Direct X Midterm: Line Universe

Objective

Create a scene showing a planet being analyzed. Use your understanding of geometry and linear algebra to create and manipulate 3D shapes using lines. Use your knowledge of 2D & Cube-Map Texturing to create a skybox with a central planet which always faces the camera.

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

3D lines are rendering in some way. 5 points

A 3D camera is available and can be used to navigate the 3D scene. 10 points

A Spiral circle is rendered from the origin. 10 points

The Spiral circle has alternating colors. 5 points

The Spiral circle appears to spin. 5 points

The Spiral circle slowly descends on the Y axis as it grows. 5 points

A dashed line circle is visible. 5 points

Multiple dashed line circles form the shape of a sphere. 15 points

The Line Sphere sits above the Spiral and rotates correctly. 5 points

A vertical Quad displays the image of the Planet properly. 10 points

Planet pixel shader rejects transparent pixels. 5 points

Planet Quad is inside the sphere and always faces the camera. 10 points

Skybox correctly displays the distant environment based on camera angle. 10 points

**Total 100 points**

Explanation of Tasks

**The Camera:**

The camera is exactly like the ones from Lab 2 & 3. You should use the mouse for rotations. If you use only the keyboard you will still get credit but you must use the arrow keys for rotation.

Tip: Global Rotation formula.

*currPos = MatrixPos*

*MatrixPos = Origin*

*Matrix \*= RotationMatrix*

*MatrixPos = currPos*

**The Spiral:**

***Geometry:***

The Spiral is built in a very similar way to the gold ring you created in Lab 1. The spiral is built in 3 dimensions however. The Spiral spans **5000 degrees** of revolution. With each increased degree the circle slowly grows in size, this creates the spiral effect.

Creating a vertex for every degree is overkill, thus you will only need a **new vertex every 6 degrees**. (The circle will still be smooth)

*Rate of Growth:* **Axis \*= Degree \* 0.002**

*Y Axis Decay:* **Degree \* *-*0.0005**

*Color of Ring:* Every degree which is a **multiple of 4 is Blue**, otherwise it is **Green.**

***Behavior:***

The Spiral appears to grow infinitely over time; however this is just a visual illusion. The spiral geometry is simply rotated on **Y axis** at a rate of ***-*57.3 degrees** (-1.0 radian) **per second**.

**The Line Sphere:**

***Geometry:***

Once again we have yet another take on the circle from Lab 1. Unlike the spiral, each circle is only **360 degrees**. However there are **multiple scaled circles stacked on the Y axis to form the sphere**. (The whole sphere is rotated 90 degrees later on)

Like the spiral, a **vertex** is only placed **every 6 degrees**. However, you may have noticed that these rings are drawn slightly differently. (Hint: Topology) This ensures the rings don’t connect when stacked one on top of the other.

The trick to making the rings into a sphere is as follows:

You will need to **loop from 0 to 180 degrees** on the Y axis. **Every 6 degrees** you should **create a new ring** as explained above. The **Y degree** value can be used to control not only the **height** of the new ring but its **size** as well. If you do this correctly you can form a **perfect sphere**. (Hint: No hard coded values are necessary, think carefully as to why 0 to 180 was chosen)

***Behavior:***

Once you have the sphere appearing correctly, you will need to move it into position. The following transforms must be applied to make the sphere work like it does in the sample.

*Translation:* **0x, 1y, 0z**

*RotationZ:* **57.3 degrees per second**

*RotationX:* **90 degrees from start**

*Scale Factor:* **2.0**

The above operations are **NOT in Order**. It will be up to you to determine what that must be.

**The Planet:**

***Geometry:***

A Quad built vertically around the origin on the X & Y axis. **Span of 4.0** across & down.

***The Texture:***

The planet uses the **“planet-2.dds”** 2D texture. Apply it **once** across the surface.

***Hiding the Edges of the Texture:***

This is actually trivial to do. In the Pixel Shader for the planet, **grab the alpha from the texture** for the pixel. **IF** the **alpha is less than 0.1**, then call the **“discard”** HLSL intrinsic. This will prevent the pixel from being output.

***Making the Planet face the camera:***

*You will need:* PlanetPos( 0x, 2y, 0z ), CameraPos in World Space( ? ? ? ), WorldUp( 0x, 1y, 0z )

*Step one:* Determine the direction from the planet to the camera. (CamWorld - Planet)

*Step two:* Normalize step one; this will be your new Z vector.

*Step three:* Cross the new Z vector with WorldUp then Normalize. This is the new X vector.

*Step four:* Cross the new X and Z vector then Normalize to form the new Y vector.

*Step five:* You now have everything you need to build the billboard world matrix. Create an Identity matrix and inject the X, Y, Z and PlanetPos vectors into this matrix at the proper locations. (If you are unsure what goes where, look at the end of the day 2 slides)

**The Background:**

The stars & planets in the background are **a skybox** like the one found in Homework 3. To draw the skybox, just do the following steps:

**Create a cube** just like the one in Lab 2.

**Position the cube where the camera is** in world space and bind the “Space2.dds” map.

Before drawing anything else, **draw the cube inside out**. (CCW)

To apply the skybox texture you **will not actually need any UV coordinates.** Instead **sample** the **cube texture** by **using a direction vector** calculated in the following way: (you can do this in the VS then send to PS)

**CubeMapSampleVector = MatrixMultiply ( LocalSpaceXYZasDirection, WorldMatrix )**

Once you have the resulting color, just return it from the pixel shader.

Finally **clear your Z buffer again** and draw the rest of the scene.

**Misc Values:**

***Initial Camera Location:*** Eye[**-3x,2y,-4z**] 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**]

**Hints & Tips:**

An inverted Camera/View Matrix is a World Matrix. The **reverse is also true**. (If orthogonal)

The function **“XMMatrixDecompose”** may prove useful.

**“GetPhysicalCursorPos”** will tell you where the mouse is at any given time.

**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 midterm 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.**