Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2008, Article ID 316790, 5 pages
doi:10.1155/2008/316790

Research Article

Efficient Terrain Triangulation and Modification Algorithms for Game Applications

Sundar Raman and Zheng Jianmin

School of Computer Engineering, Nanyang Technological University, Singapore 639798

Correspondence should be addressed to Sundar Raman, sund0010@ntu.edu.sg

Received 28 September 2007; Accepted 3 March 2008

Recommended by Kok Wai Wong

An efficient terrain generation algorithm is developed, based on constrained conforming Delaunay triangulation. The density of triangulation in different regions of a terrain is determined by its flatness, as seen from a height map, and a control map. Tracks and other objects found in a game world can be applied over the terrain using the "stenciling" and "stitching" algorithms. Using user controlled parameters, varying levels of detail can be preserved when applying these objects over the terrain as well. The algorithms have been incorporated into 3dsMax as plugins, and the experimental results demonstrate the usefulness and efficiency of the developed algorithms.

1. INTRODUCTION

The generation of a terrain is fundamental to many games today. Games like Tread marks [1] and Colin McRae Rally [2] were bestsellers mainly due to their impressive looking terrains, and by using technologies like dynamic LOD, continuous LOD, and deformable terrain over it. While all these are useful in solving specific problems related to rendering a huge terrain, they do not deal with the generation of a simple, yet realistic-looking terrain.

A terrain can be decomposed into two parts: (i) a triangular mesh, and (ii) one or more textures. In using such a terrain in a game, there are two main concerns: (i) the memory needed to store the terrain information for large worlds, and (ii) the time taken to load the big terrain into memory during a game's load-time. In this paper, the focus is on the triangular mesh part and algorithms are developed for its generation and modification.

Our terrain-generation algorithm is top-down, similar to Garland's approach [3], and it reduces the number of triangles required to represent the terrain for any particular level of detail, thus solving the two problems. The main difference between our algorithm and Garland's algorithm(s) is the introduction of a control map and several user parameters for refinement and minimization of triangle count. Our modification algorithm is based more on the real-world needs of a game artist, and uses the same triangulation technique to modify parts of the terrain. We have extended our previous work [4] to make it faster and more controllable.

The rest of the paper is organized as follows. Section 2 describes how to generate efficient triangular meshes for terrains using a height map, control map, and Delaunay triangulation. In Section 3, algorithms are proposed to easily modify the mesh so that objects found in a game, like tracks and guard rails can be applied over it. Two ways of doing this are discussed: (i) stenciling and (ii) stitching. The results for each section are shown separately. Finally the paper is concluded in Section 4.

2. TERRAIN GENERATION

The inputs of our algorithm are a height map, a control map and user defined parameters. A controlled, constrained conforming Delaunay triangulation algorithm is then used to create a triangular mesh that defines the terrain. Below is a description of each component of the procedure.

2.1. Height map

A *height map* is used to store elevation data for the terrain. The height map is a grayscale image where a white pixel

indicates the highest point, a black one the lowest point and shades of gray for heights in between. Each pixel is usually made up of 8 or 16 bits, thereby allowing us to store 2^8 or 2^{16} different-height values. Figure 2(a) shows a height map of a simple terrain and Figure 3(a) shows its corresponding triangular mesh representation.

2.2. Control map

Besides the height map, a *control map* is introduced, indicatingthe *regions in the terrain where more detail needs to be added*. The control map is also a grayscale image, where whiter regions are given more importance than the darker regions (refer to Figure 2(b)). Demarcating control regions has many uses, such as densely triangulating only those areas where the camera focuses often, like the area where we have tracks. Such an example is shown in Figure 6.

2.3. Delaunay triangulation

A *Delaunay triangulation* (DT) of a set of points P in a plane is a triangulation DT(P) in which no point $p \in P$ lies inside the circumcircle of any triangle in DT(P). The advantage of DT is that it produces a more *regular*-looking triangulation for any given point set, by maximizing the minimum angle of the output triangle set.

A planar straight line graph (PSLG), which is a collection of vertices and edges where endpoints of each edge are vertices of the PSLG, is used to initially represent the terrain. This is because the terrain needs to have well-marked boundary edges. Since the input is a PSLG, its triangulation becomes a *constrained Delaunay triangulation* (CDT). We also need to add more vertices on the inside and on each edge of the PSLG, making it a *constrained conforming Delaunay triangulation* (CCDT). The additional vertices inserted are called Steiner points [5].

The most common algorithms for DT are the *divide-and-conquer* [6], *sweepline* [7] and *incremental* [8] algorithms. Of these, Guibas and Stolfi's [9] implementation of divide-and-conquer algorithm has been found to be the fastest (see Su and Scot Drysdale [10]). Shewchuk [11] has adopted an optimization to this algorithm from Dwyer [12] to partition the vertices with horizontal and vertical cuts, thereby making it faster. Hence the same has been used in our implementation.

For constructing the CDT, first, the DT of the vertices is first performed; then, each missing input segment is forced into it by deleting the edges it crosses, inserting the segment and then retriangulating the two resulting polygons using the *ear-cutting algorithm* [13]. The CCDT is constructed using a variation of Chew's second algorithm [14], optimized for terrain generation using some assumptions (only boundary edges are constrained, etc.).

2.4. Terrain generation algorithm

The workflow of the proposed CCDT algorithm is shown in Figure 1. The algorithm generates the final mesh using a top-down approach. It begins with the four corner vertices of the height map, which determine the terrain's size. A rectangular PSLG is constructed, and an initial CDT simply splits the rectangle into two triangles.

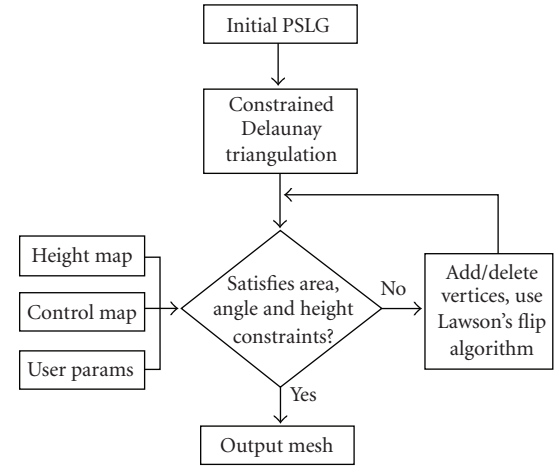
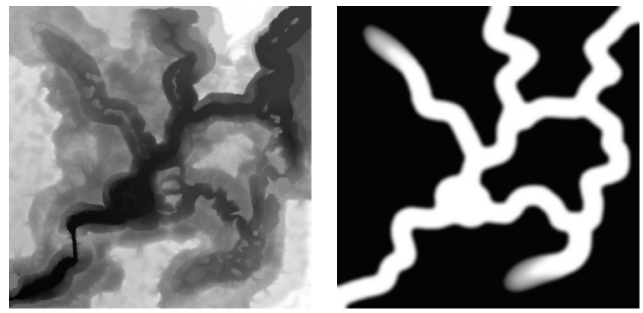


Figure 1: Proposed terrain-generation algorithm.

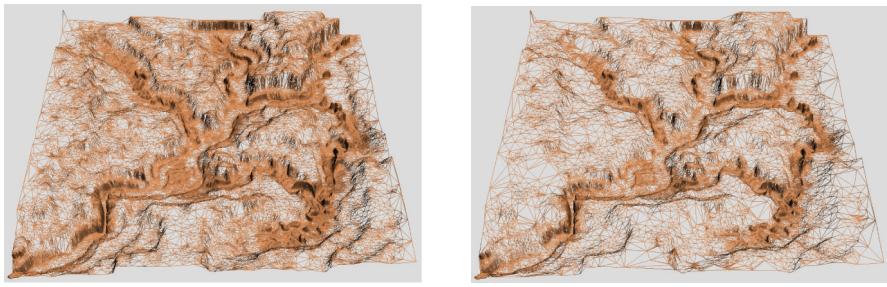


(a) (b)

Figure 2: Inputs to the terrain-generation algorithm (a) height map (b) control map.

Each triangle is checked to see if it satisfies the MinArea and MinAngle constraints. If it does not, it is rejected; else the more complex height constraint check is performed.

For the height constraint check, 1–40 equally distributed pixels are chosen inside each triangle, the exact number depending on the SkipScale parameter and the size of the current triangle. Next, all the pixels are checked to see if they satisfy the height constraint by comparing them with the height map and control map. If the control map is not specified, a default all-white or all-black control map can be assumed (we use all-black). If all the pixels satisfy the constraint, the triangle is accepted; otherwise, it is rejected. The rejected triangles are eliminated using the variation of Chew's second algorithm [14], by adding and/or deleting Steiner points, and maintaining the Delaunay condition using Lawson's *flip algorithm* [8]. The newly created triangles are checked as well, until all the triangles thus formed satisfy all three constraints.



(a) (b)

Figure 3: Output terrain mesh (a) hi-res with 175 K triangles, (b) low-res with just 75 K triangles.

2.5. User parameters

The user parameters shown in Figure 1 control the results of the output terrain. Eight parameters are introduced: SizeX, SizeY, Height, MinArea, MinError, MaxError, MinAngle, and SkipScale.

SizeX and SizeY specify the length and width of the terrain. Height is the maximum height of the terrain (all the height values from the height map will be mapped from 0 to this value), MinArea is the minimum area of any triangle in the projected 2D mesh. The height error in the control region is guaranteed to be less than or equal to MinError, whereas for the other regions, it is guaranteed to be not greater than MaxError. The algorithm further guarantees that the angle of any triangle found by projecting the mesh onto 2-dimension is greater than or equal to MinAngle. Ruppert [15] has shown that the upper limit for MinAngle is 20.7°, but Shewchuk [11] has argued that in practice, adding new vertices fails only when the angle exceeds 33.8°. To be safe, we have put an upper limit of 33° for MinAngle. SkipScale determines the overall accuracy of the algorithm.

These parameters, in addition to the control map, give a lot of flexibility to our triangulation algorithm, allowing users to generate different parts of the terrain at different resolutions.

2.6. Results

The algorithm was implemented as a 3dsMax plugin to generate the terrain. Figures 2(a) and 2(b) show the input height map and control map, respectively. Our algorithm generates a terrain with such a property (P) that the control regions and those parts where there are a lot of height variations are tessellated more densely, whereas fewer triangles are used for the relatively flat and noncontrol regions.

The terrain in Figure 3(a) was generated with the following parameters: SizeX = SizeY = 15840, Height = 2000, MinArea = 10, MinError = 5, MaxError = 50, MinAngle = 0, SkipScale = 1. Furthermore, by modifying the error parameters, a terrain of any resolution can be generated, with reduced number of triangles, while maintaining the property P at the same time.

For the terrain in Figure 3(b), MinError and MaxError were set to 10 and 100, respectively, with all the other parameters being the same. As we can see, this terrain looks reasonably detailed in the "nonflat" and control regions, but the other areas are greatly simplified to produce a mesh with 58% reduction in the number of triangles. This gives great flexibility to the artist who can generate multiple versions of the same terrain (using different control maps), which can be used in the game under different rendering contexts.

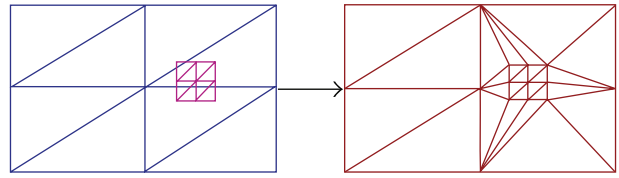


Figure 4: Stenciling a plane over a bigger plane.

3. TERRAIN MODIFICATION

After generating the Delaunay triangulated terrain, artists or programmers may need to modify it to overlay patches, tracks, and so forth, over it. This section describes two methods (stenciling and stitching) for such modification. Terminology for the sake of clarity: DM is the destination mesh SM is the source mesh. Our objective is to overlay SM over DM, at any specified position and orientation.

3.1. Stenciling

In *stenciling*, priority is given to the geometry of SM when overlaying it on DM. The amount of priority given is determined by an error parameter. To illustrate, a simple example as in Figure 4.

Let DM and SM be two simple 2×2 grids, with SM being smaller and positioned over DM as shown in Figure 4 (left). On the right, there is a single stenciled mesh in which

(i) all the edges and vertices of SM are projected and retained;
(ii) no new vertices are introduced;
(iii) the edges of DM which intersect with any edge of SM are removed;
(iv) the region around the stenciled part is retriangulated.

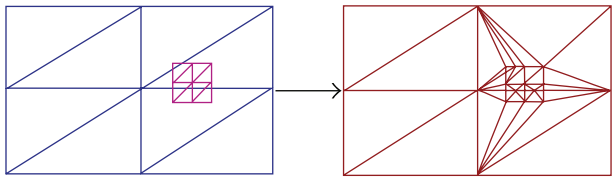


Figure 5: Stitching a plane over a bigger plane.

It is notable that SM is projected over DM "as-is." However, if DM is a terrain and SM is a track, this has disadvantages because we will lose height information pertaining to DM when vertices of DM inside the projected area are removed. Therefore, an error parameter is introduced, which controls how much height difference error is allowed over the stenciled region. This allows great flexibility to the artists who can make the track as detailed as they want to. A real life terrain-track example is shown in Section 3.3.

The main steps of the algorithm are as follows:

Stenciling algorithm

(1) Create a new empty mesh **M**
(2) Create a new PSLG **X**, initialize it to DM
(3) Generate height map **H** from SM
(4) Add projected SM vertices lying within DM to **X**
(5) Remove DM vertices lying within projected SM region.
(6) Remove all DM edges which intersect with projected SM edges, from **X**
(7) If SM extends beyond DM, calculate points of intersection of projected SM edges with bounding edges of DM, and add them to **X**
(8) Add all projected "full" SM edges (excluding those which lie fully or partially outside DM) to **X**
(9) Retriangulate
    (i) Apply the CCDT algorithm, set **M** = CCDT(**X**).
    (ii) Perform area, angle, and height constraint checks only for newly formed triangles lying within projected SM region
(10) For every new point (not belonging to DM) added to **M**, calculate its new height from **H** (from step 3)

For checking constraints, parameters can be set from the user interface, similar to terrain generation.

3.2. Stitching

Stitching is the "error-free" and more intuitive way of overlaying SM over DM. True to the word, the two meshes are simply "stitched" together, while following the height of DM. To illustrate, the same example as in Figure 4:

As compared to stenciling, the right of Figure 5 shows a mesh in which

(i) all the projected edges of SM are either retained or split;
(ii) all the edges of DM are also retained or split;
(iii) new vertices are created wherever DM and SM intersect;
(iv) the region around the stitched part is retriangulated.

The advantage of stitching is that no "new" vertices are introduced by the triangulation algorithm, except for the newly created intersection points. This implies that the height difference error is always zero, and we get a perfect stitched mesh. It also means there is no need of any user parameters for angle, area, and height error constraints, since no more triangles than necessary are created.

The stitching algorithm is outlined below:

Stitching algorithm

(1) Create a new empty mesh **M**
(2) Create a new PSLG **X**, initialize it to vertices of DM
(3) Add all nonduplicated vertices of SM lying within DM ($\mathbf{V_S}$), to **X**
(4) If SM extends beyond DM, calculate points of intersection of projected SM edges with bounding edges of DM ($\mathbf{V_B}$), and add them to **X**
(5) Add new vertices created due to intersection of every projected SM edge with every DM edge ($\mathbf{V_N}$), to **X**
(6) Add all unaffected edges of DM (not intersecting with any projected SM edge), to **X**
(7) Add all unaffected projected edges of SM (not intersecting with any DM edge) lying within DM, to **X**
(8) Add newly formed edges found by splitting DM edges to **X**
(9) Add newly formed edges found by splitting projected SM edges to **X**
(10) Retriangulate
    (i) Apply the CDT algorithm, set **M** = CDT(**X**)
(11) Calculate precise height of all new vertices ($\mathbf{V_S UV_B UV_N}$) by point-inside-triangle tests and interpolation

3.3. Results

Many experiments were conducted to test the algorithms, with terrains of different sizes as destination meshes and objects like tracks and patches representing plough lands as source meshes. Figure 6 shows a track object positioned over a terrain, at the exact orientation in which it needs to be applied. Figure 7 shows the effect of stenciling and stitching of the track over this large terrain.

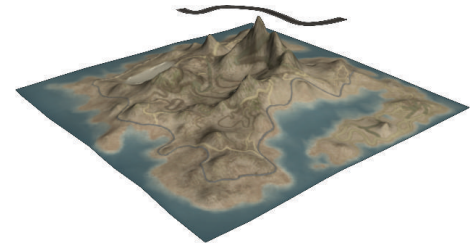


Figure 6: Input terrain and track meshes.

Figure 7: Track over terrain using stenciling.

Figure 8: Track over terrain using stitching.

It can be seen from Figure 7 that in stenciling, as few vertices are added "inside" the projected source mesh, the exact number controlled by an error parameter. For stitching, however, all the points of intersection are added, and hence there is no loss of height information in the final mesh (Figure 8). In other words, the track mesh perfectly follows the terrain and its height.

4. CONCLUSION

In this paper, an efficient terrain-generation method was introduced, based on the constrained conforming Delaunay triangulation, using a height map, control map, and some control parameters.

By using a height map together with a control map, generating a terrain using a top-down approach was automated to a large extent, providing game artists the means to create a terrain at different resolutions in different regions. Enhancements to the mesh were done via user parameters which specified angle, area, and height constraints. The nature of the chosen triangulation algorithm (Delaunay triangulation) further guaranteed the maximization of the minimum angle, hence producing a well rounded, more geometrically balanced terrain.

Two methods were proposed to modify the generated terrain: stenciling and stitching, to add objects such as tracks over the terrain. In stenciling, the geometry of the source mesh was retained as closely as possible, and the CCDT algorithm was used for retriangulation. In stitching, the source mesh followed the destination mesh perfectly, by calculating all points of intersection with it, and the CDT algorithm was used for retriangulation.

ACKNOWLEDGMENTS

The authors wish to express their sincere thanks to gameLAB Annexe, NTU, Singapore and TQ Global, Singapore, for providing them with excellent infrastructure and an ideal work environment for their research.

REFERENCES

[1] Tread Marks, Longbow Digital Arts Incorporated, 1999, http://www.ldagames.com/treadmarks/.

[2] Colin McRae Rally 04, The Codemasters Software Company Limited, 2004, http://www.codemasters.com/games/?gameid=1361.

[3] M. J. Garland and P. S. Heckbert, "Fast polygonal approximation of terrains and height fields," Tech. Rep. CMU-CS-95-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pa, USA, 1995.

[4] S. Raman and J. Zheng, "Efficient terrain triangulation and merging of world objects for game applications," in *Proceedings of the 3rd International Conference on Games Research and Development (CyberGames '07)*, pp. 46–51, Manchester, UK, September 2007.

[5] J. Steiner, "Questions proposées. Théorèmes sur l'hexagramum mysticum," *Annals of Mathematics*, vol. 18, pp. 339–340, 1827.

[6] D. T. Lee and B. J. Schachter, "Two algorithms for constructing a Delaunay triangulation," *International Journal of Parallel Programming*, vol. 9, no. 3, pp. 219–242, 1980.

[7] S. Fortune, "A sweepline algorithm for Voronoi diagrams," *Algorithmica*, vol. 2, no. 1, pp. 153–174, 1987.

[8] C. L. Lawson, "Software for C^1 Surface Interpolation," in *Mathematical Software III*, pp. 161–194, Academic Press, New York, NY, USA, 1977.

[9] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams," *ACM Transactions on Graphics*, vol. 4, no. 2, pp. 74–123, 1985.

[10] P. Su and R. L. Scot Drysdale, "A comparison of sequential Delaunay triangulation algorithms," in *Proceedings of the 11th Annual ACM Symposium on Computational Geometry (SCG '95)*, pp. 61–70, Vancouver, BC, Canada, June 1995.

[11] J. R. Shewchuk, "Triangle: engineering a 2D quality mesh generator and Delaunay triangulator," in *Proceedings of the 1st ACM Workshop on Applied Computational Geometry (WACG '96)*, pp. 124–133, Philadelphia, Pa, USA, May 1996.

[12] R. A. Dwyer, "A faster divide-and-conquer algorithm for constructing delaunay triangulations," *Algorithmica*, vol. 2, no. 1, pp. 137–151, 1987.

[13] H. Elgindy, H. Everett, and G. Toussaint, "Slicing an ear using prune-and-search," *Pattern Recognition*, vol. 14, no. 9, pp. 719–722, 1993.

[14] L. P. Chew, "Guaranteed-quality mesh generation for curved surfaces," in *Proceedings of the 9th Annual Symposium on Computational Geometry (SCG '93)*, pp. 274–280, San Diego, Calif, USA, May 1993.

[15] J. Ruppert, "A Delaunay refinement algorithm for quality 2-dimensional mesh generation," *Journal of Algorithms*, vol. 18, no. 3, pp. 548–585, 1995.