Rendering Photorealistic Mountain Terrain

USING PERLIN NOISE HEIGHT MAP, INTELLIGENT MULTI-TEXTURING & DIRECTIONAL LIGHTING

http://karteekkumarm.wix.com/terrain

[12]

Karteek Kumar Mekala | Masters Project | April 1, 2014

Abstract

Whether you are driving a tank through a war zone or watching a plane fly across Colorado, a common scene in many video games and animated movies is that of a beautiful mountain terrain. The primary goal of this project is to render a 3D scene of photorealistic mountain terrain that is vast and can be navigated using a fly through camera. Performance statistics gathered from the working demonstration are expected to prove that the implemented techniques are both performant and scalable. The techniques used to implement the solution are intended to be intuitive and options in the working demo will allow the user to understand the contribution of each stage in isolation. This will allow these techniques to be understood before being adapted into larger projects. The solution will also be made customizable to assist the same.

To render the scene - we first generate a map of heights. Using these heights we generate a list of triangles that can be rendered as a wire-mesh of the terrain. Multiple layers of grass, rock and water textures are applied to these triangles intelligently to mimic the look of real terrain. Lighting is applied, a skybox is rendered and a fly-through camera is provided to navigate through the scene.

Table of Contents

Introduction

Photorealistic simulation of mountain terrain forms the basis of many beautiful outdoor scenes in video games and animated movies. However, the application of mountain terrain rendering is more than just being used to show vibrant landscape. It forms the foundation over which many virtual environments are built upon. This ranges from depictions of outdoor farms to city experiences. Therefore it becomes essential for these programs to be performant along with being able to yield photorealistic and beautiful scenes. While rendering in real-time, it is even more important to maintain a high frame-rate. This project intends to use techniques that are intuitive and result in the targeted photorealistic image while still being scalable and performant. This is also made possible by making use of the high power GPU's and CPU's along with the advanced shader pipeline provided with the latest graphic development tools.

The techniques used to implement the solution are intended to be intuitive and options in the working demo will allow the user to understand the contribution of each stage in isolation. This will allow these techniques to be understood before being adapted into larger projects. The solution will also be made customizable to assist the same.

Background

A lot of work has been done in terrain rendering due to its vast applications in movies, video games and digital art. Various techniques can be applied at different stages of the implementation.

Historically, many fractal algorithms have been used to implement terrain. In 1979-1980 John Carpenter presented a video accompanying his SIGHPRAGPH paper – "Computer Rendering of Fractal Curves and Surfaces". This paper introduced techniques to synthesize fractal geometry and apply it to rendering. The following image is from his movie, demonstrating its application to render fractal terrain.
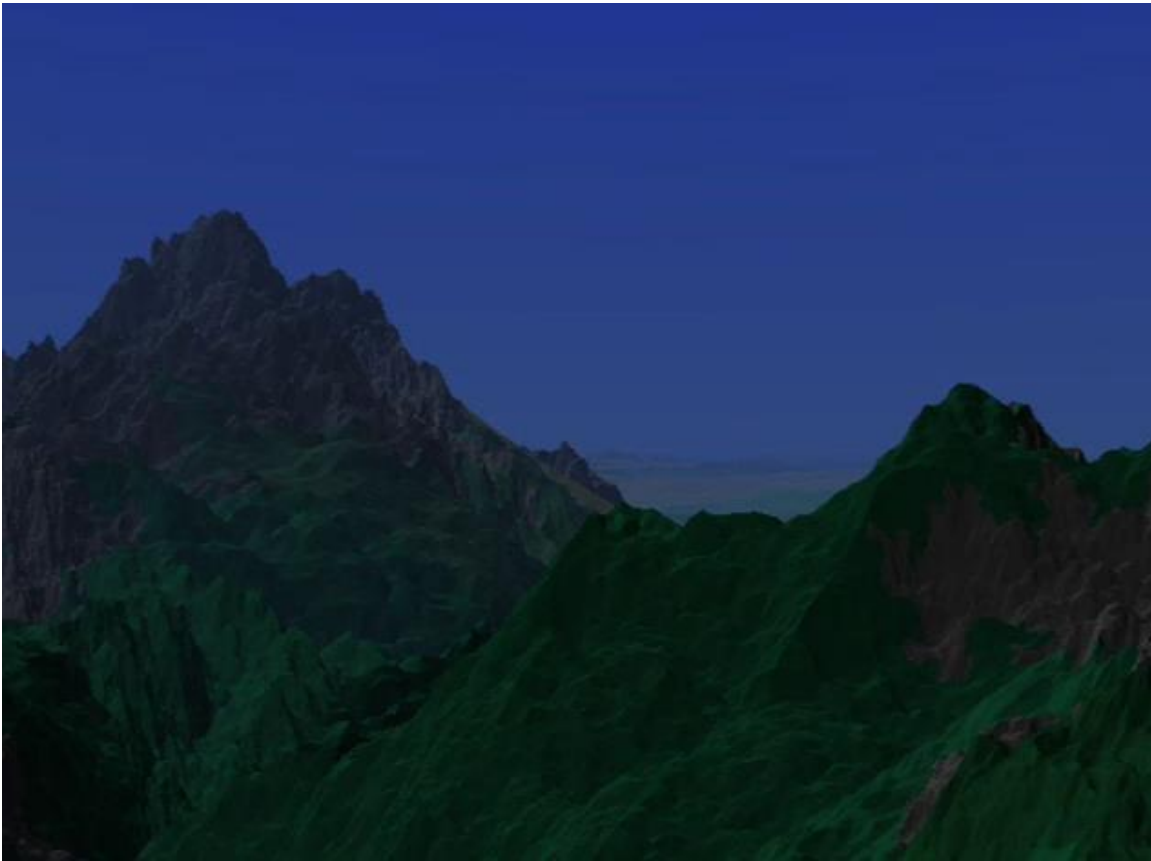
[1]

The fractal technique employed here to construct the terrain is a subdivision technique. We start with a fractal curve composed of two end points and a roughness factor to be used as an offset. We then calculate a midpoint by choosing a constrained random process. The calculated midpoint then forms a common endpoint to the two resulting fractal sub-curves. This process of subdivision is repeated to arrive at the required terrain height map. The choice of the roughness factor and the random process determines characteristics of the resulting terrain.[1]

A common variant of the above technique is the 'Diamond Square' recursive subdivision algorithm. This technique starts with height values for the four corners of a rectangle. A random value within a fixed range is added to the average of these four values. This value is applied to the center of the rectangle, therefore subdividing the rectangle into four smaller diamonds. Applying the same process to each diamond results in squares again. This recursive subdivision is continued to generate an array of height maps. [6]

In our implementation we use the Perlin noise generator to generate a 2 dimensional array of heights. The Perlin noise generation technique was introduced by K Perlin. The general idea of the proposed algorithm was to a produce pseudo random signal over 3 dimensions that seems like it has been run through a low-pass filter – that removes high spatial frequencies. A 2 dimensional snap shot of the generated noise can be encoded and saved as a resource file that the terrain rendering program can consume. [4]

Once the height map is generated, various tessellation techniques can be applied to each primitive provided by the height map generator. Using sophisticated techniques can result in more natural looking terrain with roughness and bumps. Less sophisticated techniques will result in a more flat appearance. Lighting the terrain scene by itself can be a challenging task and various models of lighting can be implemented. Depending on the scene requirement - directional lights, spot lights, light sources and shadows be implemented. For the purpose of this project a simple directional light is implemented using the Phong Illumination model.

Once the wire mesh of the terrain is available, we must apply color to the scene. This can be performed either using procedural techniques or texture mapping. Procedural techniques calculate the color of each pixel on the terrain procedurally using various mathematical models. Texture mapping techniques use pictures of real terrain elements and blend them together.

Many 3d modeling tools are now available that allow artists to author beautiful looking terrain that can then be used as part of a scene or a video game. While advances in graphics hardware and software have made it possible to achieve beautiful real-time rendering of terrain using such modeling tools, it is still a challenging task for a vast terrain to be authored by an artist. The techniques introduced by this technique are intuitive and can be easily customized and adopted into other projects. It is also a performant and scalable solution to ensure that resources are available to create more complex and interactive environments over the terrain.

Approach

SCENE DESCRIPTION

The goal of this project is to render a photorealistic mountain terrain scene with grass, rock and water. The distribution of rock and grass on the terrain along with the implementation of directional lighting will mimic what is expected in real mountain terrain. A skybox is rendered around the mountain terrain to contribute to the photorealism of the scene. A fly through camera is provided with user controls to navigate through the scene. The real-time performance statistics will be available by the use of debug controls given on the keyboard. To understand the techniques applied at each stage in isolation, the scene can be rendered as a wire-mesh, lighting can be disabled and other parameters can be adjusted by the use of the debug keys.

TECHNIQUES

The following techniques will be used in their respective stages of the program implementation. The first step is to generate a grid of heights for which we use the Perlin noise generation algorithm. These heights are converted into a list of triangles in the tessellation stage. The normals at each vertex are then calculated at each vertex by applying simple vector geometry. These normals are used in the lighting and texturing calculation. The lighting and texture calculations use techniques that work for achieving a photorealistic look for the scene. A fly-through camera is implemented using vector geometry and a simply skybox is rendered.

Height Map

The height map resource file is generated by storing the output of the Perlin noise algorithm in the form of a grey scale bitmap. The resource file is put through a smoothing face to avoid sharp transitions in height values. The height map encoded into the grey scale image is saved as a bitmap file for the terrain rendering program to read.

Tessellation

The height map needs to be converted into a set of triangles that can be rendered. We start with a flat M * N grid of rectangles in the XZ plane. The rectangles formed by the grid are divided into triangles which are the primitives for rendering. It can be observed that each vertex is shared by 6 adjacent triangles. To avoid duplicate vertex data, indices of the vertices forming the triangles are stored for rendering.

The values from the height map are then applied to the vertices in the grid. Rendering the triangles now results in the wire-mesh image of the mountain terrain.

Calculating Normals

For a triangle formed by 3 points a,b and c – the normal can be calculated using the following cross product:

$$n = (b - a) \, X \, (c - a)$$

For each quad in the grid, the normal is calculated for the upper right vertex as the average of the normals of the two triangles that form the quad. The second pass is a smoothening phase. For every vertex the normal is calculated as the average of the normal from all its neighboring vertices along with itself.

Lighting

The normal calculated at each vertex is used to implement lighting at the pixel shader. The program sets the directional light parameters. This includes the ambient light, the diffuse light intensity and the diffuse light direction. The pixel shader will use the Phong illumination model to calculate the intensity of each pixel on the terrain.

Multi-Texturing

To perform the texturing of the terrain, we try to mimic what we observe while looking at images of real mountain terrain. A noticeable observation is that areas of the terrain that have a high slope appear to be rocky and areas that are more flat appear to be relatively grassier. We use this observation in our technique of texturing by applying a higher fraction of rocky texture to areas of the terrain with higher slope while applying higher fraction of grass texture to areas of the terrain with lower slope.

This slope factor can be easily calculated for each vertex in the fragment shader by taking the dot product of the up vector with the normal of the triangle being textured.

$$S = Up \ dot \ N \ where,$$
$$S \text{ - is the slope factor being calculated}$$

Up - is the Up vector (0,1,0)
N – is the normal calculated for the vertex

We then maintain a sea level and render a flat water texture for areas of the terrain that are at a height below the sea level. We perform additional calculation and add a random factor to make the transition between the rock/grass texture to water texture seem more natural.

The multi-texturing along with lighting gives us the photorealistic mountain terrain that we target.

Skybox

The skybox is implemented by rendering a cube centered around the camera position. Tile-able sky textures are rendered onto the 6 interiors faces of the cube. As the camera moves, the cube moves along and continues to maintain the camera position at its center. This gives the player a feel that the sky is at an infinite distance.

Camera Controls

Simple camera controls are implemented to move the camera around in the 3d world. This is implemented by maintaining 3 values for the camera:

1. Eye position (e) – position of the camera
2. Look-at position (p) – the position in the 3d world that the camera is focused at
3. Up vector (u) – the vector that is up relative to the view direction

Using the values, additional values can be calculated:

1. View direction (v) = (p-e)
2. Right vector (r) = u X v

Using these values, the following operations can be performed:

1. **Move forward/back**: To move the camera forward/back we move the eye position and the look-at position along the View direction vector. We make sure the distance between the eye position and the look at position is preserved.
2. **Yaw**: To implement yaw, we simply rotate the look-at position around the axis defined by the eye position and the up vector. We make sure the distance between the eye position and the look at position is preserved.
3. **Pitch**: To implement pitch, we rotate the look-at position around the axis defined by the eye position and the right vector. Again, we make sure the distance between the eye position and the look at position is preserved.

PERFORMANCE STATISTICS

A separate window is rendered with performance statistics that are obtained in real-time from the scene rendering. This data is collected from profiling code embedded in the program. Performance statistics recorded include current FPS (frames per second), average FPS, total number of triangles drawn, memory usage and CPU time for the overall program and for different stages of the mountain terrain rendering, as discussed above.

POTENTIAL CHALLENGES

Sharp Transitions

By directly applying the techniques discussed above, we expect to see sharp transitions between vertices with respective to lighting and texturing. To prevent this, we will apply smoothing passes across the data sets. For the height map, we apply a smoothing phase where the height of each vertex is influenced by the heights of its neighbors. Shared normal will be calculated for each vertex making the lighting and texturing transitions across vertices more natural.

Sea level

Maintaining a flat sea level across the terrain map might look at the transitions between land and water. To make this look more natural, we will try to implement a texturing blending technique between land and water. We will also need to work around a threshold sea-level at transitions, so it does not look the height at which the water appears is exactly the same across the terrain.

Clipping

Since no form of collision detection is implemented, it is possible that the user flies through the terrain. To prevent this we could perform simple calculations at each user move to find and prevent clipping through the terrain without implementing a full blown collision detection system.

Skybox

Since the skybox is a fixed size cube around the camera position, it would break at the corners of the terrain because the terrain itself it finite. Choosing a skybox whose bottom face blends easily into the water texture could help make this less noticeable. Another potential solution to this is to avoid the camera from

Technology

Development System Specifications

1. Operating System - Windows 8.1 Pro 64 bit
2. 16 GB RAM
3. CPU – Intel® Core™ i5-3570K @ 3.40GHz
4. GPU – NVIDIA GeForce GTX 770 2GB VRAM

Software Development Kits

1. Windows 8.1 SDK
2. Direct X 11 SDK
3. DirectXTK

Development Environment

1. Visual Studio 2013
2. Github
3. GitExtensions

Conclusion

The multi-texturing technique introduced in this project is intuitive and easy to work with because it mimics the way real terrain looks. The final texturing of the terrain can be adjusted elaborately by tuning the blending of rock, grass and water textures. This technique is scalable and additional layers of texture such as snow or soil can be added easily. This technique will also allows us to use large high quality texture images because each texture is applied onto a region of the terrain matching its dimension and maintaining its detail.

This project also demonstrates that the proposed multi-texturing technique can easily be adopted along with some of the popular techniques. It also demonstrates that the combination of techniques described can be used to render photo-realistic mountain terrain while also allowing it be customized easily using the different parameters exposed. The performance statistics to be collected under different parameters are expected to prove that the solution is efficient and scales easily with the scene requirements.

The techniques used to implement the solution are intended to be intuitive and options in the working demo will allow the user to understand the contribution of each stage in isolation. This will allow these techniques to be understood before being adapted into larger projects. The solution will also be made customizable to assist the same.

Deliverables

1. Height map generation program
2. Working demo of the described Mountain Terrain scene
3. Multiple screen shots taken from the demo
4. A final report including the performance statistics observed using different scene-parameters
5. A presentation for the final project defense

Schedule

Target Date	Actual Date	Event	Status
18/04/2014	18/04/2014	Project Proposal	In-progress
21/04/2014		Project Website	In-progress
28/04/2014		Working demo prototype	In-progress
12/05/2014		Working demo	
20/05/2014		Final Report	
30/05/2014		Project Defense	

References

1. **Computer Rendering of Fractal Curves and Surfaces**
   Loren C. Carpenter
   Boeing Computer Services
   Seattle, Washington
2. **An Image Synthesizer**
   Ken Perlin
   Courant Institute of Mathematical Sciences
   New York University
3. **Improving Noise**
   Ken Perlin
   Media Research Laboratory
   Dept. Of Computer Science
   New York University
4. **Implementing Improved Perlin Noise**
   http://http.developer.nvidia.com/GPUGems/gpugems_ch05.html
5. **The Ultimate DirectX Tutorial**
   http://www.directxtutorial.com/Lesson.aspx?lessonid=11-1-3
6. **Generating Random Fractal Terrain**
   http://gameprogrammer.com/fractal.html
7. **Multi-textured Terrain in OpenGL**
   http://3dgep.com/?p=1116
8. **Introduction to OpenGL for Game Programmers**
   http://3dgep.com/?p=636
9. **Skybox tutorial**
   http://sidvind.com/wiki/Skybox_tutorial
10. **Multi Texturing in OpenGL**
    http://berkelium.com/OpenGL/GDC99/multitexture.html
11. **Image gallery of Kodai Kanal**
    http://filmlocationmanagerkannan.com/kodaikanal.html

Author Information

/*--*/

Karteek Kumar Mekala

Masters of Science – Computer Science
Rochester Institute of Technology
Karteek.Kumar.M@gmail.com
kkm6815@rit.edu

/*--*/