Generation and Rendering of Virtual Terrain

Harald Ammeter, Dr. Mikhail V. Mikhailyuk

Institute of Microprocessor Computer Systems RAS

Moscow, Russia

Abstract

This paper presents a computer program that displays a virtual landscape consisting of flatlands, mountains, hills, lakes, and sky as a pilot would see it while flying on low height over the surface. It highlights the terrain generation and rendering algorithms of the implemented program.

Keywords: *Terrain Generation, Real-time Rendering, Level of Detail Optimization, Flight Simulation.*

1. INTRODUCTION

The purpose of the implemented software is the development of techniques that are required in simulators for the training of pilots. Whereas many commercially available products are expensive and often require sophisticated hardware, this simulator is intended to run on different platforms, for instance on ordinary personal computers. It therefore complies with the "OpenGL" graphics standard, and in order to reduce dependency on geographical databases requiring large amounts of storage space, the simulator is able to combine real geographical data with randomly generated terrain. Its main features are the ability to create fairly realistically looking terrain and the optimization of the real time rendering algorithm to work on limited graphic resources.

2. TERRAIN STRUCTURE

In order to simulate the movement of the viewer, the program has to render frames displaying the pilot's view from continuously changing positions. The viewer's position for every frame is calculated using the current speed value and direction and the difference between the current time and the time of the last update.

Every time the current position (determined by speed and current time) or angle (modified by user keyboard interaction) of the viewer changes, the area covered by the view has to be adapted. Parts of the viewed area disappear, and new parts enter into view. The whole viewed terrain consists of single fixed-sized square blocks. Therefore, squares entering into view have to be generated and squares dropping out of view have to be removed to save memory and avoid unnecessary rendering calls. The square blocks covering the current view area are stored in a list that is updated with every new frame.

From the viewer's position and angle, the 4 corner coordinates of the visible terrain trapezoid are calculated. The far end of the trapezoid is then limited to a reasonable distance. In order to hide this artificial terrain border from the viewer, fog effects are used during rendering. The calculated trapezoid area is then filled with the necessary number of square terrain blocks.

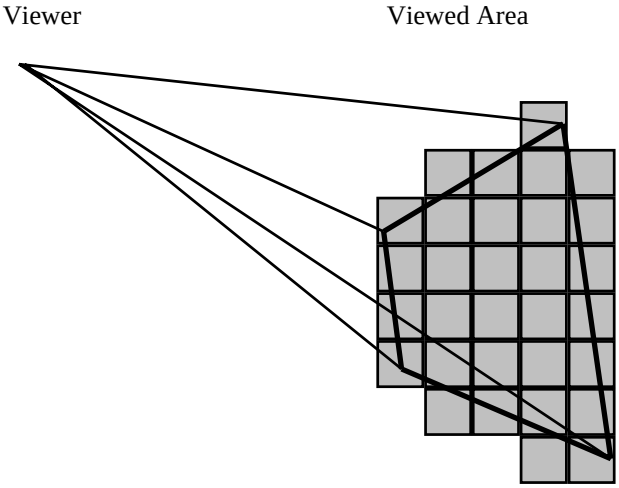

Figure 1: The dynamic list of square blocks filling the current field of view.

3. RANDOM TERRAIN GENERATION

The whole visible terrain consists of fixed-sized squares, each of which is created as an entity. Every square block contains a regular grid of height values, stored in a two-dimensional array. The characteristics of the new terrain – mountains, water, and roughness – are determined by a set of parameters that can be changed by the user at runtime. As it enters into view, a new terrain square is added to the existing list of squares. Its borders are fitted to the borders of already existing neighbor squares, and the rest of the array is then filled with random height values, using the currently active parameters.

New height values are calculated by the method of recursive subdivision: First, the corners of the height array are determined (level 0). During the next step (level 1), the midpoints between every pair of corners are set by linear interpolation of the two corner heights and adding or subtracting from this mean height a random displacement value. During consecutive steps, the midpoints between already determined points are set, until all points have been

processed. Since the distance between processed points is divided into half with every new level of detail, the displacement range of the midpoint is reduced with every step. The algorithm thus requires that the dimensions of the square array equal a power of 2.

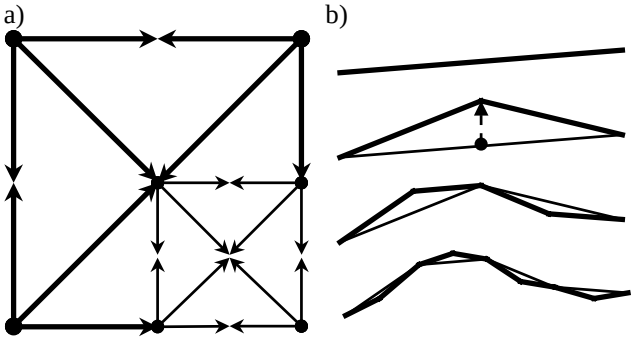

Figure 2: (a) The first two recursive steps of determining height values between already calculated points. (b) How recursive midpoint displacement transforms a straight line into a smooth surface.

Height values can be positive or negative. At rendering time, points below sea level are covered with water at height 0.

The height value for one new point is calculated using the following formula:

$$H = A + \frac{(R - W) * (1 + A)^2 * M}{L^E}$$

where

H is the resulting new height value

A is the interpolated average height of the neighbor points calculated during the previous step

R is a random value between 0 and 1

L is the level of recursion, starting from 0

W, M, E are variable parameters

Three parameters determine the characteristics of new terrain:

- M: Maximum height. M determines the range of new height values. M > 0.
- E: Level Exponent. E determines terrain roughness. New height values between two near points are displaced less than new values between two more distant points. E determines by how much this maximal displacement is reduced for consecutive steps: a value of 2 means reduction by half with every step; smaller values result in rougher terrain. E > 1.
- W: Water. W determines the amount of lakes in the terrain. A height value proportional to W is subtracted from the displacement to obtain negative values covered by water at sea level (height 0). $0 \le W \le 1$.

The interpolated height of the neighbor points is used as a base value. A random value is added. To obtain values below sea level, Parameter W is subtracted. The random value R (between 0 and 1) is multiplied by parameter M in order to reach the desired maximal height. For every level, i.e. as the distance between the already existing neighbor points is divided into half, the displacement is divided by parameter E. In order to obtain rougher terrain in the mountains and smoother plains in the lower regions, higher values must be modified to a larger extent; this effect is obtained by multiplying the displacement by a value corresponding to the interpolated average height A.

A square in the middle of already generated terrain shares the points of its 4 borders with neighbor squares to the left, right, bottom and top. In addition to that, the 4 corners are shared with 4 diagonal neighbors. Whenever a new square has to be generated, the list of already existing squares is first searched for these 8 neighbors. After its allocation in memory, corner and border points of the new square are first of all preset with corresponding points. Only points that have not already been set are then calculated using the random generator.

4. OPTIMIZATION AND RENDERING

The optimization algorithm explained in this chapter basically corresponds to those described in [2] and (to a smaller extent) in [1].

Rendering the generated height fields requires transforming them into connected triangles forming the terrain surface. In order to speed up rendering performance, the triangles of every square are connected to one single strip – a list of triangle vertices, where every vertex occurs only once – and stored in a pre-compiled command structure that can be executed several times from changing viewer positions, a so-called display list. During the triangulation process, one display list per square is generated, already existing display lists of the previous frame are updated, and obsolete squares dropped out of view are deleted. At rendering time, all these display lists are executed in turn.

The rendering speed and thus the achievable frame rate depend heavily on the number of rendered triangles. Therefore, during triangulation their quantity is reduced by an optimization algorithm that removes those details, which do not considerably contribute to image quality. Whenever possible, two small triangles located next to each other are joined to form a larger one. The same process is recursively repeated with the resulting triangle until no further optimization can be made without substantial loss in image quality.

Two optimization criteria are applied. The first one is based on the horizontal dimension of the rendered triangles,

without considering the height values of the vertices. In a square block far away from the viewer, the distances between points of the height grid are smaller than the screen resolution. This means that the smallest triangles of the grid may all be joined with their neighbors to form larger ones, and the same test may be applied to these larger triangles again recursively. Since the height array is a regular grid, the projected horizontal dimensions of all triangles of a certain level within a square block are approximately equal. So one can from the block's distance to the viewer quickly determine a "level of detail" for every block, indicating how many times the process of joining small triangles to larger ones may be repeated without notable image difference.

The second criterion is applied to the remaining triangles. But this time, triangles are joined to larger ones if they approximately lie in a plane, which is determined by looking at the height values of the vertices. For every triangle pair, the algorithm considers the vertical amount by which the common vertex moves after the join. If the distance is small enough, the join is performed. This optimization step depends not only on the horizontal distance, but also on the height distance of the block and its terrain roughness.

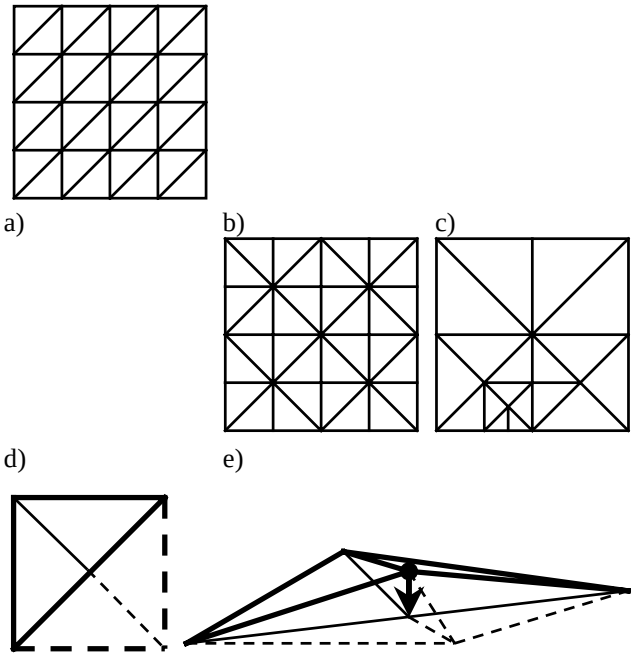

Figure 3: (a) Straightforward triangulation, but without possibility of joining triangles. (b) Hierarchical formation of triangles allows optimization by joining small triangles to larger ones. (c) Possible formation after optimization. (d,e) Two triangle pairs must be joined simultaneously in order not to produce any gaps.

While joining triangles along block borders, the algorithm has to ensure that on both sides of the common border joins occur simultaneously. Otherwise, inconsistent borderlines would be produced. Therefore every block has to be treated together with its neighbor blocks.

The rendering algorithm draws every square block by connecting all triangles created during optimization to one continuous triangle strip. Since a block after optimization may contain triangles of different sizes, these have to be processed in hierarchical order. The rendering algorithm starts with the four quadrants of the block and divides each of these triangles into two. Each of the resulting triangles is again recursively divided, until the size is reached that has been determined by optimization, upon which the triangle is rendered.

The triangle area is filled by a texture pattern. In the current implementation, one single black/white texture bitmap is repeatedly applied in all directions. The height and local inclination of the terrain determine the color of a triangle vertex: points located above a limit and parts of steep mountains are colored like rocks, the rest like grass. To achieve a realistic shading effect, a light source is positioned above the terrain, and the exposition of every triangle vertex is determined by the average of the surrounding triangles' normal vectors.

In the end, water and sky are rendered. The water surface consists of a blue textured rectangular plane that is drawn at sea level. Terrain points with negative height are covered by the water. Sky consists of another rectangular plane, drawn at a constant height above the viewer.

5. RESULTS

Achieved frame rates depend on terrain type, image resolution, and rendering hardware. On a 300 MHz Pentium II, 64 MB RAM computer with NVIDIA RIVA TNT, 16 MB RAM graphics adapter, the following typical frame rates have been obtained for a full screen image:

Mountain terrain:	14 f/s
Hills:	18 f/s
Plains:	24 f/s

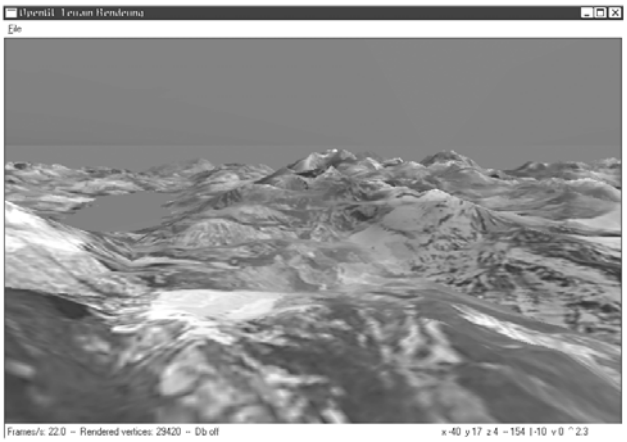

Figure 4: Sample screen display of the program, showing mountain regions.

6. ACKNOWLEDGEMENTS

This work has been made possible by cooperation between the Institute of Microprocessor Computer Systems RAS, Moscow, and SATW Swiss Academy of Technical Sciences, Zurich, Switzerland. It is the result of an eight months' exchange program by SATW.

7. REFERENCES

[1] Peter Lindstrom, David Koller, Larry F. Hodges, William Ribarsky, Nick Faust, Gregory Turner. *Level-of-Detail Management for Real-time Rendering of Phototextured Terrain.* Georgia Institute of Technology, 1995.

[2] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, Gregory A. Turner. *Real-time, Continuous Level of Detail Rendering of Height Fields.* Georgia Institute of Technology, 1996.

Authors:

Harald Ammeter, software engineer in an exchange program by SATW, Switzerland, and

Dr. Mikhail V. Mikhailyuk, chief of department at the Institute of Microprocessor Computer Systems RAS

Address:
36-1, Nakhimovski pr., Moscow, 117872, GSP7, Russia

Telephone: (095) 332 49 66
E-mail: mix@mail.ivvs.ru