Appearance-Preserving Terrain Simplification

Vladislav I. Suglobov
Lomonosov Moscow State University
Moscow, Russia

Abstract

This paper presents an efficient algorithm for piecewise-linear approximation of surfaces defined by two-dimensional discrete scalar fields (height map and color map). The presented algorithm takes into account information from color map to improve perceived approximation quality. The algorithm sacrifices the quality of geometry approximation for the sake of a better texturing of the simplified model thus increasing the overall perceived approximation quality.

Keywords: *terrain generation, terrain simplification, appearance-preserving simplification, texturing*

1. INTRODUCTION

The problem of piecewise-linear approximation of two-dimensional scalar fields or, in other words, the problem of triangulating a surface described with scalar height field arises in different applications of computer graphics, such as machine vision, cartography or computer entertainment. In computer entertainment, height fields can represent areas on which the screenplay of the game takes place. Such height fields can be a result of designers' work, either manual or automated. In any of the areas, in practical applications the height fields used tend to be of huge dimensions and thus require a sufficient storage. To make possible efficient processing, visualization or transmitting, the height field needs to be simplified. Piecewise-planar approximation is one of the existing approaches to the problem of height field and parametric surface simplification and allows straightforward rendering and efficient operation.

This paper presents an algorithm for creating piecewise-planar approximation of surfaces using not only the height field that defines the spatial shape of the surface but the color map of the surface as well. The presented algorithm exploits the fact that slight difference in the 3D shape of the surface model and the height field can be sacrificed to make possible the better texturing of the surface model and thus can increase perceived approximation quality. Taking into account the color map of the modeling surface is crucial in computer entertainment when creating a realistic-looking model of the surface.

2. RELATED WORK

Garland and Heckbert [1] give an extensive survey of different approaches to the problem of polygonal surface simplification and terrain generation algorithms that appeared since early 80s. The review covers a lot of work done in the field. However, none of the works mentioned in the review [1] address the problem of appearance-preserving simplification in whole. The algorithms surveyed approximate the surface position only, ignoring such attributes as surface color.

The efficient terrain generation algorithm proposed by Garland and Heckbert [2] was used in this work as a base algorithm for approximation of surface position. The algorithm belongs to the class of so called *refinement algorithms* and starts with the coarsest triangulation of the surface. Then, step by step, the algorithm inserts vertices into the triangulation, each time inserting a vertex where the absolute vertical deviation of approximation from the original surface is maximal over the whole triangulation. The triangulation algorithm exploited is called the *data-dependent triangulation*, as opposed to the Delaunay triangulation algorithm, and differs with the last one in what the optimal triangulation is considered to be. The data-dependent triangulation proved to produce better approximations using fewer vertices [2]. The algorithm of Garland and Heckbert [2] was chosen as a base for this work because of its efficiency and accuracy compared to other existing refinement algorithms.

However, in the recent years the *decimation* algorithms for appearance-preserving simplification appeared. The decimation algorithms for mesh simplification start with the most detailed model and gradually simplify it using a chain of simple operations like edge collapse. Hoppe's Progressive Mesh creation algorithm [4], that avoids collapsing edges that are incident to the triangles with different material can be applied to simplifying more complex surfaces than the terrain surfaces are. Cohen et al. [5] have presented an appearance-preserving simplification algorithm that makes use of texture and normal maps to increase the perceived quality of the approximated model. However, as decimation algorithms, these algorithms require the most detailed model of the surface to start and use simple per-edge or complex per-vertex error metrics. This makes them less efficient than the refinement algorithms that use simple per-vertex error metric and work starting from the simpler model to the more detailed one [2].

This work is done to fulfill the practical need for the refinement appearance-preserving terrain simplification algorithm.

3. PROBLEM

Given a need to efficiently render the resulting surface model on the present PC hardware and a need to represent a detailed surface to end user, the approach of using a color map of the surface as a single texture must be forgotten. For example, if one needs to represent a square area of the surface with dimensions of 10x10 kilometers, one will need a 200 MB 16-bit texture to only achieve a one texel per square meter accuracy. While the problem of the big texture size during the rendering process can be reduced using some texture-compressing and texture-caching techniques, the problem of manually creating such a texture is still a big one. It would require a great amount of time, computer resources, and the designer's sanity. The automated generation of such a texture from a set of smaller tiling textures would kill the benefits of using a one big texture and can be replaced with more memory-efficient techniques like the one described in this paper.

The approach used in the presented algorithm suggests creating a relatively small (256x256 texels) different texture tile for each type of the surface, like grass, sand, rocks, etc. Such tiles can be created by an artist or extracted from a color map if it is given as a photograph. Then every polygon of the surface model is mapped with the one of the created textures using some mapping technique, for example, the planar mapping.

However, due to the irregular nature of TIN (*Triangulated Irregular Networks*), the proper mapping would be a problem in low-detailed regions of the surface model. The relatively plain segment of the surface that can be accurately represented with only one polygon may have several types of the surface represented on it thus leading to a need of rendering this polygon using more than one texture or the custom texture. Rendering the polygon using more than one texture without creating custom textures can not produce detailed borderline between different textures, while creating custom texture with the detailed borderline for each of the polygons sufficiently increases memory budget.

Subdividing the polygon on a desired number of smaller polygons sufficient to represent the borderline with the needed accuracy can solve the problem. This solution only slightly increases memory budget by adding additional vertices and polygons to the surface model. The presented algorithm automatically generates a surface model using greedy insertion algorithm [2] and assures that borderlines between zones of different surface types will be present in the resulting model.

4. ZONE MAP

The zone map for each point of the surface tells what surface type or *zone* this point has. The example of the zone map is shown on Figure 1. The thin black lines represent the borderlines between different zones. Each non-black color used in the zone map represents a different surface type. The zone map needs not to be of very big size and can be easily created manually with the help of existing image processing software. This is what happened in practice. However, in the case if the aerial photograph or the drawing of the surface is provided, the image processing techniques like edge detection and quantizing can be used to turn it into the zone map.

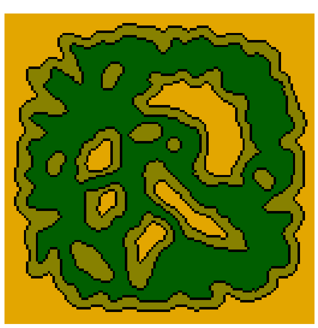

Figure 1: The zone map.

To incorporate the borderlines from the zone map into the surface model, the representation of these lines must be changed from raster to vector. Assuming that borderlines in the zone map are one pixel width, the simple algorithm can be used for this purpose. For each pair of the neighboring borderline pixels, two vertices are created at the centers of them and the directed edge is created on these vertices. The examples of the work of this process are shown in Figure 2.

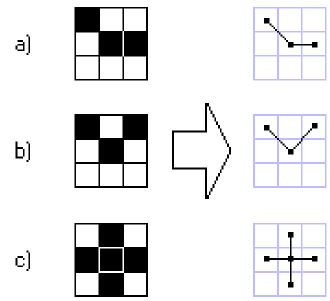

Figure 2: Creating a vector representation of the borderlines.

Each edge stores two numbers corresponding to the type of the surface on each side of the edge. The resulting set of vertices and edges is redundant and can be sufficiently optimized using the following rules:

1. The vertex is deleted from the set if its copy is present in the set.
2. If two edges incident with one vertex are collinear, the vertex and the edges are deleted and replaced with one edge.
3. The same as rule 2, but collinearity is replaced with the condition that the distance from the deleted vertex to the line connecting its neighbors is less than some *epsilon*.

Using the rules 1-2 eliminates the redundancy and using the rule 3 gives the ability to reduce the set of vertices and edges by changing the *epsilon*. The example of applying rules 1-2 is shown on Figure 3.

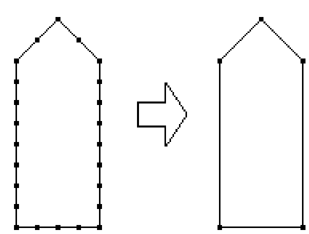

Figure 3: Simplifying the vector representation of borderlines.

The optional step can be performed after simplifying the vector representation of the borderlines. The borderlines can be doubled as shown on Figure 4 to produce a thin *transition zone* that can be rendered using two textures to create the effect of smooth transition of the texture on the one side of the borderline into the texture on the other side. In this step, new surface type is introduced between the pair of borderlines. The surface type numbers stored with edges are updated accordingly.

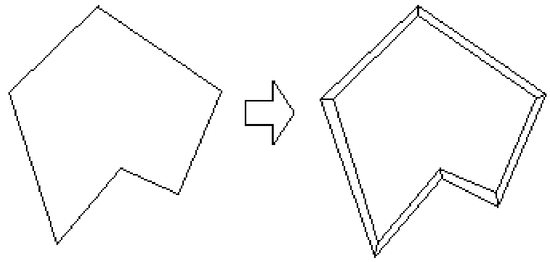

Figure 4: Doubling of the borderlines to create a thin texture transition zone.

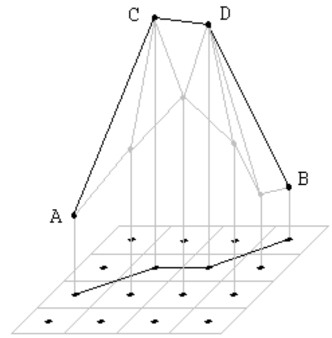

Figure 6: The eliminated slit.

5. TRIANGULATION

The set of vertices and edges representing the borderlines from the zone map are used to create the initial triangulation of the surface. The edges representing the borderlines are marked as constrained in this triangulation. The *constrained edge* can not be flipped during the re-triangulation process. Thus, the constrained edge existing in the initial triangulation will still appear (probably split into several edges) in the resulting surface model. The modified version of the greedy insertion algorithm [2] is used to refine the surface model to the needed approximation quality or vertices quantity. There were three modifications made to the refinement algorithm used. The first added a support for the constrained edges, making it possible to insert edges into the triangulation and prohibit flipping of them through the re-triangulation process that is performed after the insertion of a new vertex in the triangulation. The second modification corrected the *snap()* function behavior. The *snap()* function is responsible for splitting the constrained edge if a vertex is inserted laying close to it. This function was modified to eliminate the slits like the one shown on Figure 5 that appeared after the support for the constrained edges was added.

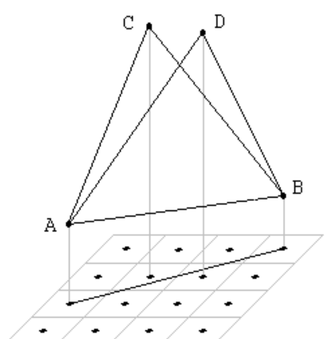

Figure 5: The slit ACBD caused by the constrained edge AB.

The corrected slit is shown on Figure 6. The modified *snap()* function do not pulls the inserted vertex to the split edge, but ties the split edge to the inserted vertex that remains in the node of the regular mesh.

The third modification was made to maintain the correct zone numbers on the both sides of the borderline edge during the split process. When the vertex inserted into the triangulation splits the constrained edge, the descendant two edges receive the same zone numbers as the original edge.

6. TEXTURING

After the refinement process completed the final texturing process takes place. This process is divided in two independent steps. The first step is assigning mapping coordinates to the vertices of the resulting polygonal model. This is done using well-known planar mapping technique. The second step is assigning textures to the polygons of the model. This is done using the surface type numbers of the constrained edges and a simple recursive filling algorithm that propagates the surface type number from the constrained edge to the triangles on one side of it through not constrained edges. The texture transition zone polygons receive two surface type numbers and are rendered using two textures to create the effect of the smooth transition of the texture on the one side of the transition zone to the texture on the other side.

7. RESULTS

The algorithm presented in this paper was implemented in software designed to generate 3D model of the terrain surface using its height field, zone map, and a set of textures. Algorithm was implemented using about 3500 lines of C++ code, 1500 of which were taken from [3]. The taken code was written by Dani Lischinski and is distributed under the GNU public license.

Appendix shows examples of the models generated using the presented algorithm. On the wire frame model, the texture transition zone can be distinguished that produces smooth texture transition visible on the solid model.

The research was performed and the software was written as a part of the “Iron Strategy” project developed by Nikita, Ltd. The software has been successfully used in preparing data for the needs of the project (interactive walkthroughs). In the process of the usage, the algorithm was tested on a great number of different height fields and zone maps showing a slight decrease in geometrical approximation quality in exchange to sufficiently increased perceived approximation quality.

8. REFERENCES

[1] Paul S. Heckbert and Michael Garland. *Survey of polygonal surface simplification algorithms. Multiresolution Surface Modeling Course (SIGGRAPH'97),* http://www.cs.cmu.edu/~ph

[2] Michael Garland and Paul S. Heckbert. *Fast polygonal approximation of terrain and height fields. Technical report, CS Dept., Carnegie Mellon U. Sept. 1995 CMU-CS-95-181*

[3] Dani Lischinski. *Constrained Delaunay triangulation implementation.* http://www.cs.huji.ac.il/~danix/

[4] Hugues Hoppe. *Progressive Meshes. Computer Graphics (SIGGRAPH'96 Proceedings) (1996), 98-108*

[5] Jonathan Cohen, Marc Olano, Donesh Manocha. *Appearance-Preserving Simplification. Computer Graphics (SIGGRAPH'98 Proceedings) (1998), 115-121*

About the author

Vladislav I. Suglobov is a graduate student of the Lomonosov Moscow State University and is currently working at Nikita, Ltd as a programmer.

Email: vlud@nikita.ru

Appendix