Direct X Final: Blades of Magic

Objective

Create two mythical swords using your knowledge of D3D11.

Exam Instructions

You have 6 hours to complete as much of this exam as possible. You may reference any previously completed labs, as well as slides and any documentation.

Scoring Breakdown

Ground is correct (textures etc...) 5 points

Swords are drawing 5 points

Swords are floating (hovering and spinning) 10 points

Swords are reflective (like the example) 10 points

Gems are drawing 5 points

Gems are orbiting (and spaced apart) 5 points

Gems are shaded correctly 5 points

Spot Light (ground and swords) 10 points

Green Gem Light (ground and swords and moving) 10 points

Red Gem Light (ground and swords and moving) 10 points

Blue Gem Light (ground and swords and moving) 10 points

Initial Camera/Working Camera 5 points

Fully working Skybox 10 points

A file based OBJ loader is not in use.”HFILE” appears on a model. -15 points

**Total 100 points available**

Explanation of Tasks

**The Ground:**

***Geometry:***

The ground is a **simple textured plane** generated by the Geometry Shader.

Stats: **60x60 units** across and has a purple marble **texture tiling 10 times** across the surface.

**The Swords:**

***Motion:***

The swords are each **rotated 45 degrees on their local Z axis** in different directions and then **separated by** a difference of **one unit on the world Z** axis.

The **hovering** effect is achieved by taking the **sin**(left blade) or **cosine**(right blade) of **½ the total time and offsetting it by 3.5** units and placing each sword at **that Y location** in world space.

**Each sword** **spins on its local Y axis at rate of ½ frame time radians**.

***Reflectivity:***

The **surface color** of the Swords are derived from the **“default\_reflection.dds”** cube map using the HLSL **reflect** command with the following two direction vectors: **CameraWorldPosToSurfaceWorldPos** and **WorldSpaceSurfaceNormal**.

This produces a third reflected vector which is used to access the cube-map.

**The Gems:**

***Motion:***

**Setup:** The Gems are first **translated** to the location **0x 2y 4z in world space**, then each gem is **orbited** away from the other Gems by **1/3 of a circle**.

With the gems placed, you now **orbit** them on the **global Y axis** at a rate of **½ frame time radians.**

***Shading:***

The gems are each a **solid color (red, green, blue)** to make them **appear faceted** use the **following formula** when determining the final surface color:

*GemColor \* DOT( - CamWrldPosToSurfaceWrldPosNormal , WorldSpaceSurfaceNormal )*

**The Lights:**

***Spot Light:***

A spotlight just like the one found in **lab 5**. It affects the ground and the swords.

Stats: Pos[**0x 25y 0z**] ConeD[**0x -1y 0z**] Color[**white**] Radius[**100**] InnerR[**0.988**] OuterR[**0.986**]

***Point Lights: (Three total)***

Also just like the point light in **lab 5** with a minor adjustment to make their **falloffs quadratic**.

Stats: Pos[**GemWorld**] Color[**GemColor**] Radius[**10**]

**The Sky:**

The Sky is a skybox just like you made in **lab 3**. It uses the **“nightsky.dds”** cube map. I’m actually doing it with a full-screen quad un-projected into world space for cube-map lookups. (Uses the world space pixel as a direction to read the cube-map) Do yours however you like.

**Misc Values:**

***Initial Camera Location:*** Eye[**0x,3y,8z**] LookAt[**0x,2y,0z**] Up[**0x,1y,0z**]

***Projection Values:*** Field of View[**75degrees**] Znear[**0.1**] Zfar[**100**]

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

***Camera Movement Speed:* [3 units per-second]**

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

**Hints & Tips:**

A skybox is not supposed to interfere with the Z values in the rest of the scene.

When multiplying **matrices, ORDER MATTERS!**

The **Sky Box and Gems ARE NOT LIT!** They are not affected by the four scene lights.

A **View** matrix is by definition **NOT** in **world space**.

When creating shader resources like constant buffers & textures, be **explicit** when assigning their **HLSL slot register**. This **eliminates confusion** during binding in C++.

**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.

**Turn in your final on Sidekick and carefully follow the Turn-In procedure shown.**

**You may only leave early if the staff confirms that you have fully completed the exam.**