CSE 190 Discussion 6

HW3: CAVE Simulator

Agenda

- Homework 3 Recap
- FrameBuffer Usage
- Projection for CAVE Screen
    - Math
    - Implementation
- Viewport Switch
    - Debug Wire Frame
- Extra Credit

Homework 3 Recap

- Link to the assignment: http://ivl.calit2.net/wiki/index.php/Project3S18
- Due Date: May 18th 2pm (This Friday!)
  - If you have scheduling conflicts, let us know
- Some updates:
  - Only need to show debug wireframe in the head-in-hand viewing mode while “A” is pressed.
  - Clarifications for some potential misunderstandings
  - Let us know if you find anything unsure

Framebuffer Usage

- Review of the framebuffer:
  - A container to hold back the stuff you want to draw
  - So that we can render them to the texture
- Note:
  - GLFW has its default framebuffer to draw onto window
  - So whenever you do:

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

  - You are binding back to the default frame buffer.

Framebuffer Usage

- The general idea is that
  - Initialize framebuffer for each of your quad
  - When drawing (for each eye):
    - Bind to your framebuffer
    - clear background and color/depth bits
    - Draw stuff that you want to render to your texture
    - Unbind your framebuffer*
    - Render your CAVE screen (the quad)
- *Caveat:
  - Don’t unbind to the default framebuffer!
  - Instead, unbind to _fbo, which is the framebuffer used by the minimal example to render to your HMD

Projection Matrix for CAVE screens

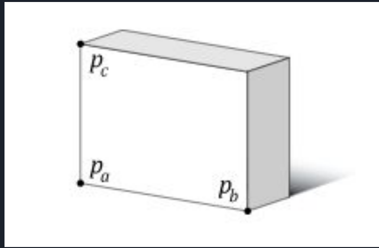
- For simplicity, I'll call each virtual CAVE screen as a "frame"
- Review of what we did to get the projection

$$P' = PM^T T$$

- This functions gives you the projection matrix for each frame

Projection Matrix for CAVE screens
- Obtain “P”

- Since each frame has different translation/rotation, the resultant projection should be different.
- Where does the difference come from?




```
PA glm::vec3(-1.0f, -1.0f, 0.0f);
PB glm::vec3(1.0f, -1.0f, 0.0f);
PC glm::vec3(-1.0f, 1.0f, 0.0f);
```

- To get correct Pa, Pb, Pc for each frame:
  - Pa = model_matrix * vec4(PA_x, PA_y, PA_z, 1.0f);
  - Same for Pb and Pc

Projection Matrix for CAVE screens
- Obtain “P”

- OpenGL gives us a wonderful function:

```
glm::mat4 P = glm::frustum(l, r, b, t, n, f);
```

- To make use of it, we need to compute l, r, b, t.
  - Last discussion gives step-by-step explanation of how to compute from Pa, Pb, Pc to get the l, r, b, t.
- ‘n’ and ‘f’ defines the near/far clipping plane
  - Depends on how you want to clip user’s view

Projection Matrix for CAVE screens

- Obtain "M_T"

- Review of the formula for M_T

$$M^T = \begin{bmatrix} v_{rx} & v_{ry} & v_{rz} & 0 \\ v_{ux} & v_{uy} & v_{uz} & 0 \\ v_{nx} & v_{ny} & v_{nz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- When you get 'P', you are likely to already have Vr, Vu and Vn
- Then?
    - Simply create a mat4 M_T and plug things in!

Projection Matrix for CAVE screens
- Obtain “T”

- Review of the formula for T

$$T = \begin{bmatrix} 1 & 0 & 0 & -p_{ex} \\ 0 & 1 & 0 & -p_{ey} \\ 0 & 0 & 1 & -p_{ez} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- What is ‘Pe’ again?
  - The eye position!
- Implementation?
  - Create an identity matrix T and translate it by ‘-Pe’

Projection Matrix for CAVE screens
- Rewind

- Now take a look at the formula again

$$P' = PM^T T$$

- Note P’ is the actual projection that we want to return, not P.
- What’s the next step when I get the projection?
  - Draw your scene to framebuffer and render them onto the texture of your frame!

Viewport Switch

- You need to be able to switch the view position from your head to your right controller.
  - Think of it as if your controller is head of the person who is wearing the head tracker. And you are in the spectator view.
- Note:
  - Just like when your rotate your head, the scene in the frame should not rotate, when you rotate your controller in this mode, the scene should not rotate.
  - You still have two “eyes” on your controller in this mode.

Viewport Switch

- Although rotation is not reflected in the scene, you are still expected to see some changes while rotating controller.
    - Controller’s forward perpendicular to your head forward
        - The image becomes mono.
    - Controller’s forward is in reverse direction
        - The image is inverted stereo.
- Chalkboard time

Viewport Switch
- Debug Wireframe

- The point of this is that
  - You can visualize the two “eye positions” on your controller
  - And visualize the six pyramids
    - Green pyramids for left eye, red for right eye
- Note:
  - You need to draw 6 pyramids to both eyes. (Not 3 pyramids for each eye)
  - GL_LINES and GL_TRIANGLES to draw lines/surface
  - If drawing surface, you might want to adjust the alpha

Extra Credit
- Fly Around Virtual Campus

- Load campus model with mtl and textures rendered correctly. Scale the model to true size. (similar to air race game in 165, but in GL, please ignore the check points)
- Fly using 6DOF: positions and orientations of the controller

Extra Credit
- Load campus map

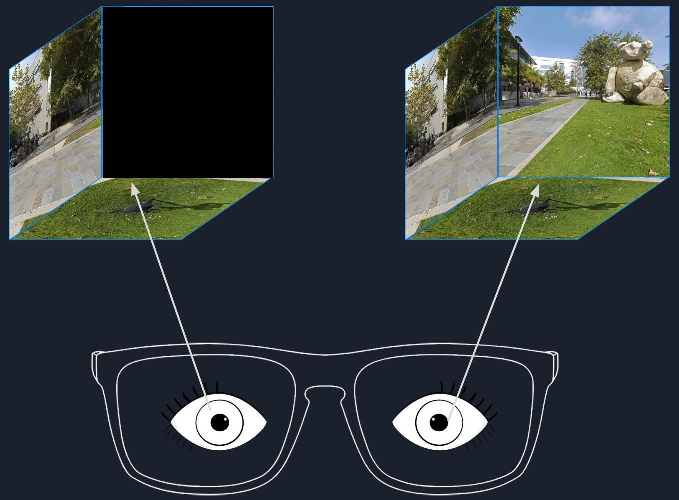
- Assimp is going to be very helpful
  - Handles textures/pictures/UV maps
  - You need to figure out how to iterate through all the necessary components.
- Challenge: Z-Buffer Precision issue
  - Why and how will it look like and how to resolve?
  - http://outerra.blogspot.com/2012/11/maximizing-depth-buffer-range-and.html

Extra Credit
- Add skybox to virtual CAVE

- Note:
  - Better choosing a different one than the bear skybox
  - It should be the skybox that wraps the entire virtual world
- Nice skybox for your to choose from
  - http://www.custommapmakers.org/skyboxes.php
  - And they are free!
  - Capture your own skybox pictures if possible.

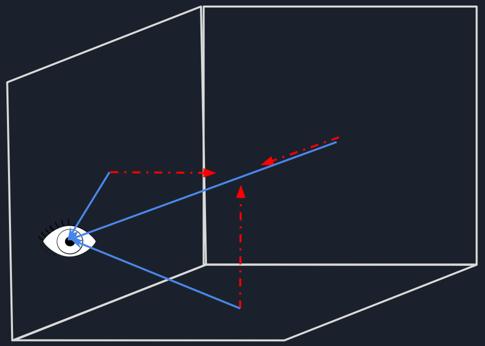
Extra Credit
- Simulate failure of one projector

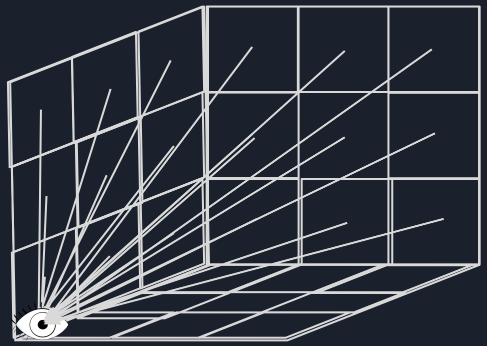
- Press button to render one random black square screen (just one eye, not an entire wall - assuming using passive stereo)



Extra Credit
- Brightness falloff of LCD

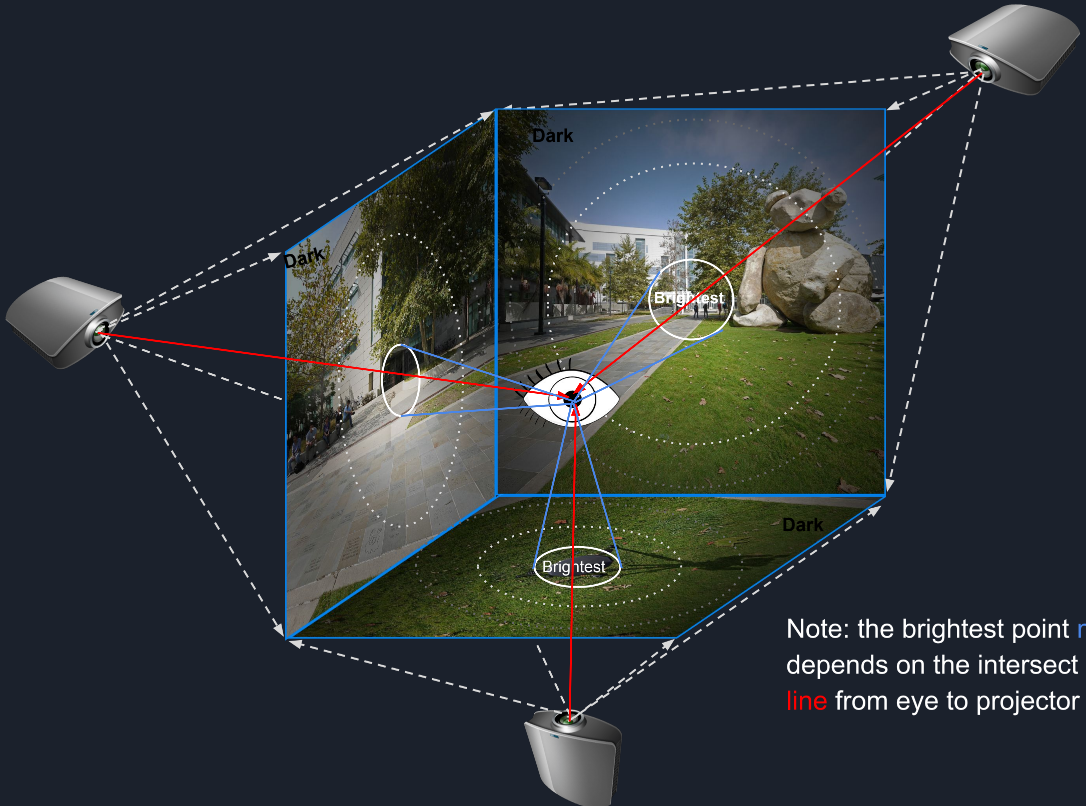
- **Reduce entire square based on the angle (0 - 90 degree). (4 pts)**
  - **Entire screen gets darker or brighter.**
- **Reduce each pixel through shader (0 - 90 degree per pixel). (4pts)**
  - **Brightest at the pixels that are** looking directly **to the eye (normal to the frame)**





Extra Credit
- Vignetting on projected screens

- Imaginary projectors (2.4 m behind screen projects to the center of the screen)
- Use shader.



Note: the brightest point moved around depends on the intersect point of where the line from eye to projector intersects the screen

Extra Credit
- Linear polarization effect

Polarized direction for glasses

Polarized direction for cave screens

QUESTIONS?