Direct X Final: Balloon Popper

Objective

Use your knowledge of D3D11 lighting, model loading and texturing to Draw and Pop three moving colorful balloons.

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

Fully working Skybox 10 points

Fully Working 3D mouse-look Camera clamped to Ground 10 points

Crossbow is correct (textures etc…) 5 points

Crossbow is mounted to camera properly 10 points

Crossbow is lit by directional and ambient lights 5 points

Crossbow does not intersect ground or balloons 5 points

Crosshairs appear correctly 10 points

Red, Green and Blue Balloons are rendering 5 points

Balloons demonstrate directional & ambient lighting 5 points

Balloons are made shiny using the specular lighting formula 10 points

All Balloons move around based on their color 5 points

Balloons can be popped when under the crosshairs with left mouse 10 points

Balloons grow back smoothly 2 seconds after being popped 5 points

The geometry "HFILE" can be seen above a model's regular geometry. **-15 points**

**Total 100 points available**

Explanation of Tasks

**The Ground:**

***Geometry:***

The ground is a **simple textured plane** generated by the Geometry Shader. (No lighting)

Stats: **600x600 units** across and has a grass **texture tiling 100 times** across the surface.

**The Skybox:**

**Just like** the one you did 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 **“LostValley.dds”** map.

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

**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 )**

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

**The Camera/Player:**

The **“player” is just a camera** like you made in **lab 3.** The only difference being that it is permanently **stuck at 3.0 on the world Y axis**. Use the change in mouse X & Y coordinates to rotate the camera at a reasonable speed.

**The Crossbow:**

***Model:*** Load this just like you did the chair in lab 5. (OBJ)

***Mounting to the Camera:***

Every frame, **move the crossbow into the local space of the Camera**. Then **use the following values to “offset”** the model so it appears just like the demo.

*Translation* [**1x,-0.85y,1z**] *Rotation* [**90degrees Y local**] *Scaling* [ **0.75** ]

**The Crosshairs:**

The crosshairs are **just a clip space quad** created with the geometry shader. The quad uses the **“crosshair.dds”** texture and discards any semi-transparent pixels. The quad has a **width of 0.06** and uses the aspect ratio to ensure it stays square.

**Lighting:**

There are **TWO lights** in the scene **that affect both the Crossbow and the Balloons**.

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

***Directional Specularity:*** (Balloons Only!) SpecularPower[**25**] SpecularIntensity[**0.75**]

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

**The Balloons:**

***Model:*** Same deal as the crossbow, however you will not actually need the UV data.

***Initialization:***

**Each balloon contains three vital pieces of information:** World Position, Color, and whether it has been popped. The **Balloons are placed around the origin 5 units apart on the X axis with a height of 4 on the Y.**

The Color of a balloon is determined based on the order of its creation. **Red, Green, Blue…**

The final property is if the balloon is **not popped or popped**. Use a scalar value to track this.

***Movement:***

Every frame affect the positions of all the balloons using following formula:

**BalloonWorldPositionXorYorZ += sine( TIME \* 2 ) \* DELTA \* 5**

Use the current balloons color to determine which axis to modify. **R=X, G=Y, B=Z**

***Popping the Balloons:***

You can use the following method to determine if a balloon is near enough to the crosshairs:

*Step 1:* Get the **location** of the balloon **in CLIP space**.

*Step 2:* Get the **2D distance** of this location **from the center of CLIP space**.

*Step 3:* If this distance is **LESS than 0.075** clip space units, you should **“Pop” the balloon**.

A **"popped"** balloon should be **hidden for 2 seconds** and **then grow back over 1 second**.

**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[**600**]

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

***Camera Movement Speed:* [7 units per-second]**

***Camera Rotation Speed:* [1/2 degree per-pixel moved by mouse]**

**Hints & Tips:**

When multiplying **matrices, ORDER MATTERS!**

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

If your crossbow & crosshair are **”intersecting”** with the rest of the scene, try drawing them **last**.

**Turn In:**

Please be sure your project compiles **AND RUNS**.

**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.

**If it does not I will dock 10 points.**

Delete your Debug/Release/x64/ipch/sdf files and 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. You may only leave early if you have reached the 100% mark.**