unrestricted quadtree subdivision.

An efficient variation of this constraint to avoid cracks is the dependency relation shown in Figure 4 that was introduced in [14]. This relation specifies for each vertex v on level l two other vertices v_a, v_b on level l as in Figure 4 b) and d), or on level l-1 as in Figure 4 a) and c), that must be included in the triangulation such that v itself can be selected without introducing a crack.

It has been noted in [7] that a conforming strictly monotonic error metric allows a simple top-down selection of vertices without having to consider the dependency relation shown in Figure 4 or any quadtree subdivision constraint. With a conforming error metric the selected vertices adhere to these constraints automatically. This observation was also made in [16] and a technique, referred to as *error saturation*, was proposed to support a broad range of error metrics.

Furthermore, this class of restricted quadtree or bintree triangulations allows the entire triangle mesh to be represented by one single generalized triangle strip [7, 14, 16] which is important for rendering efficiency.

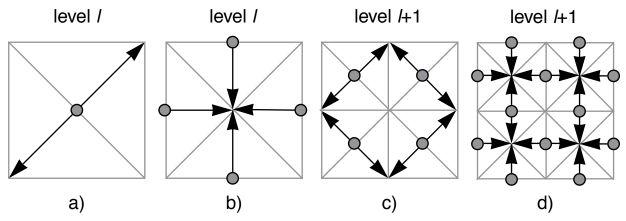


FIGURE 4. Dependency relation of a RQT. The center vertex a) depends on the inclusion of two corners of its quad region. The boundary edge midpoints b) depend on the center vertex. Dependencies within and between the next higher resolution levels are shown in c) and d).

4. QUADTIN TRIANGULATION

In this section we describe the algorithmic details of the proposed QuadTIN method. At this point we assume we are given an irregular point data set or TIN from an irregular-sampling terrain reconstruction process or from preprocessed and simplified terrain height field data.

4.1 Overview

As shown in Figure 5, the QuadTIN approach consists of a preprocess to construct the restricted quadtree hierarchy from a TIN, or irregular set of elevation points. It is important that the input data is organized in an efficient spatial index structure that allows fast range query access for the QuadTIN construction process. In our current version we use a quadtree [21] to manage the input data.

The QuadTIN construction process as described in Section 4.2 iteratively creates a restricted quadtree top-down. The resulting quadtree hierarchy is not balanced but conforming to the RQT constraints. Additional information such as approximation error, bounding spheres and normal cones used for view-dependent triangulation and rendering are also computed for each node at this stage.

The rendering application reads a QuadTIN data structure from file and adaptively triangulates and displays the terrain based on various LOD criteria such as variable (object space) geometric approximation error thresholds or (image space) view-dependent error metrics as outlined in Section 4.4.

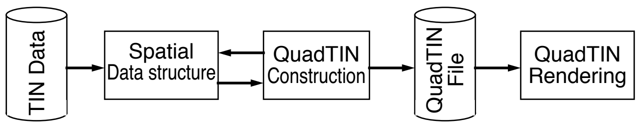


FIGURE 5. Schematic overview of QuadTIN framework.

4.2 Subdivision and vertex selection

The recursive irregular quadtree subdivision by binary triangle splitting and vertex selection for corresponding edge bisection is a crucial part of the QuadTIN construction process. The triangle subdivision as shown in Figure 2 is basically a special case of the longest side bisection triangle refinement method [20, 24] where all triangles are isosceles right-triangles. As shown in Figure 6, in QuadTIN triangles are not anymore isosceles right-triangles. However, we maintain the alternating split-base for edge bisection as in the regular restricted quadtree or bintree triangulation. Furthermore, in QuadTIN an edge is (geometrically) bisected not by introducing the edge's actual midpoint but by adding a vertex from the input data set.

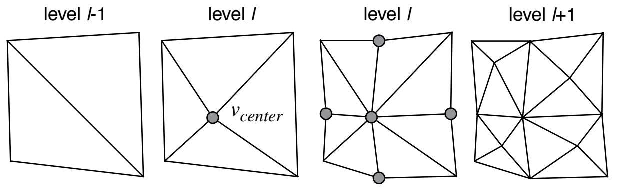


FIGURE 6. Irregular QuadTIN subdivision.

For edge bisection and vertex selection, we have to consider two triangles t and t' forming a quadrilateral region $Q_{t,t'} = t \cup t'$ that share the same base edge $e_{t,t'} = t_{base} = t'_{base}$ that is to be split. The vertical projection of $Q_{t,t'}$ into the *x,y*-plane is called the domain $\partial Q_{t,t'}$. The basic idea is that whenever two base-adjacent triangles have to be refined by edge bisection of their base edge $e_{t,t'}$, the vertex $v \in \partial Q_{t,t'}$ closest to the midpoint of $e_{t,t'}$ is selected as shown in Figure 7 a). However, to avoid bad aspect-ratio triangles only points from a restricted domain as shown in Figure 7 b) are considered.

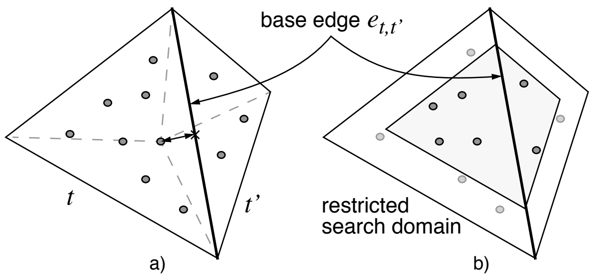
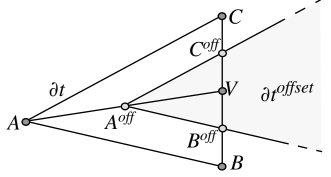


FIGURE 7. Vertex closest to the midpoint of base edge $e_{t,t'}$ is selected for edge bisection a) but only from within a restricted search domain b).

To define the restricted search domain exactly let us first define the *offset region* ∂t^{offset} of a triangle t as shown in Figure 8. For a given triangle $\partial t = (A, B, C)$ in the *x,y*-plane with base edge $e = \overline{BC}$ and offset factor *f* we define ∂t^{offset} to be the intersection of the open half spaces defined by $\overline{B^{off}A^{off}}$ and $\overline{A^{off}C^{off}}$. For a quadrilateral $Q_{t,t'}$ we define the restricted search domain to be $\partial Q^{offset}_{t,t'} = \partial t^{offset} \cap \partial t'^{offset}$.



$V = (B + C) / 2$
$A^{off} = V + f \cdot (A - V)$
$B^{off} = V + f \cdot (B - V)$
$C^{off} = V + f \cdot (C - V)$

FIGURE 8. Offset region ∂t^{offset} for f = 50% of one triangle ∂t=(A,B,C) used to define the restricted search domain.

It is possible that base-adjacent triangles t and t' that must be split form a *non convex* quadrilateral domain $\partial Q_{t,t'}$ as shown in Figure 9 a). Care has to be taken not to select any vertex $v \in \partial Q_{t,t'}$ that may cause overlapping or flipped triangles. By definition, the restricted search domain $\partial Q^{offset}_{t,t'}$ avoids such degeneracies as shown in Figure 9 b).

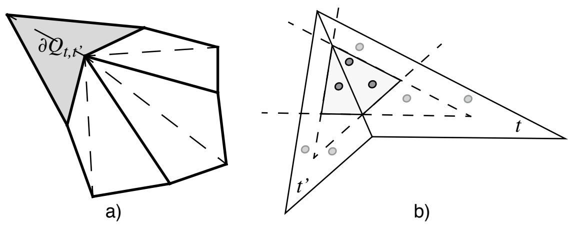


FIGURE 9. Non-convex quadrilateral a), and restricted point search domain for edge bisection b).

Due to the dependency constraints, outlined in Section 3 and Figure 4, irregular point distributions and the restricted search domain it is possible that two triangles must be split where there are no points available from the input data set for edge bisection. In that case we insert a so called *Steiner point* (see also [4]) within the region $\partial Q^{offset}_{t,t'}$ to guarantee a matching restricted quadtree triangulation. The coordinates and normal vectors of Steiner points are locally interpolated from the nearest neighbors of the input data set as outlined in the following section.

On the boundary of the input data, triangle subdivision differs slightly. Choosing the vertex closest to the midpoint of t_{base} could lead to a situation where unprocessed input points are outside of the boundary of the refined triangulation as shown in Figure 10 a). To avoid this problem and preserve the *inside-boundary* relation at any time, the vertex $v_{split} \in \partial t^{offset}$ that minimizes the largest angle formed with t_{base} is selected as shown in Figure 10 b).

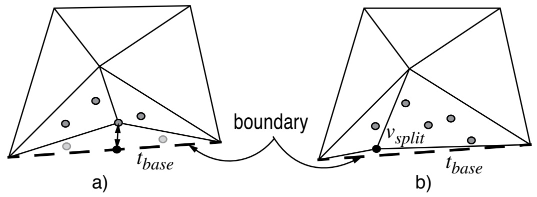


FIGURE 10. Vertex closest to midpoint causes input points to be outside the boundary after edge bisection a), and b) modified vertex selection retaining inside-boundary relation.

The restricted search domain parameter *f* mainly determines the number of inserted Steiner points. Without offset restrictions, *f* = 100%, only the minimal necessary number of Steiner points is inserted that satisfies the dependency restrictions of the quadtree triangulation outlined in Section 3, and that prevents flipped triangles due to concave quadrilaterals as shown in Figure 9.

4.3 Construction preprocess

The process to create the QuadTIN data structure consists of the following components:

- Top-down construction of restricted quadtree hierarchy.
- Vertex selection for edge bisection.
- Interpolation of Steiner points.
- Computation of error metric and view-dependent rendering parameters.

Top-down hierarchy construction: The QuadTIN hierarchy *H* is initialized with a quadtree root node containing two triangles defined by the corners and one diagonal of the bounding box of the input data set. In general, the corners of the bounding box are interpolated Steiner points. The hierarchy is iteratively created by subdividing the leaf nodes and triangles of the quadtree *H* in breadth-first order and center-vertices first (see Figure 6) which allows preserving the restricted quadtree property at any time during the construction process. If a triangle *t* has to be subdivided, first it is checked that the base-adjacent triangle *t'* already exists. If not, subdivision is propagated to the parent triangle of *t'* first. Unless a subdivision is induced by a dependency relation, the top-down triangle subdivision stops if no more input points are within $\partial Q_{t,t'}$.

Edge bisection: As outlined in the previous section, for each subdivision of triangles t and t' the input point set must be searched for candidate vertices within the restricted search domain $\partial Q_{t,t'}^{offset}$ to perform edge bisection. To support such spatial search queries efficiently, the input points are loaded into a spatial indexing structure. We currently use a region quadtree [21] to quickly select a subset of input points within the bounding box of $Q_{t,t'}$. Then we reduce this subset to valid candidate points within $\partial Q_{t,t'}^{offset}$. Finally, the vertex $v \in \partial Q_{t,t'}^{offset}$ closest to the midpoint of $e_{t,t'}$ is selected for edge bisection and removed from the input point set or marked as used. If no suitable input point can be found, $\{v | v \in \partial Q_{t,t'}^{offset}\} = \emptyset$, a new Steiner point is inserted.

Steiner points: To interpolate the coordinates and normal vectors of Steiner points, again the input point set is searched. For every Steiner vertex v_S that is inserted in $Q_{t,t'}$ at least three and up to four closest points within $Q_{t,t'}$ from the input data set, or corners of the incident triangles t and t', are selected. The *z*-coordinate and normal vector of v_S are averaged from these closest points. The *x,y*-coordinates are set to the midpoint of the bisected edge $e_{t,t'}$. If not initially given, the vertex normals of the input points are calculated from the TIN input file as average from the adjacent triangles.

Error metric: The error metric that QuadTIN uses is the L_∞ norm of the vertical distance (terrain elevation dimension) of vertices to the triangulated surface. Let us denote the minimum vertical distance of a vertex *v* to two triangles t and t' by $d(v, Q_{t,t'})$, and let v_{split} be the vertex selected for subdividing the base edge $e_{t,t'}$. Therefore, the error of v_{split} is initialized to

$$E(v_{split}) = \max_{v \in \partial Q_{t,t'}}(d(v, Q_{t,t'})) . \qquad \textbf{(EQ 1)}$$

This error metric provides a conservative LOD triangulation, the top-down vertex selection and triangle subdivision can be stopped at nodes with $E(v_{split})$ below a given tolerance. By definition of the error metric, stopping selection at v_{split} with $E(v_{split})$ below the given error threshold guarantees that there is no other vertex within $\partial Q_{t,t'}$ farther than $E(v_{split})$ from the current triangles *t* and *t'*. Because this error metric is not guaranteed to be monotonic, cracks in the triangulation have to be avoided by propagating subdivision according to the dependency relations shown in Figure 4. Thus propagated subdivisions must verify the error threshold as well. Note that in [15] a view-dependent error metric was proposed that is monotonic and does not require dependency resolution via split propagation.

Rendering parameters: Besides the geometric approximation error that is stored with each vertex, each node $h \in H$ of the QuadTIN hierarchy also stores *bounding sphere* (center *h.c* and radius *h.r*) and *bounding normal cone* [22] (cone axis *h.n* and semi opening angle *h.θ*) parameters used for view-dependent triangulation and rendering. The bounding sphere of a node *h* encloses all vertices, and the bounding normal cone bounds all triangle normals of descendants of *h*. These parameters are computed bottom-up after the basic quadtree data structure has been constructed.

4.4 Real-time rendering

Our interactive visualization application reads the preprocessed QuadTIN data structure from a binary file. The real-time rendering process performs the following steps for each frame:

1. View-dependent vertex selection.
2. Dependency resolution.
3. Triangle strip construction.
4. Rendering.

The top-down traversal of the QuadTIN hierarchy *H* for vertex selection is similar to the algorithms presented in [16, 23]. However, for a fly-through application vertices are selected based on a view-dependent (image space) error metric as outlined below. Due to this view-dependent vertex selection the RQT constraints have to be satisfied by including all vertices according to the dependency relation shown in Figure 4. Dependency resolution is efficiently performed in linear time with respect to the size of the generated triangle mesh as shown in [16] and is not further discussed here.

Furthermore, due to the imposed restricted quadtree hierarchy on the TIN input data set, including a few additional Steiner points, the presented QuadTIN approach can *represent different LOD triangulations of an irregular point set using only one single triangle strip* (with swap operations). Note that also the triangle strip generation algorithm [16] is linear in time with respect to the number of rendered triangles.

View-dependent vertex selection: The vertex selection takes three view-dependent selection criteria into account: *view-frustum culling*, *back-face culling* and *screen projection tolerance*. These criteria allow efficient back-tracking during the recursive traversal of the QuadTIN hierarchy *H* for vertex selection. If the bounding sphere of a node $h \in H$ does not intersect the view frustum (approximated by a viewing cone) or if the bounding normal cone indicates a completely back-facing region, recursive vertex selection can be stopped at this node *h*. For performance reasons QuadTIN computes view-frustum and back-face culling similar to [17] using only a few floating point operations, see also Figure 11. We assume that the viewpoint *e* and semi-angle ω (as well as its sine, cosine and tangens) of the viewing cone are given for each frame. Furthermore, for each node $h \in H$ we know its bounding sphere (*c, r*) and bounding normal cone (*n*,*θ*).

View-frustum culling as shown in Figure 11 a) is performed if $\gamma - \alpha > \omega$ or $\cos(\gamma - \alpha) < \cos(\omega)$. Using the rules $\cos(\alpha) = \sin(90 - \alpha)$ and $\sin(\alpha + \beta) \le \sin(\alpha) + \sin(\beta)$ this can be rewritten to

$$\cos(\gamma) + \sin(\alpha) < \cos(\omega) . \qquad \textbf{(EQ 2)}$$

Equation 2 can efficiently be evaluated without trigonometric functions by using a dot-product $\cos(\gamma) = ((c - e) \bullet w)/|c - e|$ and an additional division. $\cos(\omega)$ can be precomputed once as long as the FOV aperture angle does not change between frames and $\sin(\alpha) = r/|c - e|$. A few additional relations such as $\cos(\gamma) > \cos(\omega) \Rightarrow$ *continue selection*, $|c - e| < r \Rightarrow$ *continue selec-*

tion, and $(c-e) \bullet w < -r \Rightarrow$ *stop selection* need to be tested before Equation 2 can be used to stop recursive vertex selection.

While view-frustum culling as explained above is slightly different and more efficient than in [17], back-face culling as shown Figure 11 b) is practically identical. Recursive vertex selection can be stopped at node h if $\beta + \theta < 90°$ or $\cos(\beta) > \cos(90° - \theta)$. This is equivalent to

$$\cos(\beta) > \sin(\theta)\,, \qquad \textbf{(EQ 3)}$$

and can be computed efficiently without trigonometric functions as shown in [17].

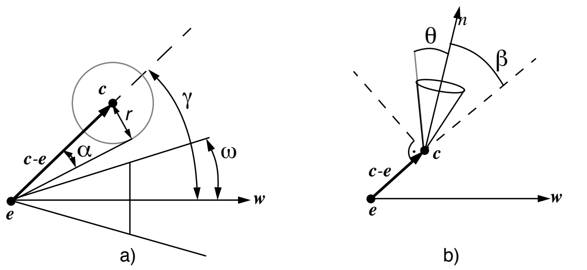


FIGURE 11. a) View-frustum culling if $\gamma - \alpha > \omega$ and b) back-face culling if $\beta + \theta < 90°$ are used for efficient back-tracking in vertex selection.

In addition to view-frustum and back-face culling which are used for back-tracking, a screen projection error tolerance is used for vertex selection as well as back-tracking. A vertex v_{split} that refines triangles t and t' is selected only if its geometric approximation error $E(v_{split})$ projected on screen exceeds a given tolerance τ. Given the normalized viewing direction w and the focal length d of the viewing plane as shown in Figure 12, the projected error on screen that is compared to τ for vertex selection is

$$E_{screen} = \frac{E(v_{split}) \cdot d}{(v_{split} - e) \cdot w}\,. \qquad \textbf{(EQ 4)}$$

Note that the approximation error $E(v_{split})$ as computed in Equation 1 does not actually have to occur at v_{split} itself but can occur at any point within the bounding sphere. Therefore, for the purpose of back-tracking we use an adjusted conservative screen projection error given by

$$E_{screen} = \frac{E(v_{split}) \cdot d}{(v_{split} - e) \cdot w - r}\,. \qquad \textbf{(EQ 5)}$$

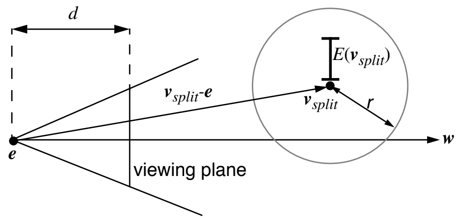


FIGURE 12. Screen projection of the approximation error is performed by perspective division.

Rendering: The rendering process mainly takes advantage that due to the imposed quadtree hierarchy the irregular point set can be triangulated with one single triangle strip. We use the algorithm presented in [16] to generate a generalized triangle strip including swap operations. The so created triangle strip (see also [7, 8, 14]) contains almost as many swap operations as triangles. If a swap operation is simulated in OpenGL by repeating the second last vertex and thus creating a degenerate line-triangle, the resulting triangle strip contains almost twice the number of actually visible triangles. Note however, that even in the worst case this triangle strip is still better then an indexed triangle list. This is because even with swap operations no more than 2 vertices (amortized) per visible triangle are processed for transform and lighting compared to 3 per triangle with an indexed triangle list.

To avoid popping artifacts in multiresolution triangulations *vertex morphing* was proposed in [11] to smoothen the transition between different LODs. In vertex morphing, a vertex v is first inserted at an interpolated start position v_{start} and then smoothly translated to its end position v_{end} by $v = (1-t) \cdot v_{\text{start}} + t \cdot v_{\text{end}}$ over some time, or number of frames. This works well when inserting vertices since the start position can be interpolated from the current triangulation and because the end position is well defined by the vertex' actual coordinates. On the other hand, vertex morphing is problematic when removing vertices to decrease the LOD. In that case the start position v_{start} is the vertex v's current coordinates as used for rendering. However, the end position v_{end} is not well defined as shown in Figure 13. If any of the vertices, from which the end point v_{end} is interpolated, is itself a morphing vertex then it is difficult to efficiently prevent any popping at the time when v has to be removed from the mesh.

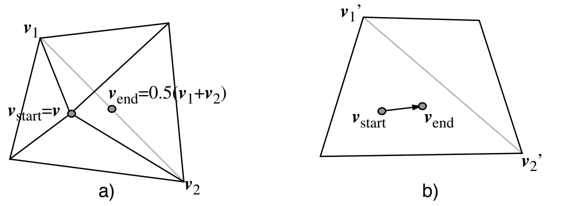


FIGURE 13. Vertex morphing endpoint interpolation at the beginning of the morph a) and at the end when the interpolated vertices are modified as well.

In addition to the above problems of correctly determining end positions for vertex removal, morphing also poses problems in the vertex selection process. Creating a particular LOD triangulation is not anymore a one-step process because vertices not only have to be selected but must also actively be removed over a number of frames. Thus vertices must be actively scheduled for removal as soon as they are selected only due to dependency relations from vertices that are already in the process of being removed. Due to the asymmetry of vertex morphing at insertion and removal time and due to open problems in LOD selection under dynamic vertex removal, morphing is currently not implemented within our rendering algorithm.

5. EXPERIMENTS

In this section we present experimental results with digital elevation models (DEMs) from the United States Geological Survey (USGS). The original DEM files are all regular grid height-field data sets from which we generated irregular TIN representations in a mesh simplification preprocess. We used the *Terra* software [9] available from http://graphics.cs.uiuc.edu/~garland/CMU/scape/ to simplify the DEMs to a fraction of the original data size using only very small error tolerances. The so generated TINs were used as example input data sets to our QuadTIN construction algorithm and QuadTIN rendering application. The downloaded Terra software has some limitations, it processes a height-field with a fixed spacing of one unit between points in x,y. To avoid any loss of resolution in the elevation dimension we did not scale (down) the elevation accordingly but processed the input file conservatively using Terra with exaggerated elevation.

We used the following terrain data sets to test our QuadTIN approach: Isabel Valley, Mindego Hill, Palo Alto,[1] San Jose[1] (all available from http://bard.wr.usgs.gov/), Puget Sound, and Grand Canyon (both from http://www.cc.gatech.edu/projects/large_models/).

5.1 QuadTIN construction

Table 1 shows results of the preprocess using Terra mesh simplification and the QuadTIN construction process. The Terra preprocess was conducted on a Sun Enterprise 450 server with 4GB memory and running on one of the four 296MHz UltraSPARC-II CPUs. The QuadTIN construction process was performed on a Sun Ultra60 workstation with a 450MHz UltraSPARC-II and 512MB main memory.

Despite the fact that the QuadTIN construction process performs demanding spatial search operations on the TIN input data, executes complicated distance and error metric calculations, inserts new Steiner points and computes a bounding sphere and bounding normal cone hierarchy it is very efficient. The construction process operated at a rate of processing more than 4300 points per second for most input files. One can also observe that the ratio of inserted Steiner points in comparison to the input data set is only about 23% for an offset factor of $f = 80\%$ for the restricted search domain (see also Section 4.2).

In Table 1 we also show the unusual results obtained with the Grand Canyon data set. This data set is unlike the others in several aspects. It has a very poor resolution of only 8 bits in the elevation dimension (only 256 different altitude values). This causes the Terra preprocess to perform poorly in terms of mesh simplification. The unusual simplification and extreme point distribution also causes the QuadTIN construction to perform poorly. Altogether, this outlier data set performs about 2 times worse in all aspects than the other data sets.

Model	#points	Terra	error	#points	QuadTIN	#points	%Steiner
Isabel Valley	154795	16"	0.4	45770	10"	56610	23.68
Mindego Hill	1507815	2'43"	1.2	230117	51"	282475	22.75
Palo Alto	6021376	14'46"	0.8	1147256	4'29"	1418299	23.63
San Jose	17958000	35'06"	1.2	2008583	7'54"	2461123	22.53
Puget Sound	16785409	36'32"	6	2073276	8'17"	2563548	23.65
G. Canyon	8394753	33'	2.0	2540172	16'29"	3825217	50.59

TABLE 1. First four columns: number of points of initial models, time used by Terra for simplification, preprocess error tolerance in meters, and size of the resulting TIN. Last three columns: QuadTIN construction time, resulting data set size including inserted Steiner points, and percentage of Steiner points added in comparison to the TIN input size.

5.2 Triangulation and rendering

In Figure 14 we show the numbers of selected vertices for different object-space geometric approximation error thresholds for a $(2^{11}+1) \times (2^{11}+1)$ subset of the Palo Alto data set. We compared a plain restricted quadtree triangulation (RQT) built on the full resolution height-field, and our QuadTIN approach with two different Terra preprocess variants to corresponding simplifications with Terra which are single resolution meshes. Despite the added Steiner points our QuadTIN approach outperforms the regular grid RQT method significantly in terms of selected vertices for the same LOD. The Terra results are shown for comparison only since Terra provides only discrete simplifications and is not a multiresolution method. Figure 19 illustrates adaptive QuadTIN triangulation for a perspective view of the Mindego Hill data set at varying image-space error thresholds τ.

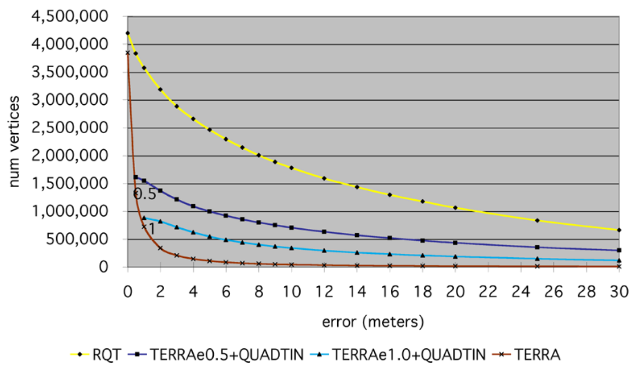


FIGURE 14. Number of vertices used to represent a LOD triangulation for different object-space approximation errors.

As shown in Figure 15 for the Palo Alto model, the ratio of Steiner points is as low as 14% for coarse triangulations with high approximation errors. This shows that the QuadTIN triangulation adopts extremely well to the TIN input at coarse levels, and the inserted Steiner points are mainly required for conforming quadtree triangulations at high LODs. This can also be seen nicely in Figure 18 where we observe that the number of selected Steiner points (red dots) is larger in regions close to the actual viewpoint (with high LOD) than in regions farther away (with low LOD).

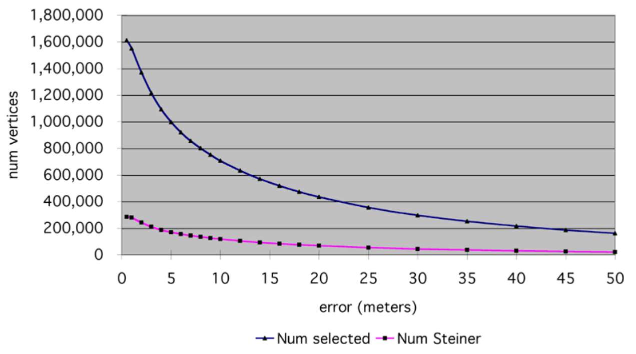


FIGURE 15. Number of Steiner points in comparison to the overall number of selected vertices for different object-space approximation errors and offset factor $f = 80\%$.

Figure 16 shows experimental results of the Palo Alto data set (1.4M vertices) rendered on a Dell 1.7GHz Pentium 4 with QuadroPro2 graphics card. The results are amortized over a large number of frames while interactively viewing the data set. Figure 16 shows per frame timing results of our rendering algorithm at varying image-space error thresholds τ. The corresponding number of selected vertices, relative per vertex timing, and frame rates are given in Table 2. We can see that the number of selected vertices and the rendering cost decreases rapidly (exponentially) as the image-space error threshold value τ increases slowly (linearly). One can also observe that the overall display performance for very small tolerances τ is dominated by the vertex

1. multiple terrain data files merged together

selection time, which performs the error calculations and LOD based vertex selection. Furthermore, looking at the display cost measured per rendered vertex it is noticeable that the system actually amortizes the cost better per vertex for small τ and larger meshes, despite the fact that the frame rate is lower (compare to Figure 16). However, this observation is not surprising since the constant overhead costs of the algorithms and data structures have more impact if the number of displayed vertices is small.

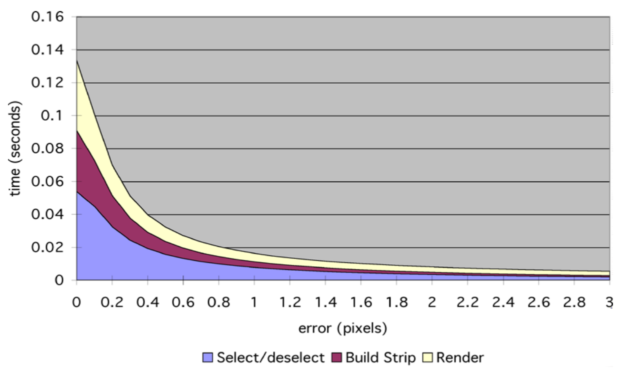


FIGURE 16. Per frame timing results of the rendering algorithm for different screen projection error thresholds τ.

Error τ (pixels)	0.0	0.5	1.0	1.5	2.0	2.5	3.0
selected vertices	254703	35983	14356	8093	5288	3761	2824
per frame timing	0.13s	0.032s	0.016s	0.010s	0.008s	0.006s	0.005s
per vertex timing	0.52μs	0.89μs	1.13μs	1.32μs	1.52μs	1.71μs	1.91μs
frame rate	7	30	61	93	124	155	200

TABLE 2. Timing experiments for different screen-space error thresholds τ.

In Figure 17 we show experimental results of the Mindego Hill data set (280K vertices) rendered on a Dell 2.2GHz Pentium 4 with GeForce4Ti 4600graphics card. Figure 17 shows per frame rendering cost comparison between standard triangle strip and vertex-cache based rendering. It does not include timing for vertex selection or triangle strip generation but only the rendering cost. The standard method keeps the vertices in the application's system memory and uses an indexed triangle strip to render the triangulated terrain model. In contrast, the vertex-cache approach stores all vertices of the data set on the graphics card's memory using a NVIDIA OpenGL extension. The performance improvement is particularly significant for very small screen-space error tolerances τ with high triangle counts. Note that all vertices of the Mindego Hill data set fit into the graphics card's memory.

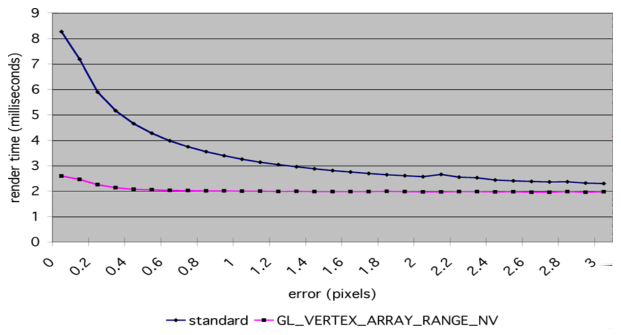


FIGURE 17. Comparison of rendering time between standard triangle strip rendering and rendering using vertex caching on the graphics card.

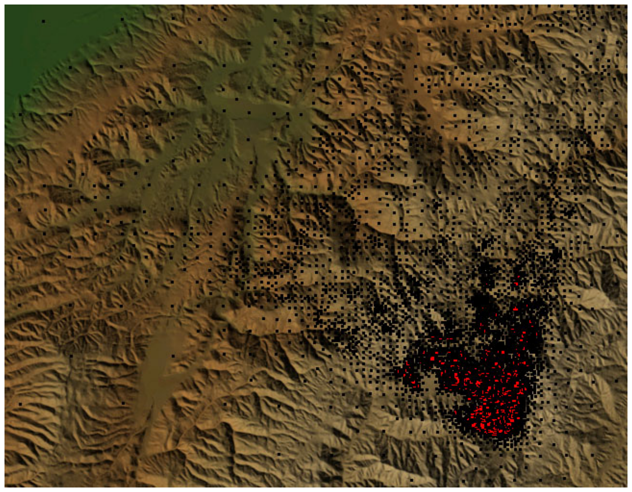


FIGURE 18. Example of Steiner points (red dots) shown in comparison to all selected vertices for a perspective view of the San Jose data set (shown from bird's eye view).

6. CONCLUSION

In this paper we presented QuadTIN, a novel approach to impose quadtree-based multiresolution triangulation hierarchy on arbitrary irregular sets of elevation points. Using a limited number of Steiner points this triangulation hierarchy allows fast view-dependent LOD triangulation, triangle strip generation and real-time rendering. We also introduced an efficient heuristic how such a hierarchy can efficiently be constructed with a minimal number of Steiner points.

Experiments on a variety of meshes have shown the efficiency and applicability of our approach. Compared to TIN based multiresolution approaches the QuadTIN triangulation hierarchy provides better triangulation and rendering speed, and compared to quadtree or bintree triangulations based on regular grids QuadTIN requires significantly fewer vertices for given error thresholds and less storage space.

ACKNOWLEDGEMENTS

The digital elevation models shown in this paper all originate from the US Geological Survey, and we thank the large geometric models project at the GVU Center at Georgia Tech for the preparation of the textured Puget Sound and Grand Canyon data sets. Furthermore, we would like to thank the University Complutense of Madrid and the Spanish research grant TIC 99-0474 for their generous support of Roberto Lario's visit at UCI. We also want to thank the students Gary Oldham, Steven Hoey and Amy Wright for their work in the early stages of this project.

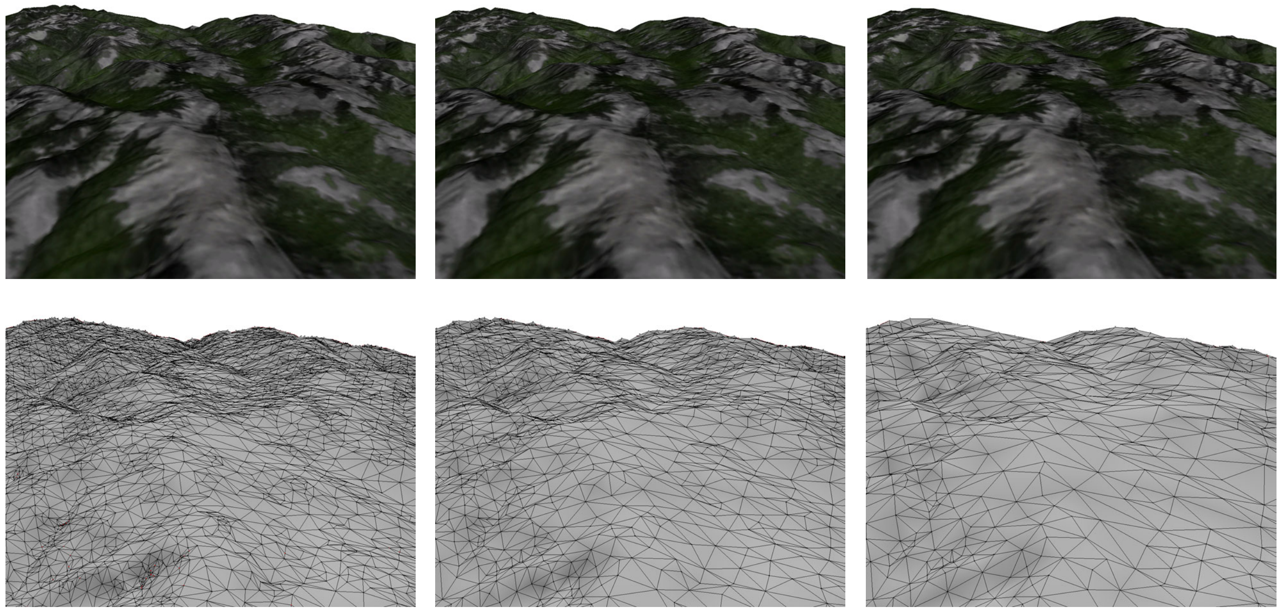


FIGURE 19. QuadTIN triangulations of the Mindego Hill data set with screen projection error threshold $\tau < 4$, 7 and 12 pixels (from left to right), resulting in 7984, 3789 and 1601 selected vertices respectively.

REFERENCES

[1] Laurent Balmelli, Serge Ayer, and Martin Vetterli. Efficient algorithms for embedded rendering of terrain models. In *Proceedings IEEE Int. Conf. on Image Processing ICIP 98*, pages 914–918, 1998.

[2] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

[3] Mark de Berg and Katrin Dobrindt. On levels of detail in terrains. In *11th Symposium on Computational Geometry*, pages C26–C27. ACM, 1995.

[4] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[5] Leila De Floriani, Paola Marzano, and Enrico Puppo. Multiresolution models for topographic surface description. *The Visual Computer*, 12(7):317–345, August 1996.

[6] Leila De Floriani and Enrico Puppo. Hierarchical triangulation for multiresolution surface description. *ACM Transactions on Graphics*, 14(4):363–411, 1995.

[7] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Marc C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In *Proceedings IEEE Visualization 97*, pages 81–88, 1997.

[8] Williams Evans, David Kirkpatrick, and Greg Townsend. Right-triangulated irregular networks. *Algorithmica*, 30(2):264–286, March 2001.

[9] Michael Garland and Paul S. Heckbert. Fast polygonal approximation of terrains and heigt fields. Technical Report cmu-cs-95-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1995.

[10] Thomas Gerstner. Multiresolution compression and visualization of global topographic data. Technical Report 29, Institut f"ur Angewandte Mathematik, Universit"at Bonn, 1999. to appear in Geoinformatica 2001.

[11] Hugues Hoppe. Progressive meshes. In *Proceedings SIGGRAPH 96*, pages 99–108. ACM SIGGRAPH, 1996.

[12] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings IEEE Visualization 98*, pages 35–42. Computer Society Press, 1998.

[13] Reinhard Klein, Daniel Cohen-Or, and Tobias H/"uttner. Incremental view-dependent multiresolution triangulation of terrain. In *Proceedings Pacific Graphics 97*, pages 127–136. IEEE, Computer Society Press, 1997.

[14] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings SIGGRAPH 96*, pages 109–118. ACM SIGGRAPH, 1996.

[15] Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. In *Proceedings IEEE Visualization 2001*, pages 363–370. Computer Society Press, 2001.

[16] Renato Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In *Proceedings IEEE Visualization 98*, pages 19–26,515, 1998.

[17] Renato Pajarola. Fastmesh: Efficient view-dependent meshing. In *Proceedings Pacific Graphics 2001*, pages 22–30. IEEE, Computer Society Press, 2001.

[18] Renato Pajarola. Overview of quadtree-based terrain triangulation and visualization. Technical Report UCI-ICS-02-01, Information & Computer Science, University of California Irvine, 2002. submitted for publication.

[19] Enrico Puppo. Variable resolution terrain surfaces. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 202–210, 1996.

[20] M. C. Rivara. A discussion on mixed (longest-side midpoint insertion) delaunay techniques for the triangulation refinement problem. In *Proceedings of the 4th International Meshing Roundtable*, pages 335–346, 1995.

[21] Hanan Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2):187–260, June 1984.

[22] Leon A. Shirman and Salim S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. In *Proceedings EUROGRAPHICS 93*, pages 261–272, 1993. also in Computer Graphics Forum 12(3).

[23] Ron Sivan and Hanan Samet. Algorithms for constructing quadtree surface maps. In *Proc. 5th Int. Symposium on Spatial Data Handling*, pages 361–370, August 1992.

[24] Christoph Stamm, Stephan Eidenbenz, and Renato Pajarola. A modified longest side bisection triangulation. In *Electronic Proceedings of the 10th Canadian Conference on Computational Geometry*, 1998. URL http://cgm.cs.mcgill.ca/cccg98/proceedings/cccg98-stamm-modified.ps.gz.

[25] Luiz Velho. Using semi-regular 4-8 meshes for subdivision surfaces. *Journal of Graphics Tools*, 5(3):35–47, 2001.

[26] Luiz Velho and Jonas Gomes. Variable resolution 4-k meshes: Concepts and applications. *Computer Graphics Forum*, 19(4):195–214, 2000.

[27] Brian Von Herzen and Alan H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *Proceedings SIGGRAPH 87*, pages 103–110. ACM SIGGRAPH, 1987.