Direct X Final: The Desert

Objective

Create a desert scene using your knowledge of Model Loading, Texturing, Lighting, and Geometry Shaders.

Exam Instructions

You have 4 hours to complete as much of this exam as possible. You may reference your graphics project, as well as slides/videos and any relevant documentation. (ex: MSDN)

Scoring Breakdown

Ground is textured with correct texture and proper repetition. 5 points

A camera exists that allows you to navigate the scene. 10 points X

Cactus model is rendering. 5 points X

Cactus model is textured and lit correctly. 10 points X

Cactus shows needles correctly. 5 points X

All 3 Rock models are rendering. 5 points X

All 3 Rock modes are textured and lit correctly. 10 points X

10 copies of each model randomly spread across the desert. 10 points X

Skybox is working correctly. 10 points X

Skybox shows “heat wave” effect. 10 points X

Ground matches the topology (height) of the demo exactly. 10 points

Ground has unique face normals and is lit by directional light. 5 points

Ground uses adjacency data to improve (soften) normal quality. 5 points

**Total 100 points**

Explanation of Tasks

**OBJ Files:**

There are a Total of **4 OBJ files** you will need to load. These files are just like any other OBJ you have loaded before. **Place 10 of each randomly across the desert.**

**Lighting:**

There are **TWO lights** in the scene that affect everything in the desert besides the skybox.

***Directional Light:*** Direction[**2x,-1y,2z**] Color[**1r,1g,0.8b,1a**]

***Ambient Light:*** Color[**0.5r,0.5g,0.5b,1a**]

**The Grid/Terrain:**

***Geometry:***

The terrain is a flat horizontal plane of **300x300** vertices centered on the origin. Each vertex is spaced apart from each other vertex by **1.0**. Besides position, each vertex contains **two** **UV coordinates as well**. **One set is for the surface color, and the other is for the height map**. To render these vertices as a **list of triangles**, you must build an **index list** to form all the needed triangles. The following algorithms can be used to assist you when **building the index array**.

**numTriangles = (TotalGridWidth -1) \* (TotalGridLength -1) \* 2**

(The below formula might be used during a nested loop, forming your triangles)

**vertexStartIndex = currGridRow \* GridWidth + currGridColumn**

There are **multiple ways to form the vertex and index buffers you will need**, (I actually used indexed triangle strips) but the algorithms above can be used to **assist** you when attempting to figure out how many triangles you will need, and how to convert a 2D grid location into a 1D vertex index location.

***Making the Grid a Terrain:***

One texture is used to **offset the World Y location in the Vertex Shader**. Use a UV set that **CLAMPS** from one edge of the terrain to the other. **(No Repeating)** The offset in the vertex shader is done by **reading the red channel of the Height-Map and multiplying by 30**.

The other texture is used to **apply surface color in the Pixel Shader**. This texture should be **WRAPPED** across the terrain **exactly 10 times**.

***The Normals:***

Manipulating the heights of individual vertices will not yield normal data for the ground. To determine what the normal of a triangle on the terrain is you will need to **deploy a geometry shader to intercept the terrain triangles and compute the face normals**. (Tip: calculating face normals was discussed on CGS day 7)

Once you have basic face normals you **improve the quality by adding adjacency data to your vertices** and then **computing** the face normals of **nearby triangles in the geometry shader** and then **averaging the results** together. (The Geomtery Shader accepts Adjacency Data)

**The Sky:**

The Sky is a standard skybox. It uses the **“DuneCube.dds”** cube map. Implement it as you like.

***The Heat-Wave:***

Use the following formula in the **skybox pixel shader** to re-create the distant rising heat effect:

***CubeLookUpX += SIN ( CubeLookUpY \* 200 + ( TIME \* 5 ) ) \* 0.005 \* ( CubeLookUpY – 1 )***

**Misc Values:**

***OBJ Model Scale:* [ 0.05 ]**

***Initial Camera Location:*** Eye[**0x,20y,-15z**] LookAt[**0x,15y,0z**] Up[**0x,1y,0z**]

***Projection Values:*** Field of View[**75degrees**] Znear[**0.1**] Zfar[**300**]

***Back-Buffer Resolution:*** [**1280w x 768h**]

***Camera Movement Speed:* [30 units per-second]**

***Camera Rotation Speed:* [45 degrees per-second]**

**Hints & Tips:**

If you can figure out what the starting vertex for your indexed triangle is, the others are close by.

The HLSL “Sample” function does not work in Vertex Shaders due to Mip-Mapping. There are other ways to access textures in shaders besides using the vanilla “Sample” though.

Adding **adjacency** data exactly **doubles your index count**. (Each index gets an adjacent index)

To **draw 10 copies** of each model **easily** use **Instancing**. (Not required though)

**Turn In:**

Please be sure your project compiles. **A project that has compiler errors is a ZERO.**

**Take a screenshot of your exam and include it with your turn-in.(printScreen/Fraps)**

Please release all **COM** objects and check that your program runs with the **DEBUG** libraries.

Delete your Debug/Release/x64/ipch/sdf files and Model/Texture folders, zip and turn in as **LastName.FirstName.Final.zip**(Make sure to unzip & test your submission!!!)

**When you have reached the end of the exam, please confirm your file was turned in before leaving. It is recommended you hang around a bit after the exam if possible, if only to make sure it can be successfully graded as there is no opportunity to retake this exam if you mess up your submission.**

