Procedural Terrain Generation at GPU Level with Marching Cubes

Bruno José Dembogurski
MediaLab-UFF

Esteban W. Gonzalez Clua
MediaLab-UFF

Marcelo Bernardes Vieira
GCG-UFJF

Fabiana Rodrigues Leta
LMDC-UFF

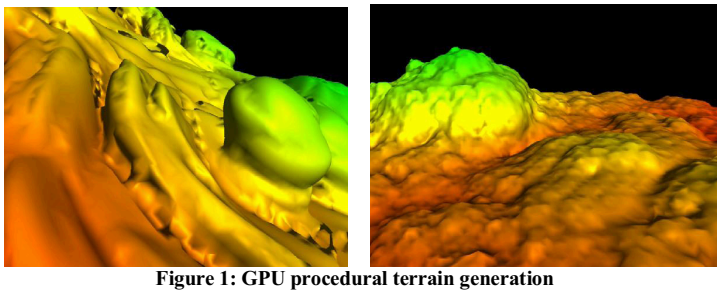

Figure 1: GPU procedural terrain generation

Abstract

This work presents a procedural terrain generation using the recent Marching Cubes Histogram Pyramids (also known as HPmarcher) implementation. Perlin Noise function is used to procedurally create the terrain. It runs entirely on the Graphics Processing Unit of Shader Model 3.0 and 4.0 graphics hardware.

Keywords: Marching Cubes, GPU, Procedural Generation, Terrains, Histogram Pyramids.

Authors' contact:
{bdembogurski,esteban}@ic.uff.br
marcelo.bernardes@ufjf.edu.br
fabiana@lmdc.uff.br

1. Introduction

Traditionally procedural terrains have been limited to height fields that are generated by the CPU and rendered by the GPU. However the serial processing architecture of the CPU is not suited to generating complex terrains which is a highly parallel task, Geiss [2007]. Another issue with height fields is the lack of interesting features such as caves and overhangs.

The GPU, in the other hand, is an excellent option for this kind of processing since it is a highly parallel device. GPUs of graphics cards are designed for large computational tasks with large requirement for memory bandwidth, based in massive parallelism.

Terrain creation is becoming the most costly element in video games and simulators, due the player high expectations for realistic virtual words. Procedural generation can create seamless terrains with a realistic and natural look with a nice level of customization, without the need of huge amounts of disk space and loading time.

A voxel representation of the terrain allows much more features such as natural caves, tunnels, overhangs, cliffs without stretched walls and also allows a dynamic environment in a real time application.

This work presents a procedural terrain generation implementation using Histogram Pyramids Marching Cubes approach producing interesting terrains with a considerable amount of triangles achieving interactive frame rates.

2. Related Work

In the last years, GPU iso-surface extraction algorithms have been a topic of extensive research. In that field procedural terrain generation is also a well studied topic.

The main issue is that volumetric data consumes memory quite rapidly and the visualization of an extremely large and complex terrain can be a very challenging task. Another point to consider is the control over the terrain due the pseudo-random nature of this approach. Creating a specific type of terrain is a hard job usually requiring some additional technique to obtain the desired results.

Recently Geiss [2007] presented a good approach called *Cascades* to generate and visualize volumetric complex terrains using shader model 4.0 graphics hardware and new DirectX 10 capabilities such as the *geometry shader* (GS), stream output and rendering to 3D textures. It was able to create interesting and customizable terrains and also adding some nice features like particle system for waterfalls. However, this implementation has the

limitation of running only under Windows Vista since it is based on the DirectX 10 API.

The first use of the Histogram Pyramids algorithm running entirely on the GPU was presented by Zigler *et al.* [2007] for stream compaction in a GPU point listing generation algorithm.

Later Dyken and Ziegler [2007] presented a Marching Cubes approach extending the Histogram Pyramids structure to allow stream expansion, which provides an efficient method for generating geometry directly on the GPU. Currently, this approach outperforms all other GPU-based iso-surface extraction algorithms.

3. Marching Cubes

The Marching Cubes algorithm (MC) proposed by Lorensen and Cline [1987] is the most known and used algorithm for extracting an iso-surface from a scalar field.

Basically, in the MC algorithm, from a 3D grid of $N \times M \times L$ of scalar values a grid of $(N-1) \times (M-1) \times (L-1)$ cubic MC cells (voxels) is created. Each cell has a scalar value associated with each of its eight corners. The idea of the algorithm is "march" through all the cells, producing a set of triangles that approximates the iso-surface of that cell.

The topology of the iso-surface is defined by the classification of the eight corners of the MC cell as inside or outside the surface (being inside represented as "1" and outside as "0"). If all values in the cell corners are the same it is possible to discard that voxel, since it will be totally inside or outside the surface. If the corners have different values then the cell intersects the iso-surface.

According to that it is possible to determinate the voxel class using the following notation:

$$C^{i,j,k} = P_0, 2P_1, 4P_2, 8P_3, 16P_4, 32P_5, 64P_6, 128P_7$$

Suppose a corner $P_5^{i,j,k}$ is inside the iso-surface and all others are outside, according to the binary notation we will have the following MC class:

$$(P_0^{i,j,k},..., P_7^{i,j,k}) = (0, 0, 0, 1, 0, 0, 0, 0),$$

That corresponds to the class 8 of a total of 256 possible classes, which can be reduced to 14 patterns due symmetry [Lorensen and Cline 1987]. The class also determinates which of the twelve edges are piercing the surface. If one end-point of the edge is inside the iso-surface and the other end-point is outside, that edge is intersecting the surface. For each of the 256 classes there is a corresponding triangulation of the edge intersections.

The position in the edge where the iso-surface intersects is obtained approximating the scalar field along the edge using a linear polynomial and finding the zero-crossing of that polynomial.

As presented in [Dyken and Ziegler 2007] the Marching Cube algorithm is implemented as a sequence of data stream compaction and expansion operations. The first stream operation determines the class of each voxel and check the triangulation table in order to obtain the number of triangles produced by each voxel. Discarding voxels that doesn't produce geometry generate the stream compaction. Each element in the output stream will be expanded according to the number of triangles determined by the triangulation table. The iso-surface is formed by connecting edge intersections on each element to form a set of triangles.

4. Histogram Pyramids

The core of the Marching Cubes implementation in this paper is the Histogram Pyramids algorithm [Dyken and Ziegler 2007], which is used to compact and expand data streams on the GPU. The structure of the Histogram Pyramid is identical to a MipMap where each level is a quarter size of the previous level pyramid but instead of taking the average of the elements to build the higher level, the algorithm sum the values.

In the input, the 3D voxel domain is mapped in the 2D domain (a large tiled 2D texture where each tile represents a slice of the voxel volume) to index a sequence of texture coordinates, which is known as a Flat 3D layout [Harris 2003]. The base layer have the number of elements to be allocated in the output stream, the rest of the pyramid is constructed following the MipMap reduction idea but adding the values of the 2 x 2 block in the texture. In the end, the top element will have the length of the output sequence.

All elements are extracted one by one descending into the sub-pyramids and inspecting the 4 elements in the corresponding 2 x 2 block until the base layer is reached. The stream compaction and expansion is relative to the number of elements that the input allocates in the output. For example, if one input element produces 4 elements in the output a stream expansion occurs. Otherwise, a stream compaction is performed if the input element produces no element in the output or a stream pass-through is performed when one input element produce one element in the output sequence.

5. Terrain Generation

Conceptually, the terrain surface can be described in a implicit form f: $R^3 \rightarrow R$. For any point given in the 3D space this function returns a single floating point value. Those vary over the space from positive and negative values where positive means outside and negative means inside the surface. Thus, one may choose a constant c for which the locus of points $f(p) = c$ form an isosurface defining the terrain.

The most common function used to create procedural terrains is the Perlin Noise function, which generates a natural look with no perceptive pattern.

The noise generation approach used in this work is similar to the one presented by Green [6]. This method implements the noise directly in the pixel shader instead of using pre-computed 3D textures. Green also uses a new interpolation function which is a Hermite polynomial of degree five resulting in a C^2 continuous noise function. It is also possible to use the original interpolation function which is less expensive to evaluate but has discontinuous second derivatives.

This approach has several advantages like less texture memory required and a higher quality interpolation. But the main enhancement in this approach is that it has a larger period, meaning that the noise patterns do not repeat often. This is essential for a natural and realistic look of the terrain.

Since the terrain can have an arbitrary size, it needs to be created in several passes because volumetric data consume memory really fast (e.g.: 256^3 voxels require a GeForce 8800GTX). The idea is to tile the scene using several cubes adding the appropriate coordinate's transformations when sampling the scalar field and extracting the geometry.

Considering a static terrain, it is possible to store the geometry in a buffer, since it is not need to sample and extract the geometry every frame. Using the Shader Model 4.0 hardware this is quite straightforward using the new *transform feedback* mechanism, which records vertex attributes of the primitives processed. The selected attributes can be written with each attribute in a separate buffer object or with all attributes interleaved into a single buffer object. If a geometry shader or program is active, the primitives recorded are those emitted by the geometry program. Otherwise, it will capture the primitives whose vertices are transformed by a vertex program or shader [5].

Basically, the general idea is to create a large buffer and incrementally fill this with the iso-surface each tile of the scene.

6. Results

All tests in this work were performed on a Nvidia GeForce 8800 GTS with 512mb ram on a Windows XP SP2 computer with an AMD64 2500+ CPU at 2.2GHz and 2GB of ram using the 175.16 version of the ForceWare (display driver).

The terrains in Figs. 2, 3 and 4 were generated with 127^3 voxels, 8 octaves for the turbulence function, a 2.0 lacunarity and a 0.5 gain for each iteration. These values can be changed for different results like a higher number of mountains or a more regular terrain. This experiment provided real time results with about 40.0 fps.

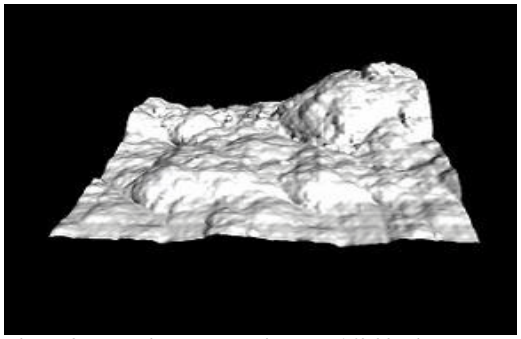

Figure 2: Terrain example with over 160.00 triangles and over 2 million voxels at a 38.2 fps.

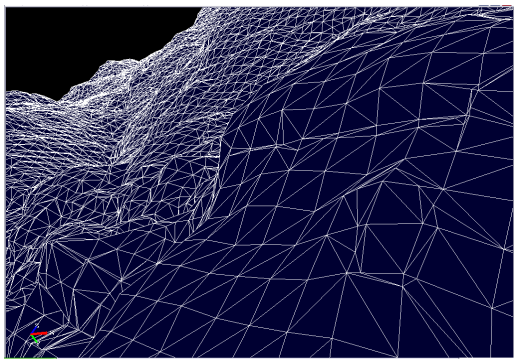

Figure 3: Wireframe representation of the same terrain.

It is also possible to manipulate the terrain creating flat spots in some areas (e.g. to support some building or any kind of construction in a game). For this, Geiss [2007] present a method to replace the density function with a flat spot within a linear radius of some point center. Other approaches like using a hand-painted texture and warping the coordinates to break the homogeneity of the terrain when using many octaves are possible ways to customize the terrain.

The Fig. 4 shows a terrain using a coordinate warping before applying the noise function. The result is an alien/organic look, showing the wide range of customization possibilities for this approach.

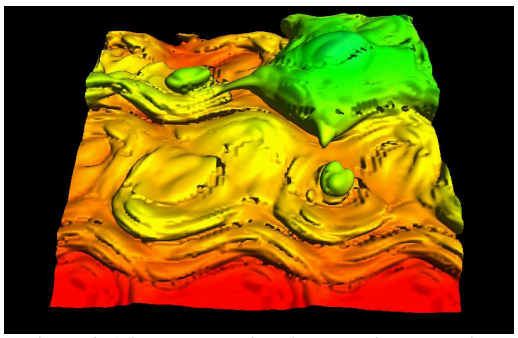

Figure 4: Alien look terrain using coordinate warping before applying the noise function.

Other volume dimensions were also evaluated for performance purposes: 31^3, 63^3 and 255^3. The relation between volume dimensions and the respective FPS values are shown in Fig 5. The frame rate depends on the number of triangles obtained the terrain.

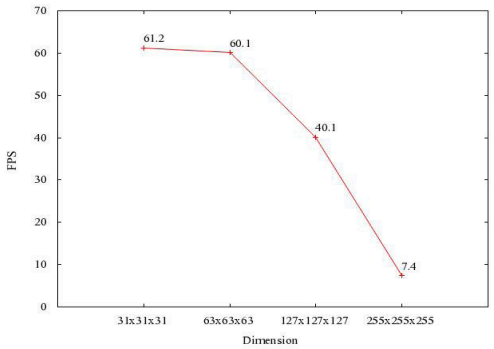

Figure 5: Dimension x FPS comparison

The FPS is related to the number of triangles presented by the terrain with a higher dimension volume, the FPS drop is expected since the number of triangles is about 5 times higher in each version. This graph shows that even with an extremely large number of polygons the visualization is possible and a very detailed terrain is shown in iterative frame rates. The low fps presented at 255^3 is related to the number of triangles used in this massive representation (in this case over 600.000 triangles were used).

The number of marching cubes cells processed per second is also a good way to evaluate the algorithm effectiveness (Fig. 6). One may see that even with cubic time/space complexity, the histogram pyramid approach is a powerful isosurface extraction algorithm. It performs fast with larger datasets and is well suited for this non restrictive terrain representation.

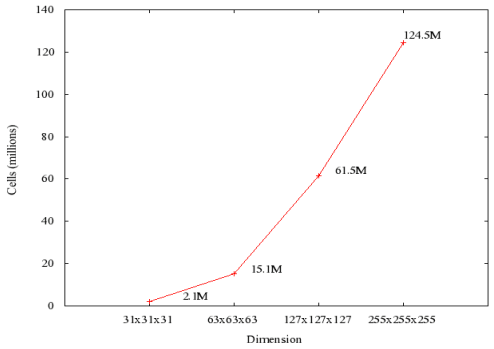

Figure 6: Dimension x Cells processed relation

7. Conclusion and Future Work

This work presented a fast procedural terrain generation using the Histogram Pyramids Marching Cubes approach. The terrain is generated with a Perlin noise function to create a scalar field. This strategy provided good results with interactive frame rates and customizable features.

The performance of the presented approach was evaluated in function of grid dimensions. The results are promising since the implementation achieved a real time frame rate with 127^3 voxels.

As a future work, we intend to extend the Histogram Pyramids approach with a CUDA framework. The voxel classification and the pyramid construction pass can be enhanced because CUDA supports 2D buffer texture sampler.

We also intend to improve the terrain with texturing, shading and some kind of erosion (thermal or hydraulic). Texturing is a challenge to procedural generations due to its arbitrary topology but one solution is to apply three different planar projections one along each primary axes with some blending between areas [Geiss 2007].

References

EBERT D.S *et all," Texturing & Modeling. A Procedural Approach",* 3rd ed, Morgan Kaufmann Publishers, 2003.

DYKEN, C., ZIEGLER, G. "High-speed Marching Cubes using Histogram Pyramids". Proceedings of EUROGRAFICS 2007, Volume 26, Number 3.

GEISS, R. "Generating Complex Procedural Terrains Using the GPU". GPU Gems 3. Chapter 1, pp. 7-37. 1st Ed. 2007.

GREEN, S., "Implementing Improved Perlin Noise", GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation., 2005, pgs 409-415.

HARRIS M. J., III W. V. B., SCHEUERMANN T., LASTRA A.: Simulation of cloud dynamics on graphics hardware. Proceedings of Graphics Hardware 2003.

LORENSEN W., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. Computer Graphics (SIGGRAPH 87 Proceedings) 21, 4 (1987), 163–170.

OLSEN, J. "Real-time Procedural Terrain Generation: Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games", IMADA University of Southern Denmark, 2004.

ZIEGLER G., TEVS A., TEHOBALT C., SEIDEL H.-P.: "GPU Point List Generation through Histogram Pyramids". Tech. Rep. MPI-I-2006-4-002, Max- Planck-Institut für Informatik, 2006.

OLSEN, J. "Real-time Procedural Terrain Generation: Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games", IMADA University of Southern Denmark, 2004.