



# Discussion 8

## CSE 167



# Outline

- FAQ and More OpenGL Quirks
  - Skybox and Textures
  - Disco Ball
- Introduction to Project 4
  - Lighting (Directional lights and Toon Shading)
  - Implementing Collision Detection

Any Questions on  
Project 3?





# So Your Skybox Ain't Displayin'...

Check the following suggestions:

- Can your program render a cube to screen?
- Are you actually parsing all 6 skybox pictures in your code and generating the textures correctly?
- Did you properly activate the relevant texture unit and bind the cubemap texture before the draw call?



# So Your Skybox Ain't Displayin'...

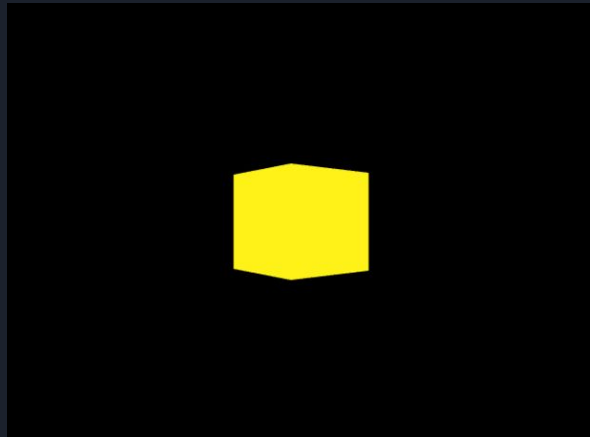
Check the following suggestions:

- Can your program render a cube to screen?
- Are you actually parsing all 6 skybox pictures in your code and generating the textures correctly?
- Did you properly activate the relevant texture unit and bind the cubemap texture before the draw call?



# A Cube to Render

- A skybox looks like a big cube with textures painted over its inside walls
- Render a cube then make it big!
- The starter code from Project 1 has a cube class...





# Using Textures in OpenGL

From the OpenGL Programming Guide v4.3:

Using OpenGL's texture-mapping capabilities requires the following steps

1. Create a texture object and load texel data to it
2. Include texture coordinates with your vertices.
  - a. In this case, the cubemap's vertices serve as the texture coordinates
3. Associate a texture sampler with each texture map in your shader
4. Retrieve the texel values using the texture sampler

*The following slides are taken from [learnopengl.com](http://learnopengl.com) and *The OpenGL Programming Guide v4.3**



# 1) Create a Texture

- Similar to how one would create and store data in a VBO
- Use `glGenTextures` to reserve a name/ID for the texture

```
void glGenTextures(GLsizei n, GLuint *textures);
```

- Use `glBindTexture` to give the texture actual properties
- `glBindTexture(GL_TEXTURE_CUBE_MAP, texID)` to bind a cubemap to a given texture
  - `texID` is a GLuint texture reserved using `glGenTextures`
  - The 1st time this is called on `texID`, it will be assigned a type of `GL_TEXTURE_CUBE_MAP`
  - Subsequent calls on `texID` will activate it
  - Binding to 0 = removing any texture from `GL_TEXTURE_CUBE_MAP`



# 1) (cont'd) Load Data

```
int width, height, nrChannels;
unsigned char *data;
for(unsigned int i = 0; i < textures_faces.size(); i++)
{
    data = stbi_load(textures_faces[i].c_str(), &width, &height, &nrChannels, 0);
    glTexImage2D(
        GL_TEXTURE_CUBE_MAP_POSITIVE_X + i,
        0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data
    );
}
```

- We use stb\_image to load image data into memory.
- glTexImage2D will load data into the texture object.
- Make sure you bind the texture you're going to modify first!

## 2) Associate texture coordinates per vertex

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoords;

out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    TexCoords = aTexCoords;
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```




```
#version 330 core
layout (location = 0) in vec3 aPos;

out vec3 TexCoords;

uniform mat4 projection;
uniform mat4 view;

void main()
{
    TexCoords = aPos;
    gl_Position = projection * view * vec4(aPos, 1.0);
}
```

- For a cubemap, the texture coordinates are 3D vectors.
- If the cubemap is centered at the world origin (0, 0, 0), we can just use the vertex positions!
- Otherwise, you may have to load in a VBO of texture coordinates



## 3) - 4) Use a Texture Sampler to get Texel Data

Retrieve texel value using texture coordinates

```
#version 330 core
out vec4 FragColor;

in vec3 TexCoords;

uniform samplerCube skybox;

void main()
{
    FragColor = texture(skybox, TexCoords);
}
```

Tell OpenGL how the Texture Sampler deals with “edge cases”

```
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
```



# But What About LearnOpenGL's Tutorial?

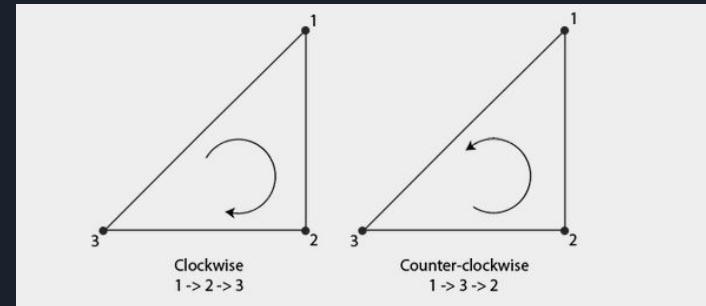
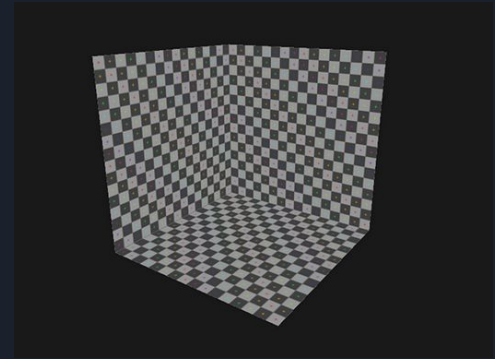
- What is `glDepthMask(GL_FALSE)`?
  - It disables the z-buffer algorithm, meaning that the cube will just write over the entire frame regardless of distance to the camera.
  - **We're using a really big cube so we don't use it.**
- Why are they using a small cube?
  - They use the depth mask trick and drawing the cube to the color buffer first. Any subsequent draw call to another object will just overwrite the pixels
  - Removing the translation part of the view matrix using `mat4(mat3(view))` means that the cube is always rendered as if the camera was centered at (0,0,0)
- What about `glDepthFunc`?
  - It's part of the optimized implementation of the skybox. It sets how depth values are compared.
  - The optimization trick is pretty cool but **you don't need to know about it**

# Skybox Culling

- To use single-sided rendering, call following functions
  - `glEnable(GL_CULL_FACE);`
  - `glCullFace(GL_FRONT);`

OR

- `glEnable(GL_CULL_FACE);`
- `glCullFace(GL_BACK);`
- Should be called before you call draw your skybox

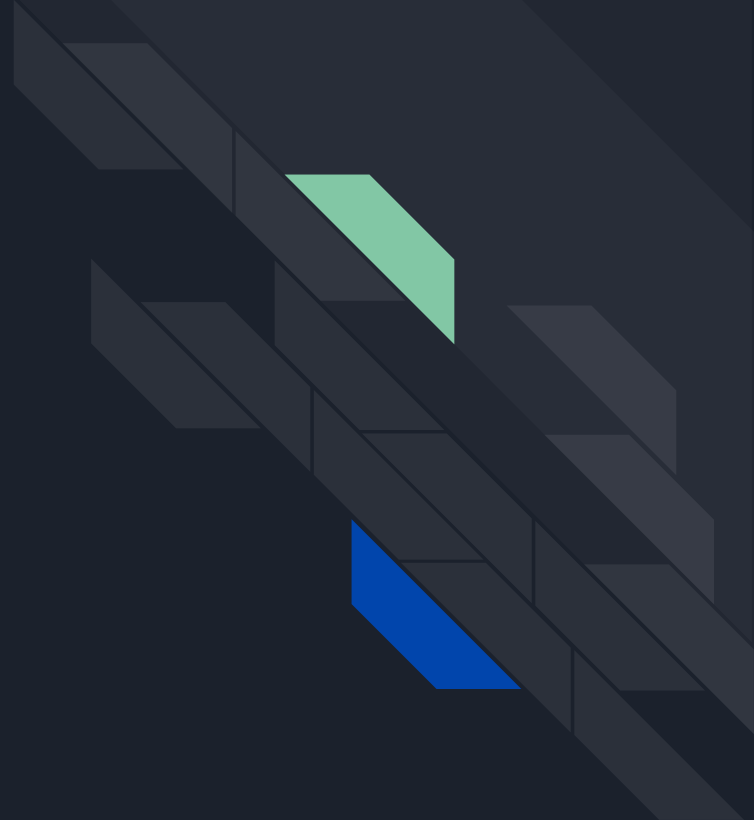


# Disco Ball Reflections

- Reflections look off?
  - Try calling `glDisable(GL_CULL_FACE)` right after you draw your skybox, so it does not interfere with other objects being drawn



Project 4   
Among Us in 167



Recommended Settings  
Map: Polus  
# Impostors: 1 (Limit: 0)  
Confirm Ejects: On  
# Emergency Meetings: 1  
Emergency Cooldown: 15s  
Discussion Time: 15s  
Voting Time: 120s  
Player Speed: 1x  
Crewmate Vision: 1x  
Impostor Vision: 1.5x  
Kill Cooldown: 45s  
Kill Distance: Normal  
Visual Tasks: On  
# Common Tasks: 1  
# Long Tasks: 1  
# Short Tasks: 2

Ping: 60 ms



PUBLIC

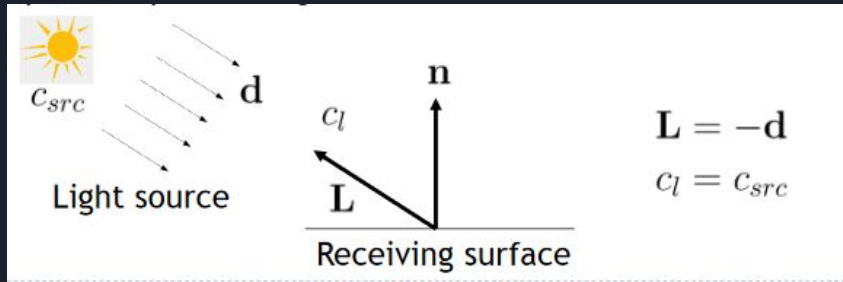
Code  
HV FJ RQ





# Directional Lights

- Light from a certain direction
- Passing in light direction to shader, as opposed to light position
  - No attenuation (light is infinitely far away)
  - Remember to negate passed in direction before using in calculations ( $\mathbf{L} = -\mathbf{d}$ )
  - <https://learnopengl.com/Lighting/Multiple-lights>



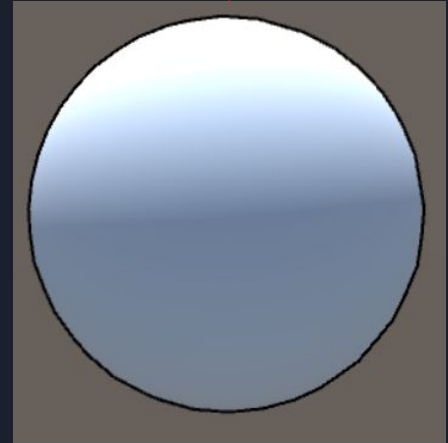
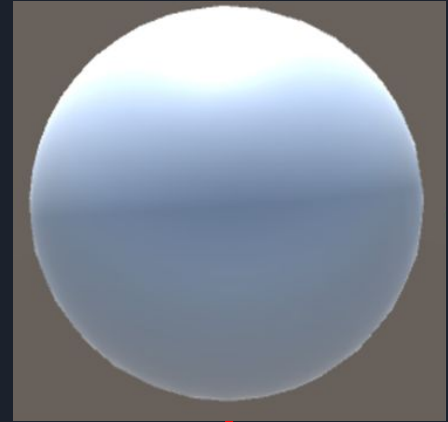
# Toon Shading

- Silhouette edge detection
- Discretize shading



# Silhouette Edge Detection

- Gives black outline to edges of your obj
  - Emphasize pixels with normals perpendicular to viewing direction.
- $\text{Edge} = \max(0, \text{dot}(n, v))$ ;
  - $n$  = normal
  - $v$  = viewing direction
- If  $\text{Edge} < 0.01$ , draw black.



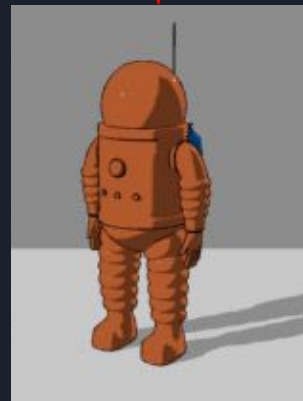
# Discretize Shading

- Create thresholds to create fewer shades, creating a cartoonish look

Intensity: Calculate diffuse and specular, then multiply them together

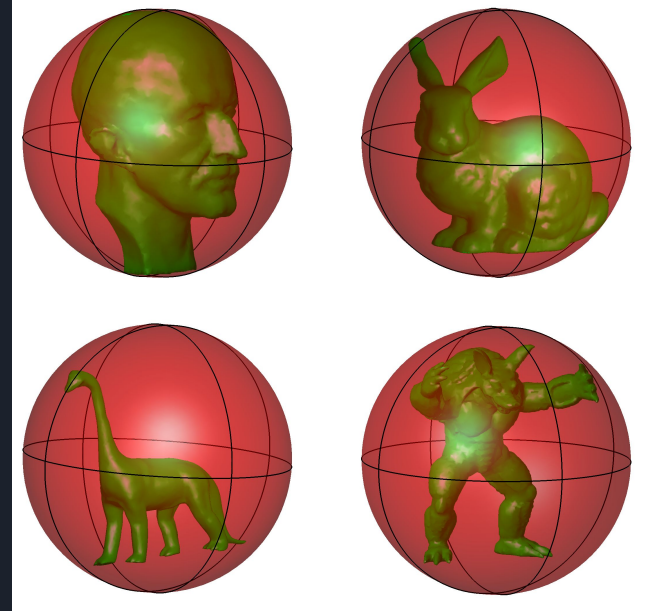
$$\text{diffuse} = \mathbf{n} \cdot \mathbf{L} \quad \text{specular} = (\mathbf{n} \cdot \mathbf{h})^s$$

```
if (intensity > 0.95)
    color = float4(1.0, 1, 1, 1.0) * color;
else if (intensity > 0.5)
    color = float4(0.7, 0.7, 0.7, 1.0) * color;
else if (intensity > 0.05)
    color = float4(0.35, 0.35, 0.35, 1.0) * color;
else
    color = float4(0.1, 0.1, 0.1, 1.0) * color;
```



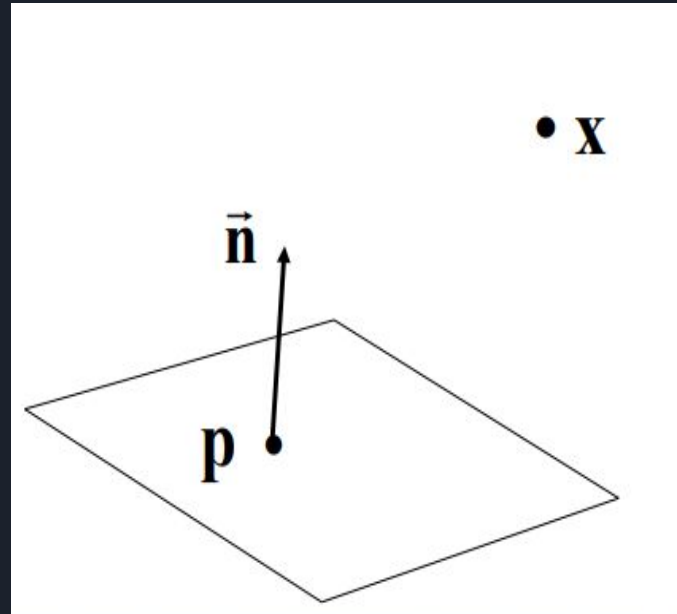
# Bounding Spheres

- Encase the entire object in a tight sphere
- Pros
  - Easy to understand
  - Sphere/sphere & sphere/plane intersection testing inexpensive and simpler to implement
- Cons
  - Not a snug fit for the objects => inaccuracy compared to bounding boxes or comparing each individual triangle
- Just need two pieces of info
  - Radius
  - Center



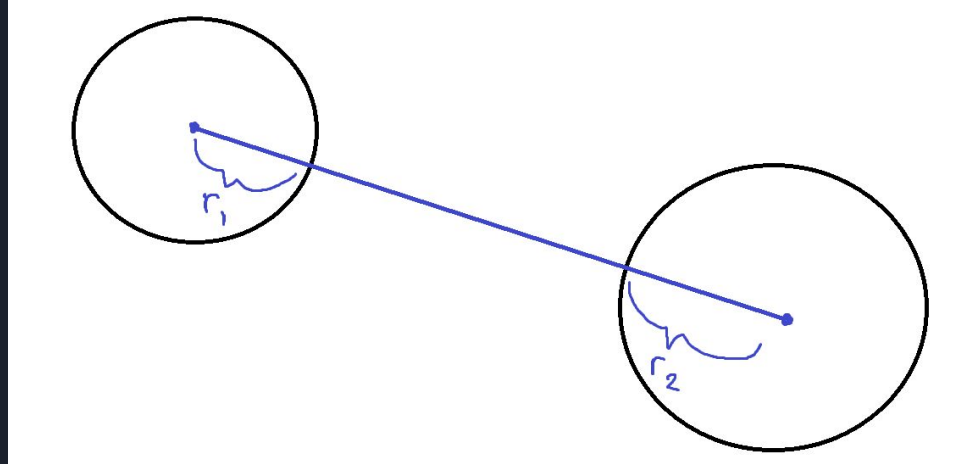
# Bounding Plane

- Can be represented by a normal vector  $\underline{n}$  and a distance from origin to plane  $\text{dot}(\underline{p}, \underline{n})$  where  $\underline{p}$  is some point on the plane
- 6 of these make a bounding box



# Sphere-Sphere Collisions

- Simple
- If the distance between the two centers is  $< r_1 + r_2$ , then we have an intersection!





# Sphere-Plane Intersection

- Essentially:
  - a. Plug center into point-plane distance formula (see Lecture 13: Visibility Culling)
  - b. If  $\text{dist} \leq r$ , we have an intersection!