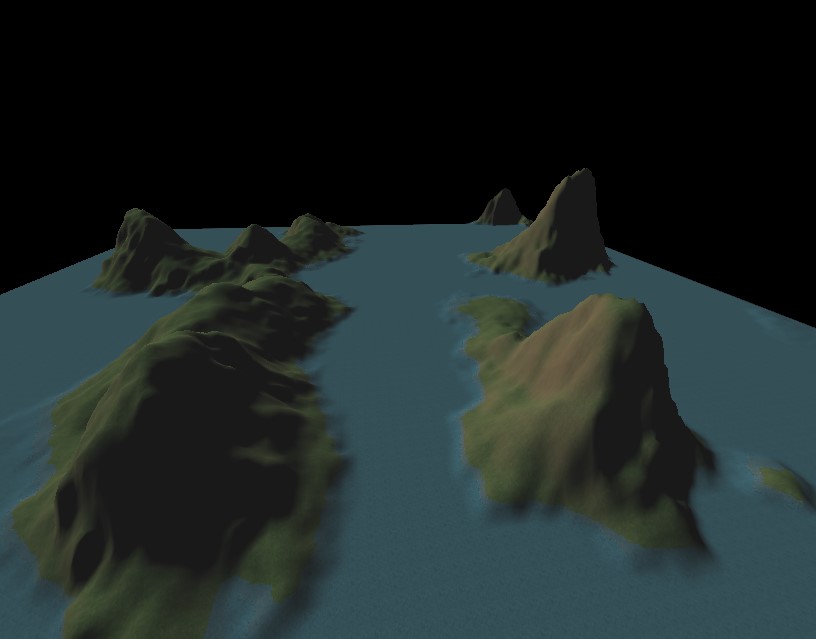
Rendering photorealistic mountain terrain

using PERLIN noise height map, intelligent multi-texturing & Directional Lighting



Karteek Kumar Mekala | Masters Project | March 1, 2014

# Abstract

Whether you are driving a tank through a war zone or watching a plane fly across Nevada, a common scene in many video games and animated movies is that of a beautiful mountain terrain. The 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.

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.

# Introduction

This is it.

# Background

This is it.

# Approach

## Scene Description

The final scene of this program will render a photorealistic mountain terrain scene in the middle of a water body. 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 can be brought up by the use of debug controls given on the keyboard. Optionally, 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

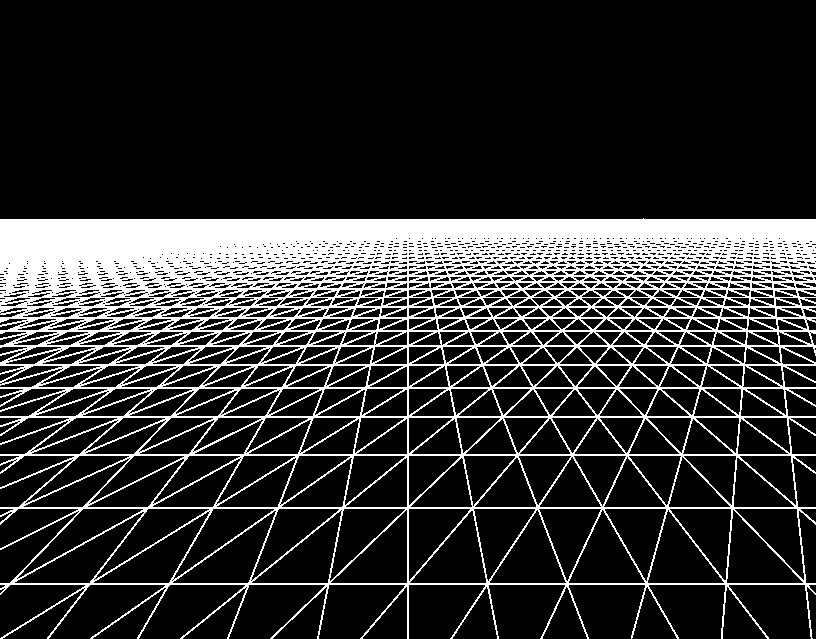
This is it.

### Height Map

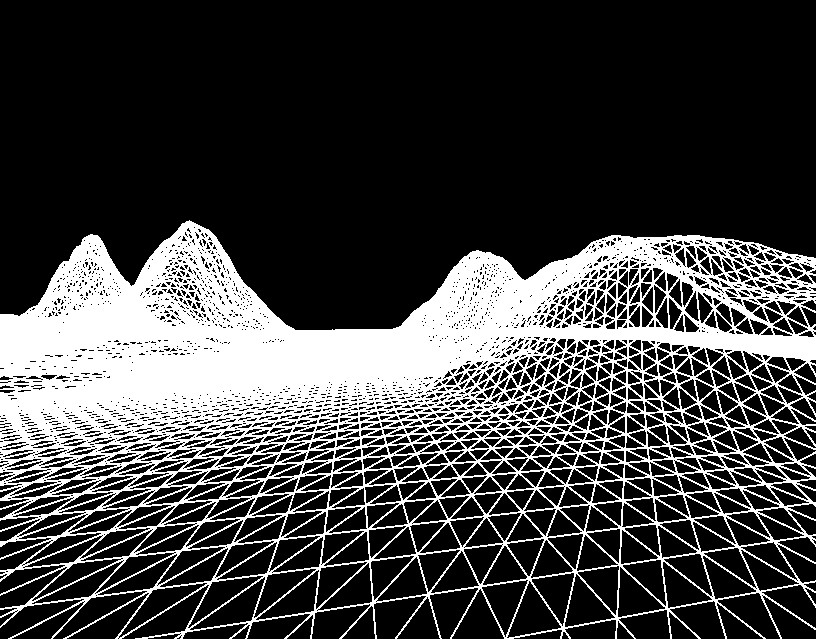
The height map resource file is generated by storing the output of the Perlin noise algorithm in the form of a grey scale bitmap. The resource file is put through a smoothing face to avoid sharp transitions in height values. The height map encoded into the grey scale image is saved away as a bitmap file 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. Each point in the grid is a vertex that contributes to two triangles in the following method.



The values from the height map are then applied to the vertices of the rectangles in the grid. This results in the tessellated from of the mountain terrain.



### Calculating Normals

As a first pass, we calculate the normal of each point P(m,n) by taking the average of the normal of the two triangles that it contributed to in the tessellation phase.

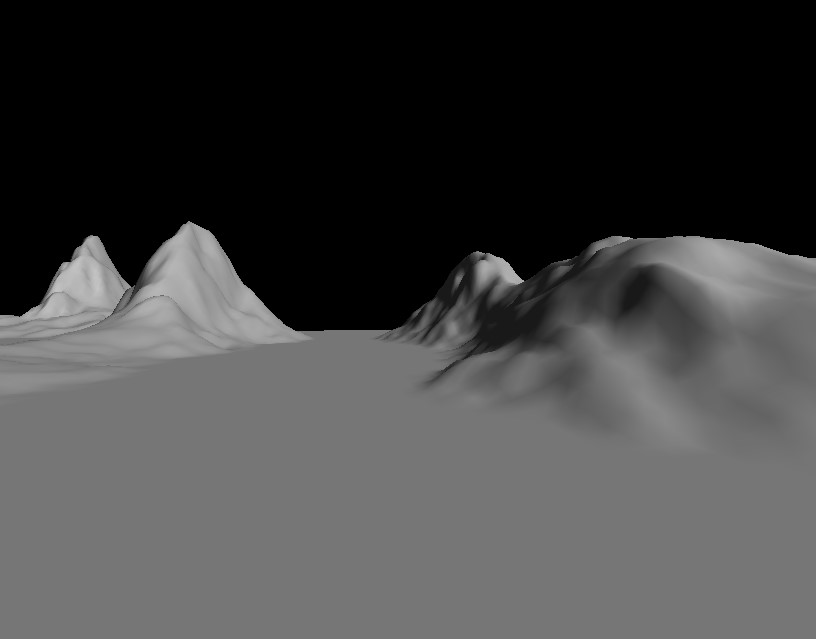
<Diagram and Vector equations explaining it>

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.

<Diagram and Vector equations explaining it>

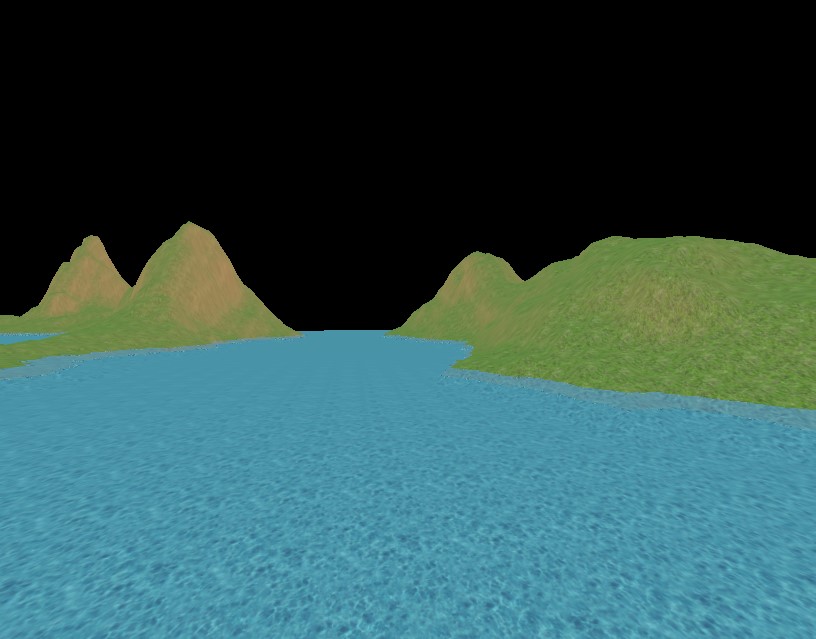
### Lighting

The normal calculated at each vertex is used to implement lighting at the pixel shader. The program sets the Directional light parameters i.e the ambient light, the diffuse light intensity and the diffuse light direction. The pixel shader takes the dot product of the calculated normal with the difuse light direction to compute the intensity of light at the current pixel.

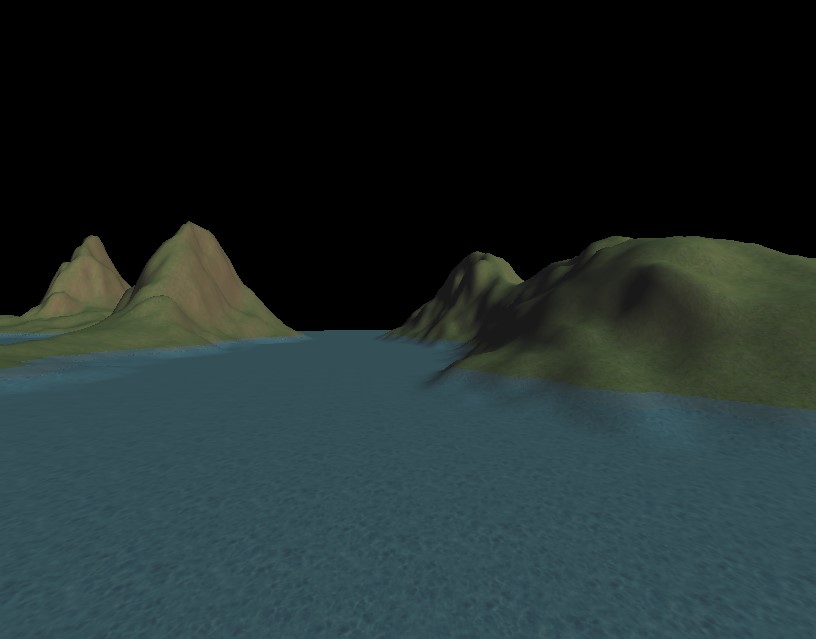


### Multi-Texturing

This is it.



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



### Skybox

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

### Camera Controls

This is it.

## Performance Statistics

This is it.

# Implementation

## 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

## Program Structure

This is it.

## Sequence Diagrams

## Use Case Diagrams

# Program User Guide

## Running the program

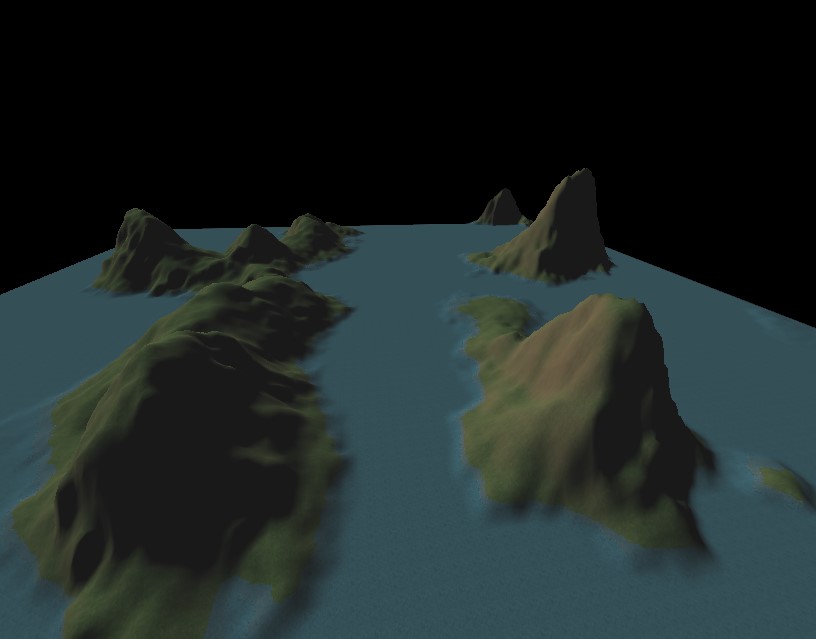
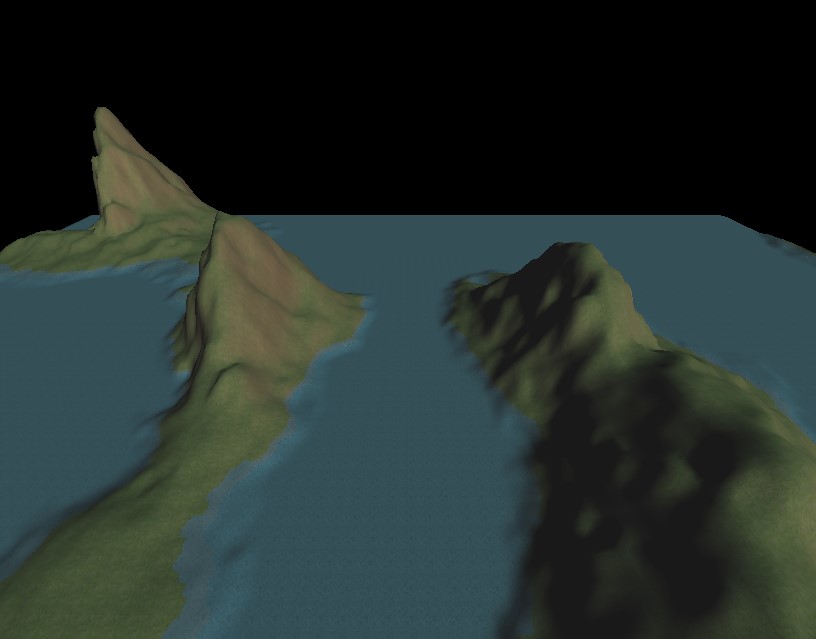
Given the source code, we compile and build the 2 solutions – “GenerateHeighmap.sln” and “MountainTerrain.sln” in Microsoft Visual Studio 2013.

First we run the height map generation program – “GenerateHeightmap.exe” to output a random height map in the form of the grey scale bitmap image. We then “MountainTerrain.exe” to see the rendered mountain terrain scene. The scene can be navigated and tweaked using the input controls described below in the “Input” section. Some screenshots are also attached in the “Screenshots” section.

## Input

This is it.

## Screenshots



# Conclusion

This project intends to demonstrate that the combination of techniques described can be used to render photo-realistic mountain terrain. The performance statistics to be collected under different parameters are expected to prove that the solution is efficient and scales easily as per the scene requirements.

# Deliverables

1. Working demo of the described Mountain Terrain scene
2. A final report including the performance statistics observed using different scene-parameters
3. A presentation for the final project defense

# Schedule

|  |  |  |  |
| --- | --- | --- | --- |
| **Target Date** | **Actual Date** | **Event** | **Status** |
| 11/09/2012 | 11/09/2012 | Pre-Proposal | Accepted |
| 04/30/2013 | 04/30/2013 | Select frameworks & toolkit | Done |
| 05/01/2013 |  | Project Website | In-progress |
| 05/14/2013 |  | Project Proposal | In-progress |
| 06/21/2013 |  | Design 1 implementation |  |
| 07/08/2013 |  | Design 2 implementation |  |
| 07/22/2013 |  | Design 3 implementation |  |
| 07/29/2013 |  | Final Report |  |
| 08/01/2013 |  | Project Defense |  |

# Future Enhancements

1. Infinite World
2. Collision Detection
3. Shadows
4. Parameter selection menu

# References

This is it.

# Author Information

Karteek Kumar Mekala  
Masters of Science – Computer Science @ Rochester Institute of Technology  
[Karteek.Kumar.M@gmail.com](mailto:Karteek.Kumar.M@gmail.com)  
kkm6815@rit.edu