On-the-Fly Adaptive Subdivision Terrain

Dirc Rose, Martin Kada, Thomas Ertl

University of Stuttgart, Institute of Computer Science
Visualization and Interactive Systems Group
Breitwiesenstraße 20-22, 70565 Stuttgart, Germany
Email: {rose|ertl}@informatik.uni-stuttgart.de
martin.kada@ifp.uni-stuttgart.de

Abstract

In this paper we present a method to achieve interactive rendering of smooth terrain based on coarse data. Therefore we use adaptive subdivision surfaces which are calculated on the fly by using information from the directed edge based mesh of the previously rendered frame and throwing away triangles not used any longer. This way we are able to gain fast, high quality meshes which are refined depending on the point of view with marginal memory consumption. Additionally, this technique is compatible to traditional data structures like progressive meshes.

1 Introduction

Although terrain rendering has improved a lot with regard to quality and speed in the recent years, there are still some shortcomings when approaching a surface. Massive use of textures may give a terrain a natural appearance and this works fine when viewed from afar, but when closing up the flat facets of a terrain can be clearly spotted in general cases. This may be annoying, e.g. in flight simulators which show a smooth surface from above but coarse hills when in low level flight or rolling on the ground. Low resolution can originate from several facts. Rendering speed should not be the bottleneck, since progressive meshes and level-of-detail overcome this problem. The main aspect is memory consumption, both main memory and hard drive or CD/DVD capacity. It seems impossible nowadays to save the elevation data of the whole world with a resolution of 1m×1m×1m. When disregarding the oceans this still yields about $1.5 \cdot 10^{13}$ bytes of uncompressed, difficult to manage data. Therefore either the range of interaction or the resolution has to be restricted. In many cases the latter is chosen and the typical distance between two sample points is about 100m, and 30m or below for more interesting regions.

This results in the mentioned visual defect when approaching the ground. We want to show a way to polish rough edges with almost no supplementary use of memory by adaptively subdividing the terrain data on the fly. This strategy meets today's typical computer equipment, e.g. console platforms, providing a low- to mid-end graphics board, relatively little memory, and nevertheless a quite fast processor.

2 Related Work

In recent years a lot of research has been conducted in simplification of complex meshes. In a first preprocessing step these techniques [14, 9] can be used to reduce high resolution data to a manageable amount of samples. In a second preprocessing pass a progressive mesh structure [3] may be calculated. During interaction these progressive meshes or similar algorithms [8, 10, 11, 15] can be used to build up a view-dependent level-of-detail representation of the scene.

In our scenario we adapt data structures based on these methods. However, when approaching the ground we want to switch the direction and refine the mesh. Since we had to drop most of the detailed "smooth" information due to lack of memory, we must find a way to regain or reinvent the lost data. For this purpose we use subdivision patches originally developed as an enhancement to B-spline surfaces. In general these are used to describe objects, e.g. in CAD, with just a few control points. The possibility to use subdivision algorithms as a method to reconstruct surfaces is shown in [12].

So far subdivision approaches were too slow to allow interactive frame rates. In the following sections we will present a way to overcome this problem, therefore allowing smooth looking surfaces even on low-memory platforms.

3 Subdivision Surfaces

There are many forms of subdivision surfaces [17]. These surfaces are generated by consecutively dividing an initial coarse mesh, converging against a smooth surface. The subdivision is controlled by a subdivision mask or stencil. This mask determines which coordinates of neighboring points are combined to calculate the newly inserted point of an edge. Loop and Catmull-Clark surfaces [5] are approximating subdivision surfaces. Similar to B-splines, these surfaces are close to the control points but do not contain them. Due to this property we cannot use this kind of surfaces because we want to keep the original coordinates and not alter them. We use an interpolating scheme instead, the butterfly surface [6], whose initial mask (the solid lines of Figure 1) looks like a butterfly. For better results this mask can be extended by two additional points [7, 18], as shown in Figure 1. The points are multi-

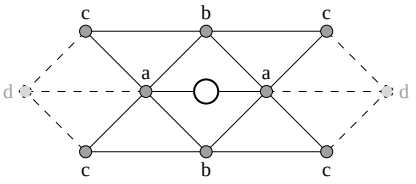


Figure 1: Butterfly subdivision mask for the white point in the middle

plied by different weights $a - d$, with the parameter w allowing a more subtle control of the resulting surface:

$$a : \frac{1}{2} - w, b : \frac{1}{8} + 2w, c : -\frac{1}{16} - w, d : w \quad (1)$$

Setting $w = 0$ results in the original butterfly mask. The tangent space can be calculated using the complete 1- and 2-ring neighborhood of a point as shown in Figure 4 and with this information it is easy to obtain the normal at this point. For the weights please refer to [17].

We have implemented an adaptive algorithm that only subdivides a triangle when this will result in a visible change. This can be either a change in height by one pixel or more or it can be a significant change in normal direction. As a first order approximation for both cases we can use the distance between the yet to be inserted point and the center point of the corresponding edge. Scaled by the ratio of window size to z depth, it is a quality measure in screen space dimensions. Setting the error bound to 1 pixel or below will produce an adaptive mesh that looks the same as an infinitely subdivided surface.

During the subdivision process a red-green triangulation strategy [1, 2] is used to avoid cracks between neighboring triangles. Therefore we permit only a difference of one level in subdivision between two neighboring triangles. This procedure also guarantees the easy registration of the subdivision mask at any time.

The C^1 continuity at the borders permits a smooth transition to the surrounding original coarse mesh. Subdivision of the border will not be allowed until it is needed due to higher order divisions of neighboring triangles. In this case the border is subdivided linearly to guarantee that there will not be any cracks between the subdivided parts and the original surrounding mesh.

4 On-the-Fly Control

The key to achieve interactive frame rates lies in an efficient algorithm for inserting and destroying newly subdivided triangles which are not needed any more. The main data structure is based on a lean version of directed edges [4]. Here a half-edge is represented by the address of the vertex where the directed edge points to. Thus a triangle is defined by three consecutive half-edges. The neighboring triangle of an edge can be accessed via another pointer that addresses the opposing half-edge. Progressive meshes use the same or a similar data organization in general which facilitates the combination with our algorithm. If a neighboring triangle is subdivided one level less than the original triangle, then its half-edge points to only one of the juxtaposed half-edge as can be seen in Figure 2.

This is one reason why we introduce mid-vertices to the original half-edge structure. These mid-vertices belong to every edge and contain the position which would arise when subdividing this edge. Additionally we can attach the pointer to the second neighboring half-edge to it and therefore we

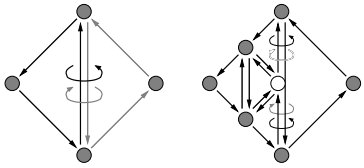


Figure 2: Half-edge structure with white mid-vertex

can save the costly search for this second edge.

Another advantage of mid-vertices is the ease to calculate the next subdivision level of a triangle. In regions of high curvature we can benefit from these precalculated vertices because the registration of the subdivision mask is more favourable. The calculation of the mid-vertices itself is quite simple since the neighboring triangles do not have the strong fragmentation they may have one step later. In some cases such a fragmentation may induce a laborious search for the butterfly mask vertices along the half-edges, as shown in Figure 3a. The retrieval is straightforward if the mid-vertices have been precalculated on a coarser level (Figure 3b).

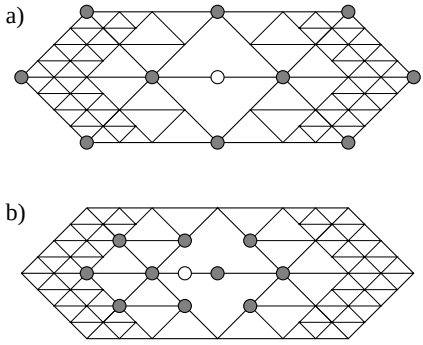


Figure 3: a) The way along the directed edges to the outer gray points can be lengthy
b) Using precalculated mid-vertices may save work

The mid-vertices are also useful when calculating the tangent vectors since we need the 1- and 2-ring neighborhood. The mid points help to limit the expansion, since they are already subdivided one level higher as can be seen in Figure 4. In most cases this prevents the need for virtual vertices (see below) and allows a faster calculation of the tangents and accordingly the normals.

Finally, we need the mid-vertices for the fast quality evaluation in screen space. As mentioned in

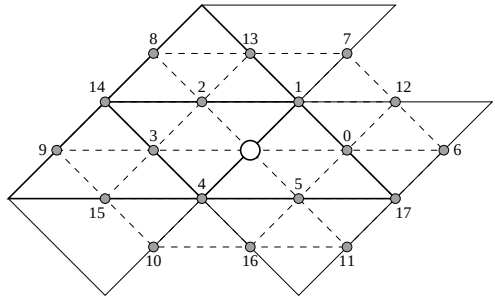


Figure 4: Butterfly tangent mask of the white point using precalculated mid-vertices

Section 3 we use this as an overestimated approximation of the expensive Hausdorff distance, which would be the exact error measure.

Another improvement to the original half-edge structure concerns to border vertices. Sometimes it can be difficult or impossible to find all edges belonging to a border vertex by simply moving along neighboring triangles and edges. A separate pointer to the incoming border edge solves this problem and results in a speedup.

Butterfly surfaces need the 1-ring of the vertices defining the edge to be split (see Figure 1). At borders it is not possible to find a 1-ring that fulfills the requirements for the subdivision mask because a border vertex does not have a valence of 6 in most cases. Therefore we have to mirror vertices from inside along the border to define virtual vertices outside so we regain a valence of 6. When the subdivision surface is embedded in a surrounding progressive mesh, it is possible and advisable to use neighboring vertices from the outer mesh since this will mostly result in a better quality than using virtual vertices.

A topological operator manages the reorganization of triangle and edge relationships. When a triangle needs to be subdivided, a dyadic split is performed, which divides this triangle into four triangles. The middle triangle becomes the new master triangle, which reuses the memory of the parent triangle. The surrounding three slave triangles are newly created. The master triangle saves the information about the subdivision depth or the level, respectively. The slave triangles do not need this information because they will either be of the same depth or they will get their own master tri-

angle defining their level of subdivision if they are subdivided further. If the difference of levels between neighboring triangles is greater than one, the coarser triangle needs to be subdivided too. In these cases the stored depth can be used for a fast comparison. When refining it is reasonable to first split all coarse triangles and subsequently the more and more refined ones. This leads to the following algorithm:

```
for all triangles T (increasing level)
{
  if needed then subdivide(T)
}

procedure subdivide(Triangle T)
{
  for all neighboring triangles N of T
  {
    if subd_level(N) < subd_level(T)
    {
      subdivide(N)
    }
  }
  dyadic_split(T)
}
```

However, before the refinement process is performed, the mesh is investigated in areas where it can be simplified again if the new point of view permits it. Simplification is only possible when all triangles in the neighborhood have the same or smaller subdivision level because otherwise the condition that only a difference of 1 level is allowed would be broken. If the triangles meet the condition, the information of the master triangle will be used to rebuild the old, one level coarser triangle and to undo the dyadic split. It makes sense to start with the finest triangles ascending to the coarsest triangles, resulting in the pseudocode:

```
for all mastertriangles TMid
                      (decreasing level)
{
  if quality is too good then
  {
    get_slaves T1,T2,T3 of TMid
    undivide(TMid,T1,T2,T3)
  }
}

undivide(Triangle TMid,T1,T2,T3)
{
  for all neighboring triangles N of
                           TMid,T1,T2,T3
  {
    if subd_level(N) > subd_level(TMid)
    {
      return
    }
  }
  undo_dyadic_split(T)
}
```

Finally the surface can be rendered. Although we used a red-green triangulation strategy, we allow differences in subdivision depth of level 1. This may result in gaps where different levels meet. But fortunately it is easy to fill these gaps with a final "just in time" subdivision which does not need to be stringently dyadic in analogy to a green split. Basically there are three cases which need a patch to avoid holes shown in Figure 5.

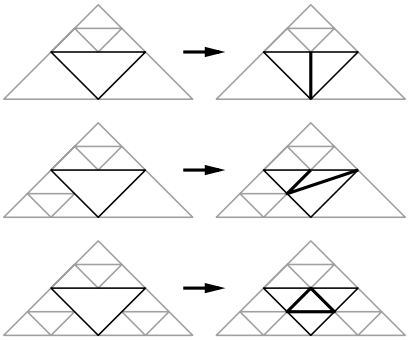


Figure 5: Temporary split to avoid cracks

The texture coordinates are calculated during the rendering process by a simple orthographic projection of the texture along the height axis onto the fine mesh. This is valid since terrain textures are gained mostly from pictures taken from vertically above by satellites or planes, which also corresponds to an approximately linear mapping function. Additionally textures may be used for hardware accelerated lighting calculations of small details [16].

5 Results

The frame rates of Table 1 were obtained on an Athlon K7C 1.33GHz with a GeforceII-MX graphics board by rotating the heightfield—which can be seen on the color plates in the appendix—around various axes and averaging the times.

When using uniform subdivision, the number of triangles is the same, for a viewport size of both 640×480 and 1280×960 pixels. The frame rates for higher resolutions are limited by the fill rate of the graphics board. The appearance does not improve very much at subdivision level 5 or higher, good results can be achieved at level 3 or 4.

Accordingly, the adaptive method with an error bound of 2 or 1 pixels yields the same quality but

viewport x × viewport y			640 × 480	1280 × 960
uniform	level of subdiv.	number of triangles	frames per second	
	0	128	300	125
	1	512	273	121
	2	2048	154	93
	3	8192	51	43
	4	32768	14	13
	5	131072	3.5	3.5
adaptive	error in pixels	number of triangles	frames per second	
	< 3	250-320 680-810	275	117
	< 2	440-540 1280-1400	273	108
	< 1	1300-1450 2800-3300	150	61

Table 1: Frame rates of terrain data starting with 8×8 facets

is 2.5 up to 10 times faster than the uniform division. Here the lower values in higher resolutions are caused by fill rate limits as well. Additionally there are more triangles to be rendered due to the dependency of the error limit on the viewport dimensions.

6 Conclusions and Future Work

In this paper we have presented a fast and efficient implementation of the butterfly subdivision algorithm. By using a modified directed edges data structure and by introducing mid-vertices, which are easier and faster to calculate on a coarser level, it is possible to achieve highly interactive frame rates. The compatibility with conventional techniques like progressive meshes alleviates the handling and allows the use of both methods at the same time. The field of application may be an improvement of traditional terrain rendering or LOD-rendering of CAD models described as subdivision surfaces.

In the future we would like to study if an implementation via hardware accelerated vertex programming is possible. This would result in an enormous speed burst on contemporary PC graphics hardware.

7 Acknowledgement

The implementation and ideas are based on the diploma thesis [13] of one of the authors and we would like to thank Rüdiger Westermann for valuable hints during this work.

References

[1] Randolph E. Bank, Andrew H. Sherman, and Alan Weiser, "Refinement algorithms and data structures for regular local mesh refinement", *Scientific Computing*, R. Stepleman, ed., North Holland, Amsterdam, pp. 3–17, 1983.

[2] Jürgen Bey, "Tetrahedral Grid Refinement", *Computing*, 55, pp. 355–378, 1995.

[3] Swen Campagna, *"Polygonreduktion zur effizienten Speicherung, Übertragung und Darstellung komplexer polygonaler Modelle", PhD-thesis*, Herbert Utz Verlag, 1999.

[4] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel, *"Directed Edges – A Scalable Representation for Triangle Meshes", Technical Report 2/1998*, Lehrstuhl für Graphische Datenverarbeitung, Universität Erlangen-Nürnberg, 1998.

[5] Edwin Catmull and James Clark, "Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes", *Computer Aided Design 10*, 6, pp. 350–355, 1978.

[6] Nira Dyn, David Levin, and John A. Gregory, "A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control", *ACM Transactions on Graphics 9*, 2, pp. 160–169, 1990.

[7] Nira Dyn, Sigalit Hed, and David Levin, "Subdivision Schemes for Surface Interpolation", *Workshop in Computational Geometry*, pp. 97–118, 1993.

[8] Michael Garland and Paul S. Heckbert, "Fast Polygonal Approximation of Terrains and Height Fields", *Technical Report CMU-CS-95-181*, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1995.

[9] Paul S. Heckbert and Michael Garland, "Survey of Polygonal Surface Simplification Algorithms", *SIGGRAPH '97*, 1997.

[10] Hugues Hoppe, "View-Dependent Refinement of Progressive Meshes", *SIGGRAPH '97 Pro-*

ceedings, pp. 189–198, 1997.

[11] Hugues Hoppe, "Smooth view-dependent level-of-detail control and its application to terrain rendering", *IEEE Visualization '98 Proceedings*, pp. 35–42, 1998.

[12] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, Werner Stuetzle, "Piecewise Smooth Surface Reconstruction", *SIGGRAPH '94 Proceedings*, pp. 295–302, 1994.

[13] Martin Kada, *"Unterteilungsverfahren zum beschleunigten Rendering von Oberflächen", Diplomarbeit*, Institut für Informatik, Universität Stuttgart, 2001.

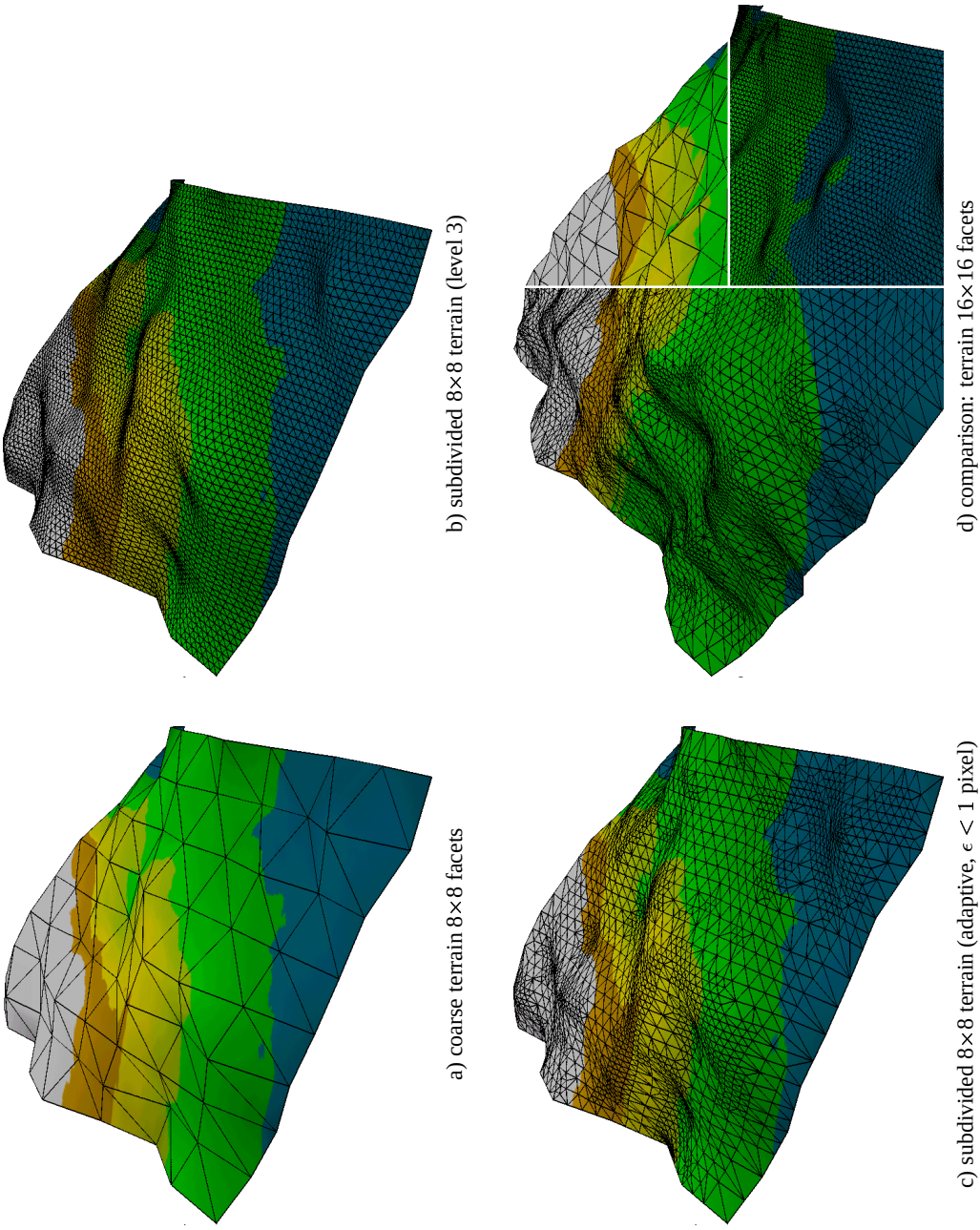
[14] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer, "Mesh Reduction with Error Control", *IEEE Visualization '96 Proceedings*, pp. 311–318, 1996.

[15] Stefan Roettger, Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel, "Real-Time Generation of Continuous Levels of Detail for Height Fields", *Procceedings of WSCG '98*, pp. 315–322, 1998.

[16] Dirc Rose and Thomas Ertl, "Rendering Details on Simplified Meshes by Texture Based Shading", *Workshop on Vision, Modelling, and Visualization VMV '00*, pp. 239–245. infix, 2000.

[17] Denis Zorin, *"Stationary Subdivision and Multiresolution Surface Representations", Ph.D. Thesis*, California Institute of Technology, 1998.

[18] Denis Zorin, Peter Schröder, and Wim Sweldens, "Interpolating Subdivision for Meshes with Arbitrary Topology", *SIGGRAPH '96 Proceedings*, pp. 189–192, 1996.


a) coarse terrain 8×8 facets
b) subdivided 8×8 terrain (level 3)
c) subdivided 8×8 terrain (adaptive, $\epsilon < 1$ pixel)
d) comparison: terrain 16×16 facets