CSE 190 Discussion 5

PA3: CAVE Simulator

Agenda

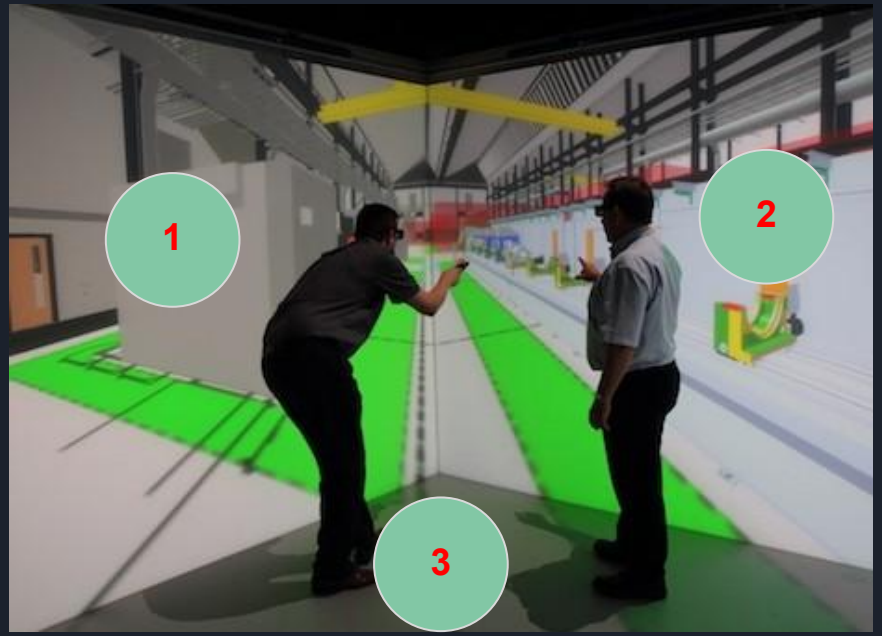
- PA3:
  - CAVE Simulator Intro
  - Rendering to Texture using OpenGL
  - Generalized Perspective Projection
- Helpful references

Project 3

- Project 3 Due Date: May 17th 2pm
  - If you have scheduling conflicts, let us know
- The idea of this assignment:
  - Understand the concept of the CAVE system
  - Learn how to render the scene to textures on quads
  - Figure out the implementation of Perspective Projection
  - And to have fun!

CAVE Simulator

- Features to implement:
  - Render the scene to 3 squares
  - Ability to switch the viewport from HMD position to the Controller position
  - Ability to freeze the viewport position
  - Manipulate calibration cube
  - Details in assignment page



Render Scene To Texture

Render to Texture

- Goal:
  - Create CAVE screens, rendering different views to different screens
- To achieve this:
  - Create a texture out of the different views
  - Render each screen as a texture
  - Paste texture onto a quad
- We have three screens and two eyes so
  - Need to render the scene six times to off-screen buffers

Framebuffers

- Framebuffer:
  - A container for textures
  - Holds textures we can use later
  - Allows us to render to places other than the screen we see
- To use the framebuffer:

```
GLuint fbo = 0;
glGenFramebuffers(1, &fbo);
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
```

Textures

- Will need a texture to hold what to draw on our CAVE screens
- Note:
  - Pass in NULL for the data since this is a placeholder for our screen information
  - Also need to attach the texture to the framebuffer

```
GLuint texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, TEXTURE_WIDTH, TEXTURE_HEIGHT, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texture, 0);
```

Renderbuffers

- Renderbuffers:
  - A type of framebuffer attachment (like textures)
  - Store data in a format that is optimized for off-screen rendering to a framebuffer (write only)
- Good for PA3 since we want depth information but don’t need to render the depth information (don’t need to read)

Renderbuffers

- To create, follow similar steps as with VBO/VAO/textures...
  - Generate
  - Bind
  - Information about what it will contain

```
GLuint rbo;
glGenRenderbuffers(1, &rbo);
glBindRenderbuffer(GL_RENDERBUFFER, rbo);

glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, TEXTURE_WIDTH, TEXTURE_HEIGHT);
glBindRenderbuffer(GL_RENDERBUFFER, 0);
```

Renderbuffers

- Renderbuffer is a framebuffer attachment so
  - Attach Renderbuffer to currently bound framebuffer (similar to attaching our texture)

```
glFramebufferRenderbuffer(      // attach the renderbuffer object
    GL_FRAMEBUFFER,             // 1. framebuffer target
    GL_DEPTH_ATTACHMENT,        // 2. attachment point
    GL_RENDERBUFFER,            // 3. render buffer target
    rbo);                       // 4. Renderbuffer ID
```

Rendering to the texture

- To render:
  - Bind the new framebuffer to make it the active framebuffer
  - Render as normal
    - This colors the texture in our framebuffer
  - Bind the default framebuffer
  - Render the screen quad with the resulting texture

```
// bind our framebuffer
glBindFramebuffer(GL_FRAMEBUFFER, fbo);

// render scene

// bind the default framebuffer
glBindFramebuffer(GL_FRAMEBUFFER, 0);

// render quads with the texture
```

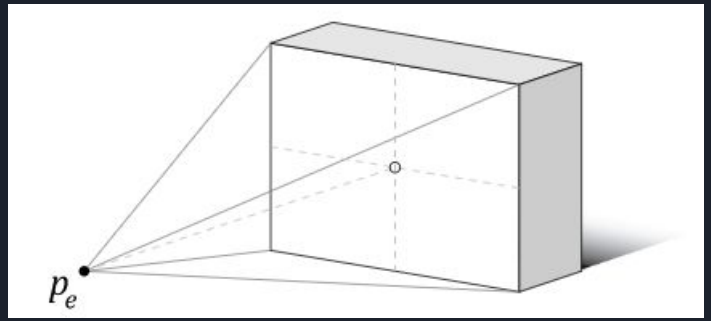
Rendering to the texture

- NOTE:
  - The texture width and height need to match what is used in glViewport() from RiftApp class
- So when you are rendering your scene:
  - Save the glViewport parameters before rendering to FB
  - Set the glViewport to match the texture size
  - Render the scene onto the texture
  - Reset the viewport
  - Render the Cave

Generalized Perspective Projection

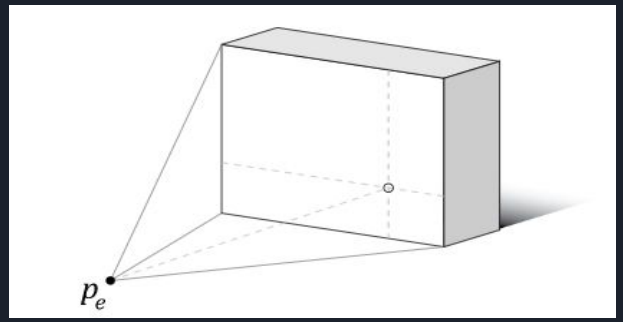
Perspective Projection

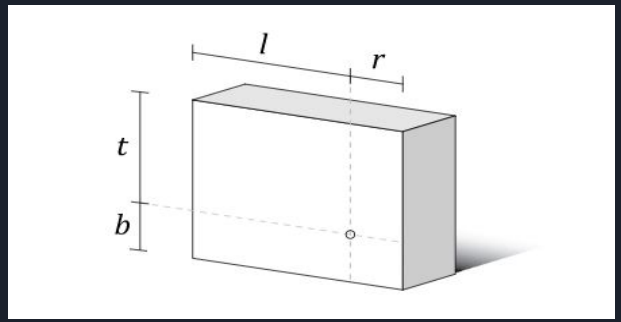
- Typically we use the projection matrix generated by gluPerspective (or glm::perspective)
- This works under the assumption that we are directly in front of the screen and perpendicular to it
  - So we are looking at the center of the screen



Off-axis Perspective Projection

- In a CAVE, we cannot view every screen head on, so each screen needs a different perspective
- glFrustum (or glm::frustum) can generate the perspective matrix for us given several parameters (left. right, top, bottom, nearPlane, farPlane)



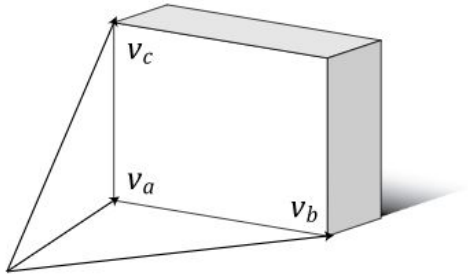


(near and far plane parameters are not shown)

Calculating Frustum Parameters

1. Calculate vectors from eye position to the screen corners
2. Calculate distance from eye position to the screen space origin

1.

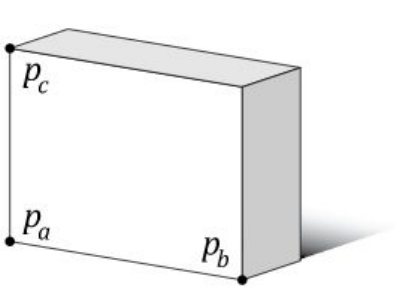


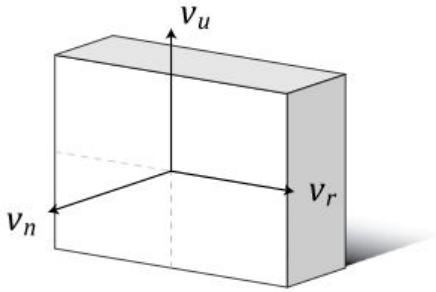
$v_a = p_a - p_e$ $v_b = p_b - p_e$ $v_c = p_c - p_e$

2. $d = -(v_n \cdot v_a)$

Calculating Frustum Parameters

3. Calculate the frustum extents at the near plane





$$l = (v_r \cdot v_a)\, n/d \qquad r = (v_r \cdot v_b)\, n/d$$

$$b = (v_u \cdot v_a)\, n/d \qquad t = (v_u \cdot v_c)\, n/d$$

Almost there

- glFrustum assumes that the viewer is perpendicular to the screen
- We need two more capabilities:
  - Rotate the screen out of the XY plane
  - Correctly position it relative to the user

Projection Plane Orientation

- We want to transform the screens XY plane to be aligned with the viewer XY plane
- M: maps into screen coordinates
- Want to go from screen coordinates to viewer so:
  - Use inverse of screen coordinate system (M)
  - Note: $M^{-1} = M^{T}$ since M is orthogonal

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

View Point Offset

- Need to account for eye offset
  - Reposition the center
- Can be accomplished using the OpenGL function glTranslatef (or glm::translate)

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

Generalized Perspective Projection

- Finally, all put together:

$$P' = PM^T T$$

- A sample implementation of the perspective matrix:
  - http://csc.lsu.edu/~kooima/articles/genperspective/

Helpful References

- Framebuffers
  - https://learnopengl.com/Advanced-OpenGL/Framebuffers
  - http://www.songho.ca/opengl/gl_fbo.html
- Render to Texture
  - http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/
- Generalized Perspective Projection
  - http://csc.lsu.edu/~kooima/articles/genperspective/

QUESTIONS?