Discussion 4

Render To Texture - GL
Generalized Perspective Projection

Rendering to Texture

- We will need to render different views to different screens
- To do this, we will render each screen as a texture, then paste it onto a quad

Framebuffers

- Framebuffers allow us to render to places other than the screen we see.
- Framebuffers hold textures we can use later

```
GLuint fbo= 0;

glGenFramebuffers(1, &fbo  );

glBindFramebuffer(GL_FRAMEBUFFER, fbo);
```

Textures

- We will need a texture to hold what we are going to draw on our screen

```
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

- We need depth testing, but we don’t need to render the depth information
- Renderbuffers are more optimized than textures if you don’t need access

```
GLuint rboId;
glGenRenderbuffers(1, &rboId);
glBindRenderbuffer(GL_RENDERBUFFER, rboId);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT,
                      TEXTURE_WIDTH, TEXTURE_HEIGHT);
glBindRenderbuffer(GL_RENDERBUFFER, 0);
```

Renderbuffers (cont.)

- Attach to the framebuffer similarly to our texture

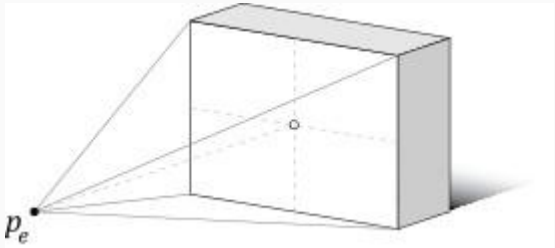
```
glFramebufferRenderbuffer(GL_FRAMEBUFFER,      // 1. fbo target: GL_FRAMEBUFFER
                          GL_DEPTH_ATTACHMENT, // 2. attachment point
                          GL_RENDERBUFFER,     // 3. rbo target: GL_RENDERBUFFER
                          rboId);              // 4. rbo ID
```

Rendering

- Just render as normal with the new framebuffer bound
- Then render the resulting texture to a quad that represents the screen with the default framebuffer bound

Generalized Perspective Projection

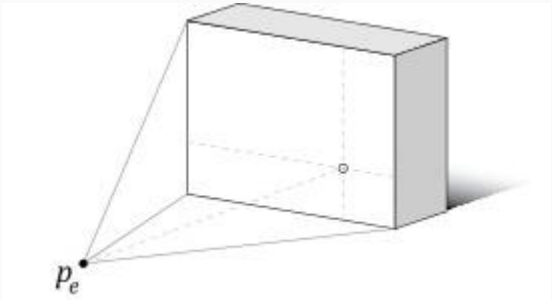
Typically, the projection matrix we use (generated by gluPerspective) is on-axis.

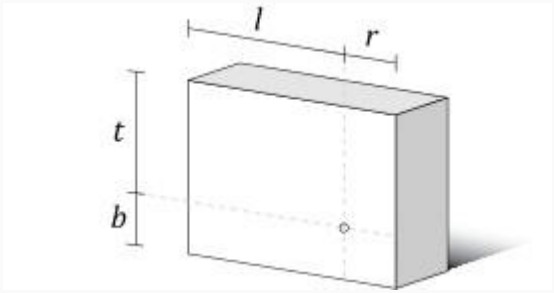


Off-axis perspective

In a CAVE, we cannot view every screen head on, so each screen needs a different perspective.

glFrustum can generate the perspective matrix for us given several parameters

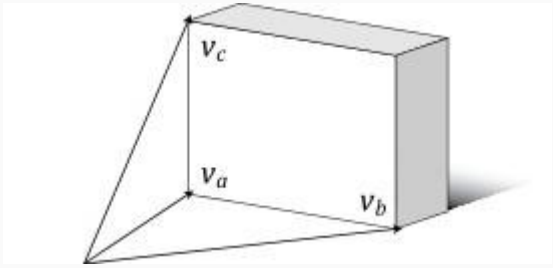




(Not shown: near and far plane parameters)

Calculating frustum parameters

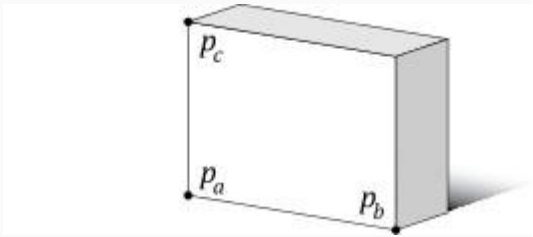
1. Find vectors from eye to corners

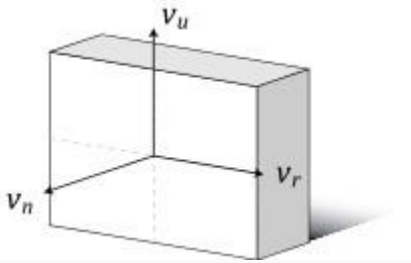


$$v_a = p_a - p_e \qquad v_b = p_b - p_e \qquad v_c = p_c - p_e$$

2. Find distance from eye to plane

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





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

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

3. Find extents, scaling to the near plane

Almost there

Still need to:

- Rotate the screen out of the XY plane
- Position it relative to viewer

Screen Orientation

- We want something lying in screen plane to be transformed to XY plane
- Use inverse of screen coordinate system (since they are orthogonal we can use transpose)

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

View Point Offset

- Need to account for eye offset

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

Finally, all put together

$$P' = PM^T T$$

References

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