

Rendering Photorealistic Mountain Terrain

Karteek Mekala
Graduate Student
Department of Computer Science
Rochester Institute of Technology
November 2014

Agenda

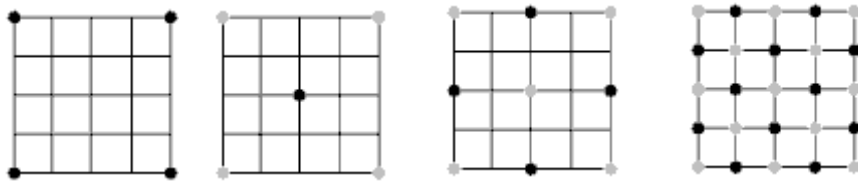
- ▶ Overview
- ▶ Background & Motivation
- ▶ Height-map Generation
- ▶ Rendering Techniques
- ▶ Technology
- ▶ Challenges
- ▶ Conclusion
- ▶ Future Work
- ▶ Demo
- ▶ Questions

Overview

- ▶ Height map generation
- ▶ Tessellation
- ▶ Lighting
- ▶ Texturing
- ▶ Skybox
- ▶ Camera Controls
- ▶ Performance Statistics

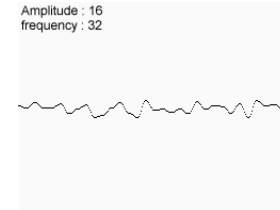
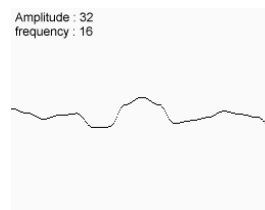
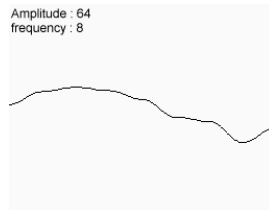
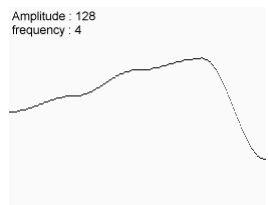
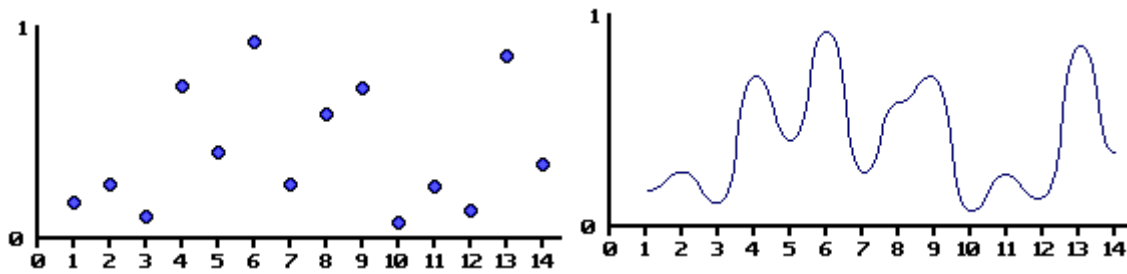
Height map generation

- ▶ John Carpenter - “Computer Rendering of Fractal Curves and Surfaces” 1979-80
- ▶ Diamond Square Recursive Subdivision



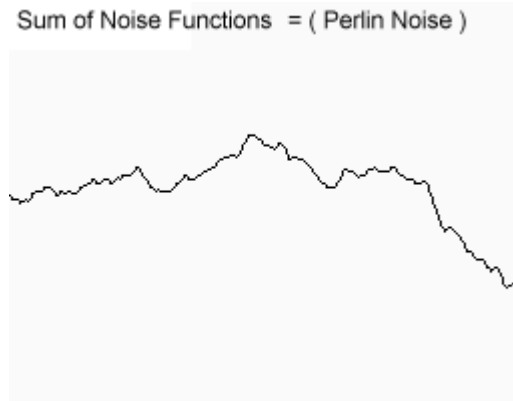
Height-map Generation

► Perlin Noise Generation



Height map generation

► Perlin Noise Generation

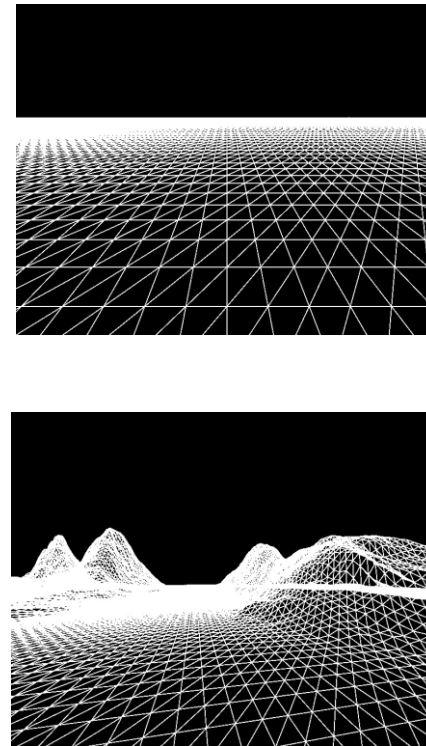


Learning here

- ▶ Perlin Noise generation - Good for parallel programming
- ▶ Save the generated height map
 - ▶ Avoid run time calculation
 - ▶ Can use bitmap compression techniques if needed
- ▶ Control of the roughness/smoothness of the height-map is desired

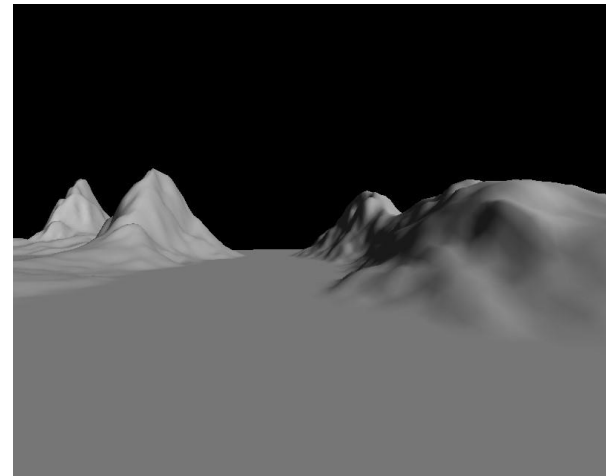
Tessellation

- ▶ Convert the height map into triangles
- ▶ Grid of $M \times N$ rectangles - divided into 2 triangles each
- ▶ Use of index buffer
- ▶ Render wire-mesh
- ▶ Calculating Normals
 - ▶ $n = (b - a) \times (c - a)$
- ▶ Tessellation shader (future work)



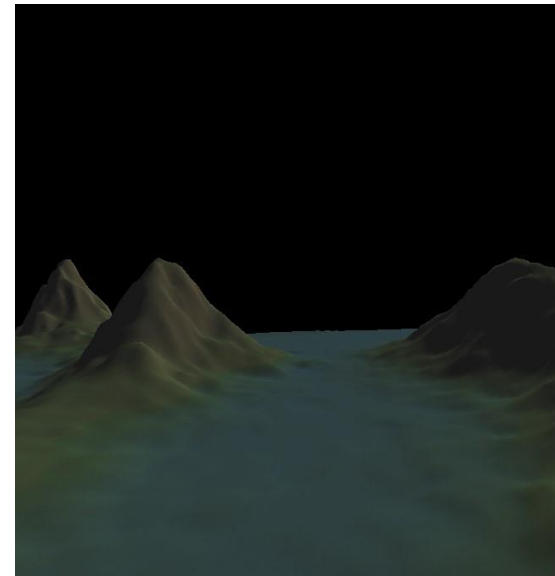
Lighting

- ▶ Fixed shader pipelines vs DirectX 11 shader pipeline
- ▶ Ambient Light
- ▶ Diffuse Light
- ▶ Single source - directional light
- ▶ Calculated at the pixel shader



Texturing

- ▶ $S = \text{Up} \cdot N$ where,
S - is the slope factor being calculated
Up - is the Up vector (0,1,0)
N - is the normal calculated for the vertex
- ▶ S is used to calculate percentage combination grass and rock texture
- ▶ Sea level
 - ▶ Transition from water to rock



Gains in this technique

- ▶ Uses the Normal and the up-vector
- ▶ Normal vector is needed for lighting calculation
- ▶ Up-vector is required for camera movement
- ▶ No additional resources such as UV maps are required
- ▶ Simple and intuitive
- ▶ Can easily be implemented at the pixel shader

Tri-planar mapping

- ▶ Image stretching
- ▶ Cannot calculate texture mapping in 2 dimensions
- ▶ Must take into account height
- ▶ 3 phases
 - ▶ XZ mapping
 - ▶ XY mapping
 - ▶ YZ mapping
- ▶ Slope determines the contribution of each phase (already calculated)

Gains in this technique

- ▶ Need to sample thrice - but is still fast
- ▶ Textures are already loaded into memory
- ▶ No additional resources/complex computations required
- ▶ Slope factor has already been calculated
- ▶ Works well even with extremely steep and extremely flat surfaces
- ▶ Simple and intuitive
- ▶ Easily implemented in the pixel shader

Camera Controls

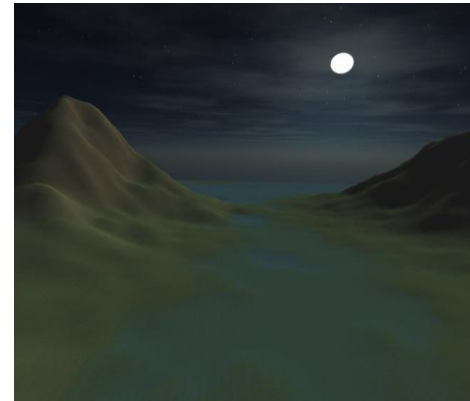
- ▶ Eye position (e) - position of the camera
- ▶ Look-at position (p) - the position in the 3d world that the camera is focused at
- ▶ Up vector (u) - the vector that is up relative to the view direction
- ▶ View direction (v) = $(p - e)$
- ▶ Right vector (r) = $u \times v$

Camera controls

- ▶ Move forward/back
 - ▶ Move eye position and the look-at position along the view direction
 - ▶ Maintain distance between eye and look-at
- ▶ Yaw
 - ▶ Rotate look-at position around the axis defined by up-vector and the eye position
 - ▶ Maintain distance between eye and look-at
- ▶ Pitch
 - ▶ Rotate the look-at position around the axis defined by right-vector and the eye position
 - ▶ Maintain distance between eye and look-at

Skybox

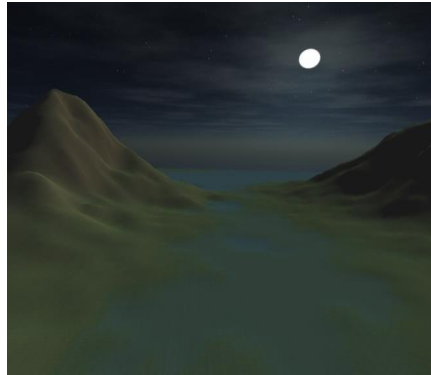
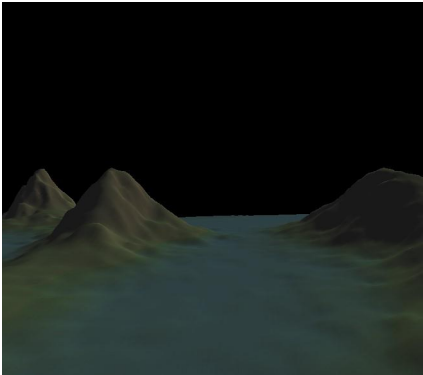
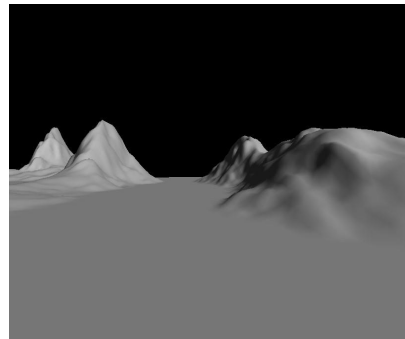
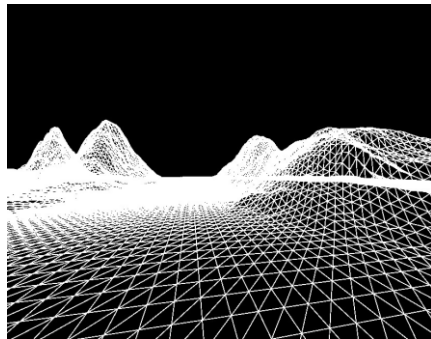
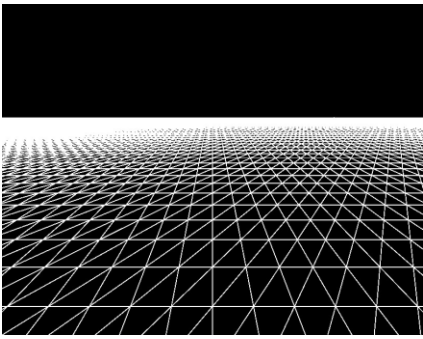
- ▶ Cube centered at the camera position
- ▶ Perception of infinite distance
- ▶ Must be closer than the far plane
- ▶ Try to line up direction of light with placement of sun / moon



Feature Toggling

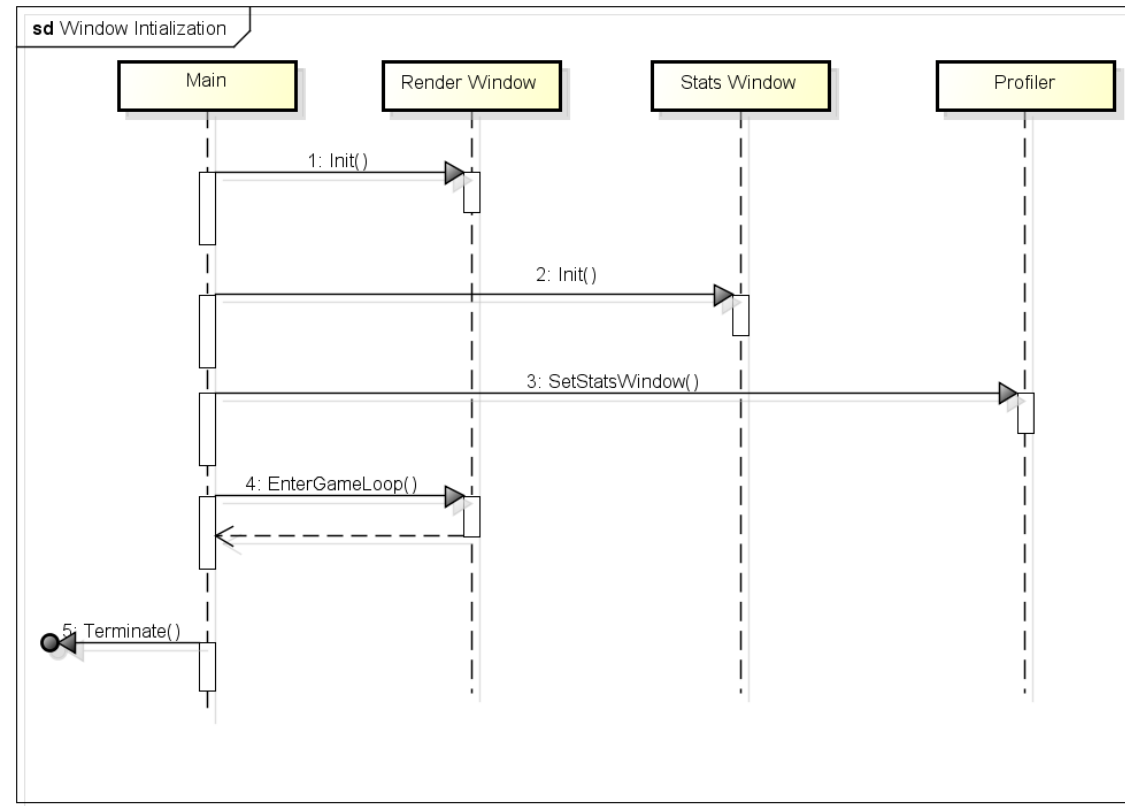
- ▶ Importance learnt from previous project
- ▶ Valuable while developing features in isolation and combination
- ▶ Extremely valuable while debugging
- ▶ Great to explain the techniques to a graphics enthusiast

Feature Toggling



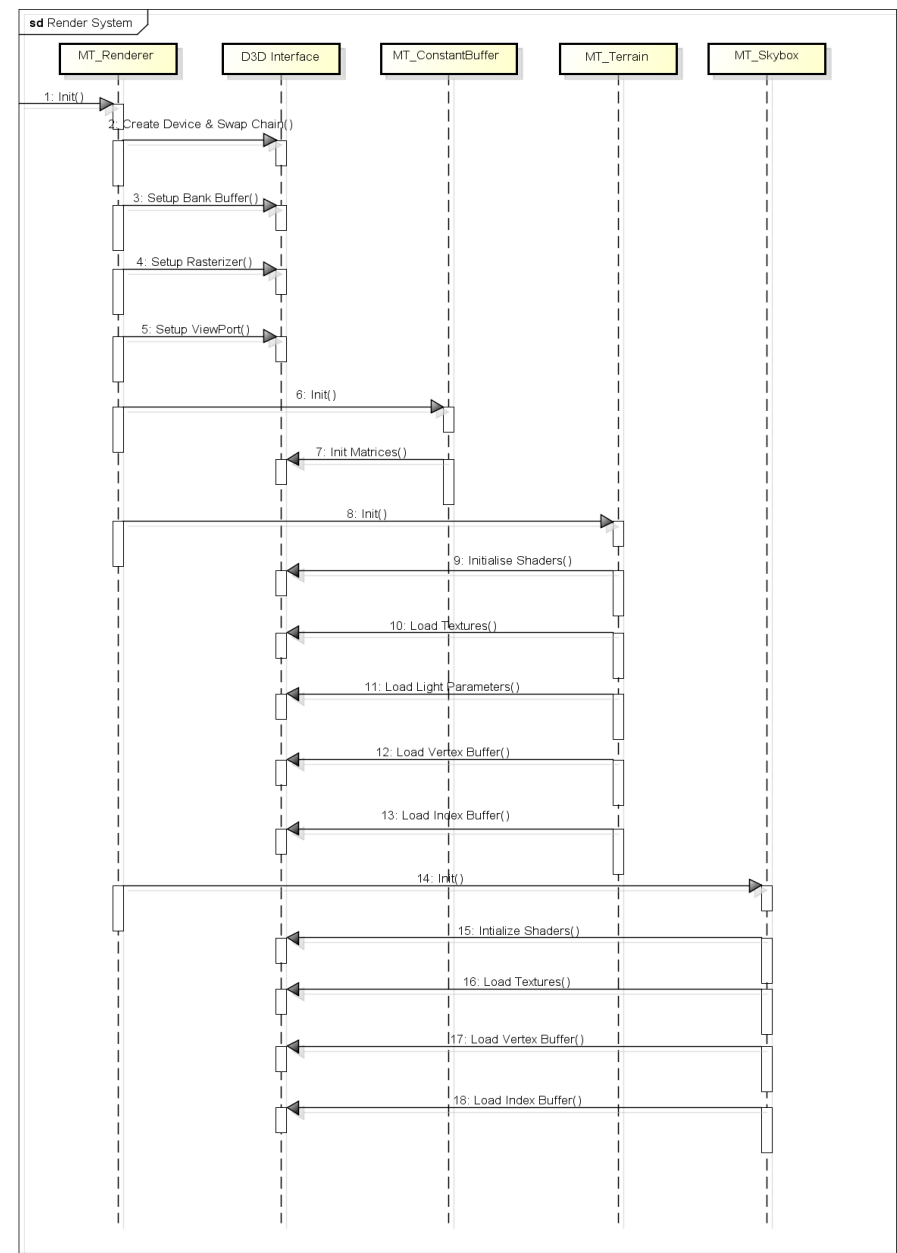
Implementation

► Window Initialization



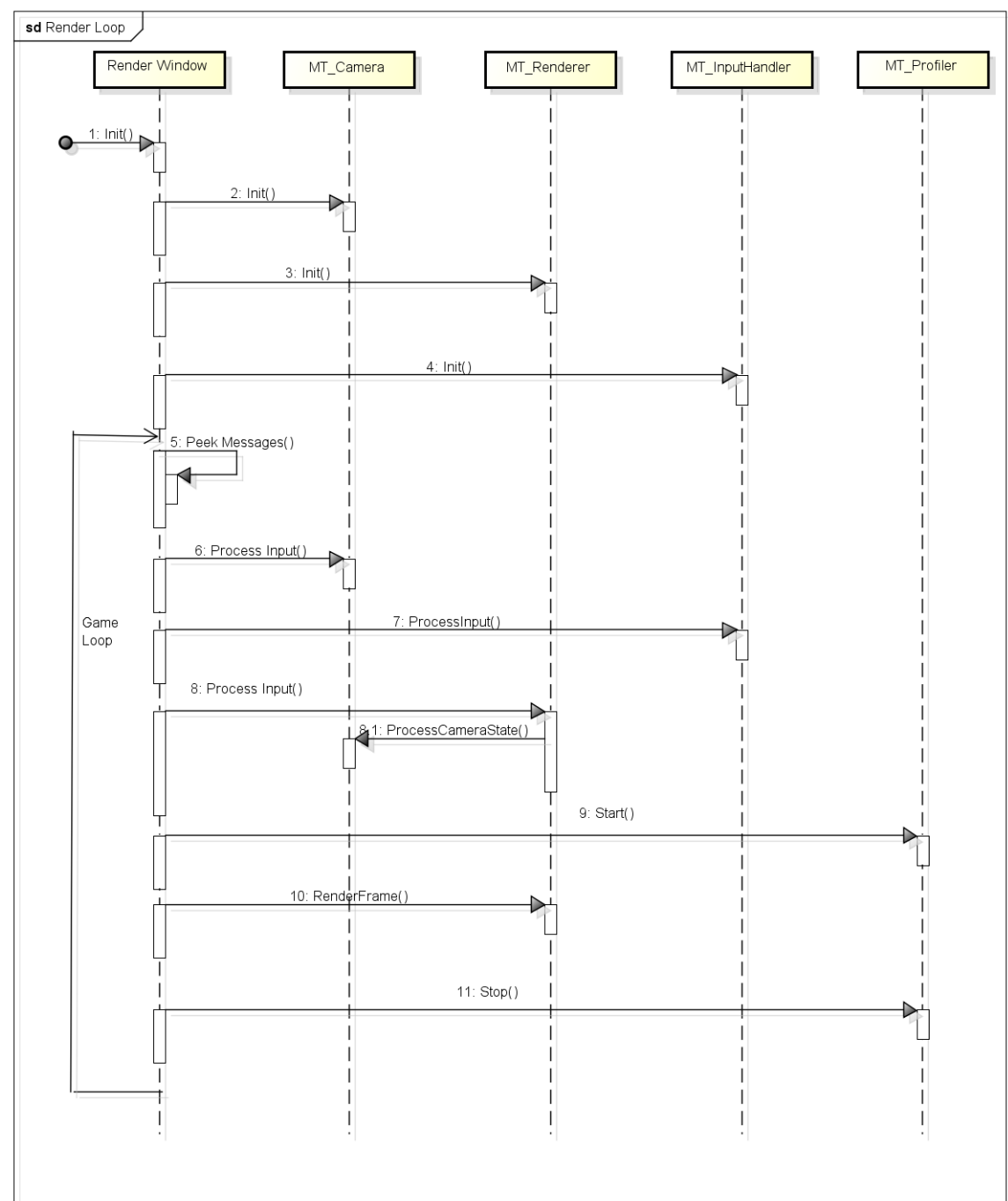
Implementation

- DirectX interface setup
 - Create device and swap chain
 - Setup Back buffer
 - Setup Rasterizer
 - Setup Viewport
- Preparing for the render loop
 - Initialize Shaders
 - Load Textures
 - Initialize Lighting parameters
 - Load Vertex buffer
 - Load Index buffer



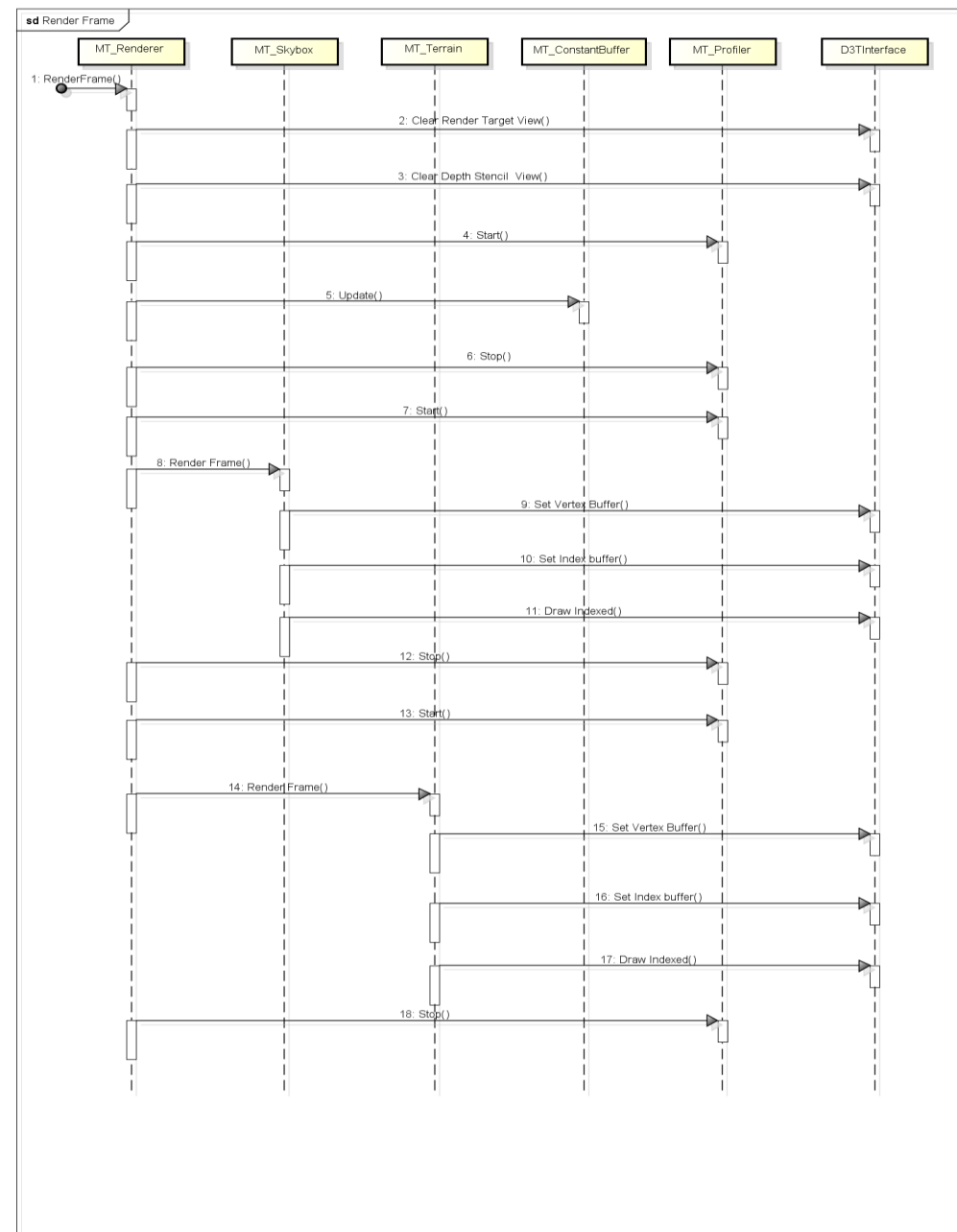
Implementation

- ▶ Render Loop
 - ▶ Message queue
 - ▶ Input Processing
 - ▶ Render frame
 - ▶ Profiling



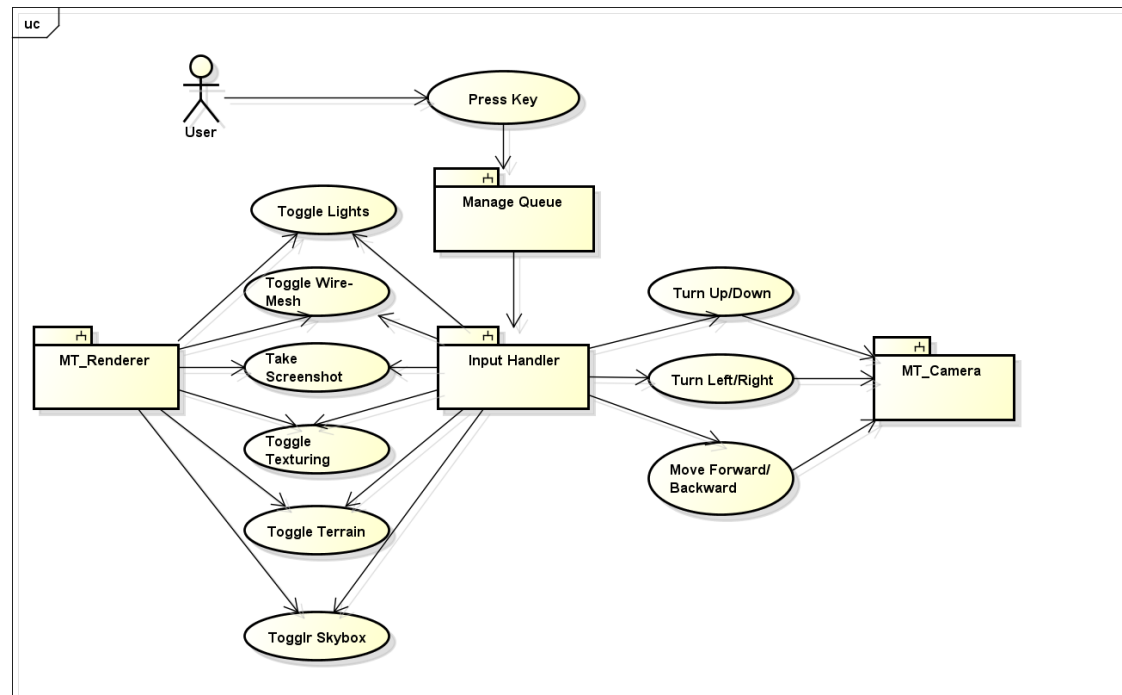
Implementation

- ▶ Render Frame
 - ▶ Update scene
 - ▶ Clear Render Target
 - ▶ Set active shaders
 - ▶ Set active vertex buffer
 - ▶ Set active index buffer
 - ▶ Draw calls
 - ▶ Profiling



Implementation

► User Interaction



Technology

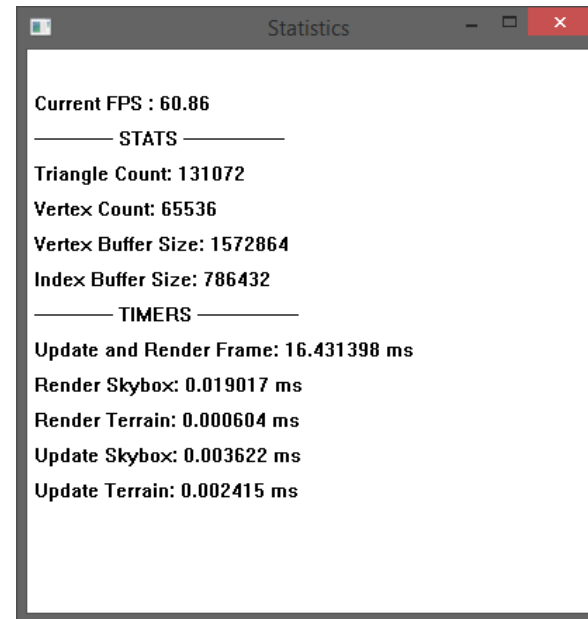
- ▶ C++
- ▶ Windows 8.1 SDK
- ▶ DirectX 11 SDK
- ▶ DirectXTK
- ▶ Visual Studio 2013
- ▶ Github / GitExtensions

Gains from DirectX 11

- ▶ Fixed shader pipeline vs DirectX 11 shader pipeline
- ▶ Vertex buffers and Index Buffers
- ▶ CPU vs GPU computation power
- ▶ Performance Gains
 - ▶ Unable to achieve 60fps in original implementation
 - ▶ Enough spare time to incorporate into much larger projects with this implementation
- ▶ No more keeping track of current “state” and matrix operations
- ▶ Development Tools
 - ▶ Visual Studio Editor and Debugger
 - ▶ Visual Studio Graphics Debugger
- ▶ Larger adoption - Windows

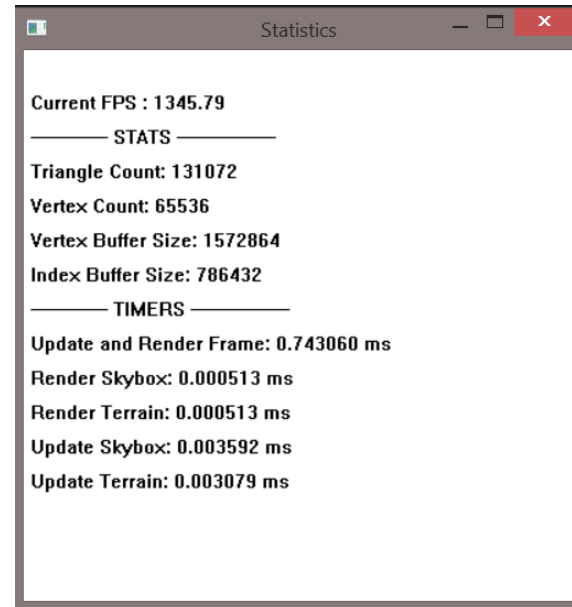
Performance Statistics

- With sync enabled



Performance Statistics

- With sync disabled



Performance Statistics

- ▶ Observations
 - ▶ Small frame time
 - ▶ Majority of the frame time can be utilized to implement complex and/or interactive scenes
 - ▶ Small memory footprint
 - ▶ no additional resources used for texturing

Challenges

- ▶ Sharp Transitions
 - ▶ Had to play with height generation parameters and smoothing
- ▶ Texture Stretching
 - ▶ Tri-planar texture mapping
- ▶ Sea Level Transition
 - ▶ Interpolating texture mapping between water and land
- ▶ Clipping
 - ▶ Skybox vs Far-plane
 - ▶ Collision detection (future work)
- ▶ Skybox Edges
 - ▶ Matching choice of skybox and water texture
 - ▶ Bounds to the scene

Conclusion

- ▶ Simple and intuitive techniques that can be examined in isolation
- ▶ Recap of texturing techniques gains
- ▶ Advantages of height map generation
- ▶ Gains observed in performance statistics
 - ▶ Frame time
 - ▶ Buffer sizes
- ▶ Candidate for integration into larger projects
- ▶ Importance of feature toggling
- ▶ Understanding contribution of each stage
- ▶ Benefits of DirectX 11
- ▶ Great for explaining techniques to a graphics enthusiast

Future Enhancements

- ▶ Shadow Mapping
- ▶ Tessellation Shader
- ▶ Terrain Editor
- ▶ Ocean Shader
- ▶ User Interface
- ▶ Collision detection
- ▶ Segway to Game Engine

Demo

- ▶ 'w' - Move camera forward
- ▶ 's' - Move camera back
- ▶ 'a' - Yaw camera left
- ▶ 'd' - Yaw camera right
- ▶ 'e' - Pitch camera up
- ▶ 'c' - Pitch camera down
- ▶ 'k' - Screen shot
- ▶ 'l' - Toggle Lighting
- ▶ 'm' - Toggle wire-mesh
- ▶ 't' - Toggle texturing
- ▶ '1' - Toggle terrain
- ▶ '2' - Toggle skybox

Questions ?



THANK YOU !
Karteek Mekala