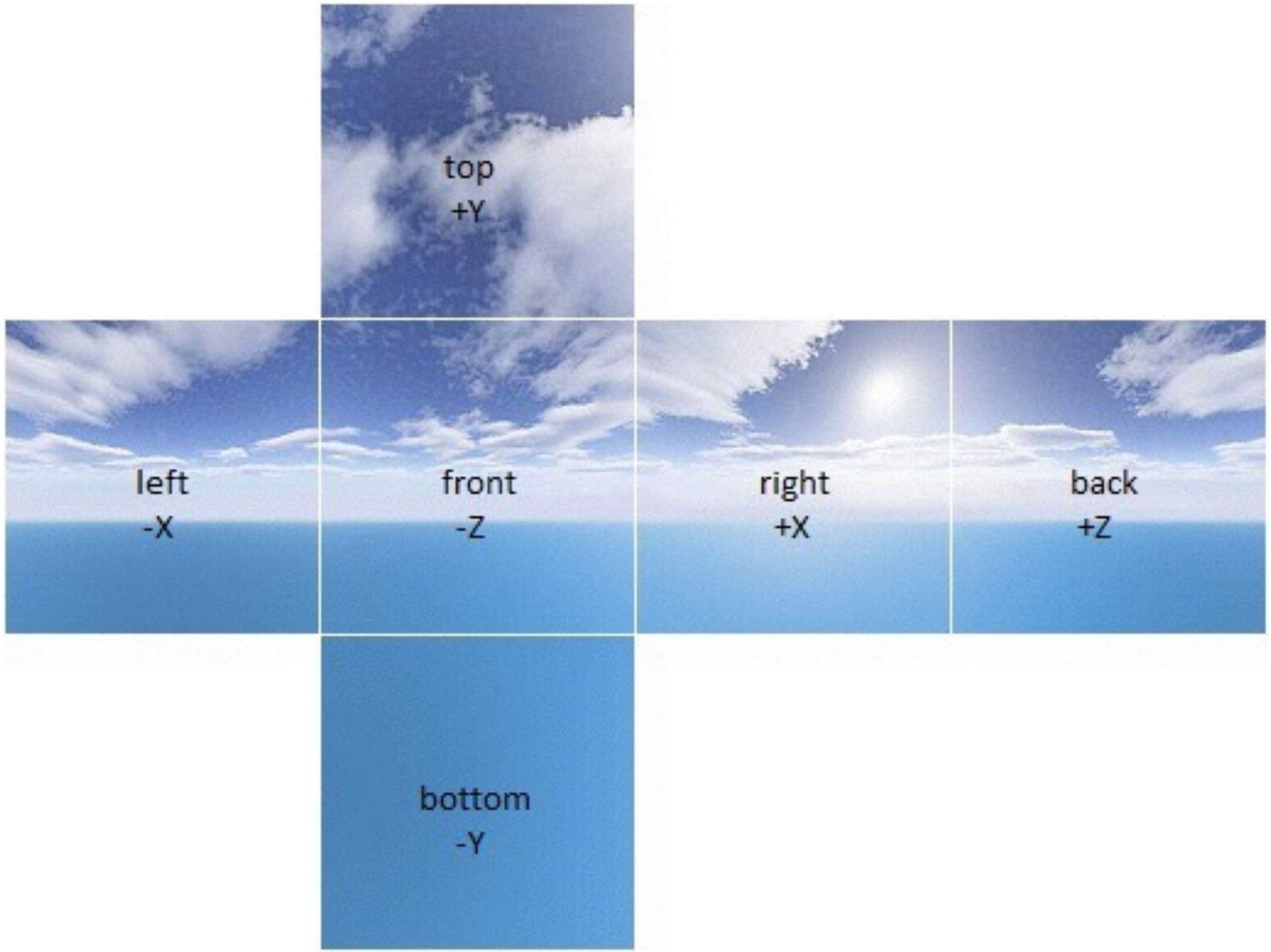
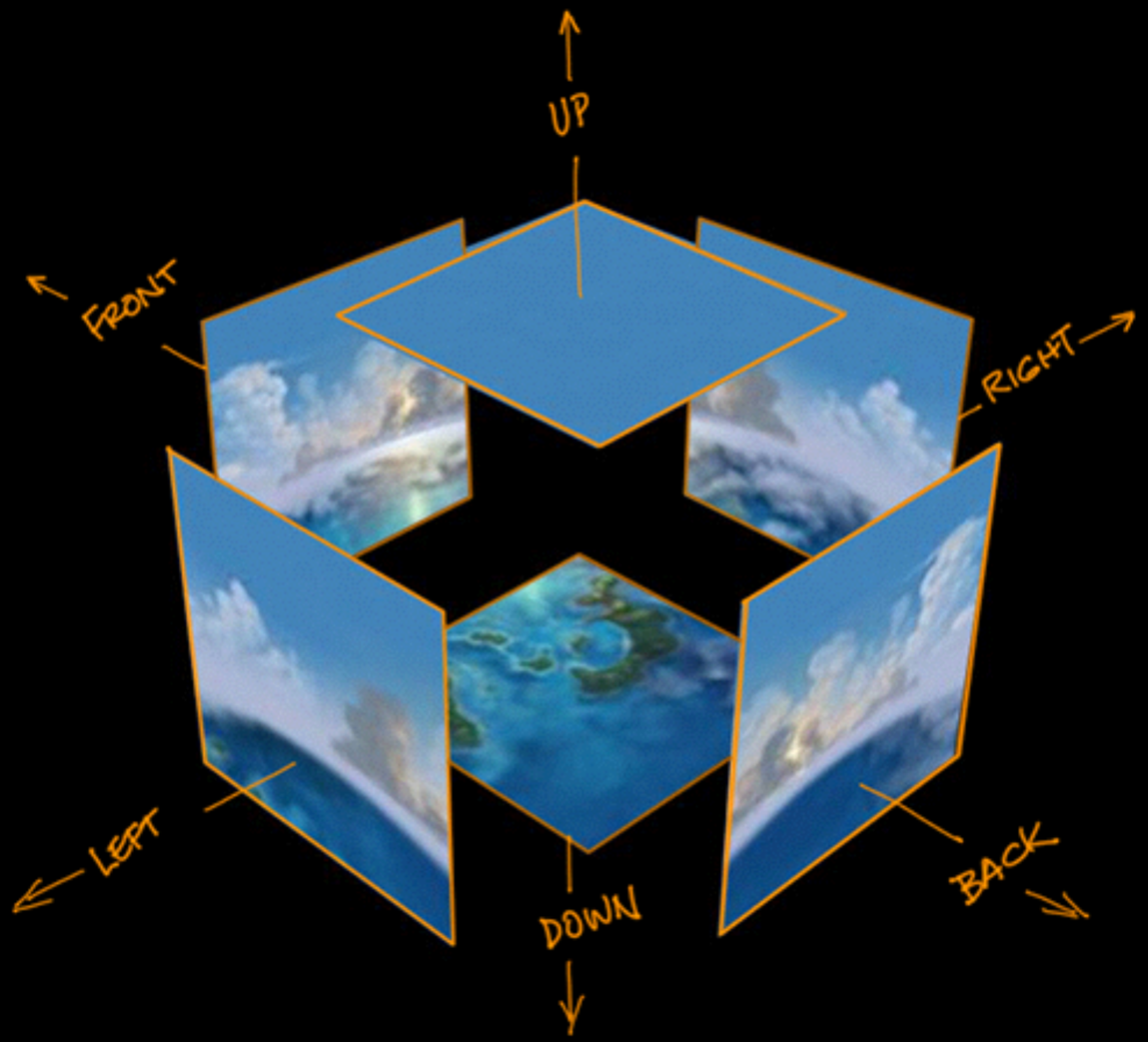
Reflections, refractions and shadows

CS GY-6533 / UY-4533

Cubemaps


top
+Y
left
-X
front
-Z
right
+X
back
+Z
bottom
-Y


UP
FRONT
RIGHT
LEFT
DOWN
BACK

```
GLuint loadGLCubemap(std::vector<std::string> faces) {
    GLuint textureID;
    glGenTextures(1, &textureID);
    glActiveTexture(GL_TEXTURE0);

    glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);
    for(GLuint i = 0; i < faces.size(); i++) {
        int w,h,comp;
        unsigned char* image = stbi_load(faces[i].c_str(), &w, &h, &comp, STBI_rgb_alpha);
        if(image) {
            glTexImage2D(
                    GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0,
                    GL_RGB, w, h, 0, GL_RGBA, GL_UNSIGNED_BYTE, image
                    );
            stbi_image_free(image);
        }
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_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);
    glBindTexture(GL_TEXTURE_CUBE_MAP, 0);

    return textureID;
}
```

```
GLuint cubeMap;

// . . .

    std::vector<std::string> cubemapFiles;
    cubemapFiles.push_back("alps_ft.tga");
    cubemapFiles.push_back("alps_bk.tga");
    cubemapFiles.push_back("alps_up.tga");
    cubemapFiles.push_back("alps_dn.tga");
    cubemapFiles.push_back("alps_rt.tga");
    cubemapFiles.push_back("alps_lf.tga");

    cubeMap = loadGLCubemap(cubemapFiles);

// . . .

glBindTexture(GL_TEXTURE_CUBE_MAP, cubeMap);
```

Rendering a skybox

Render a large cube and sample the cubemap at the direction of the fragment.

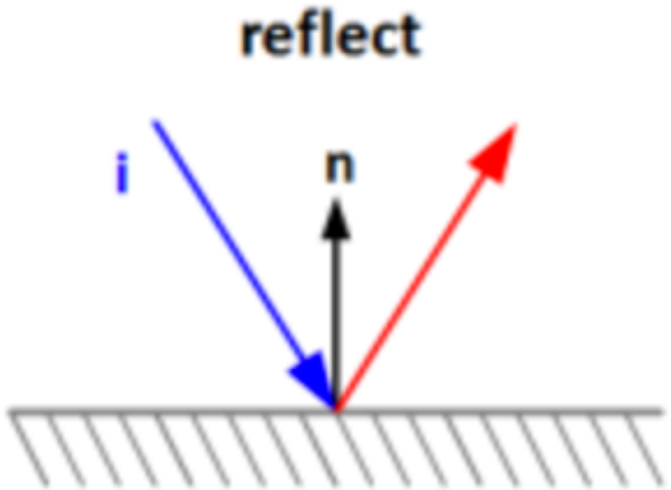
```
varying vec3 varyingPosition;
uniform samplerCube environmentMap;

void main() {
    gl_FragColor = textureCube(environmentMap, normalize(varyingPosition));
}
```

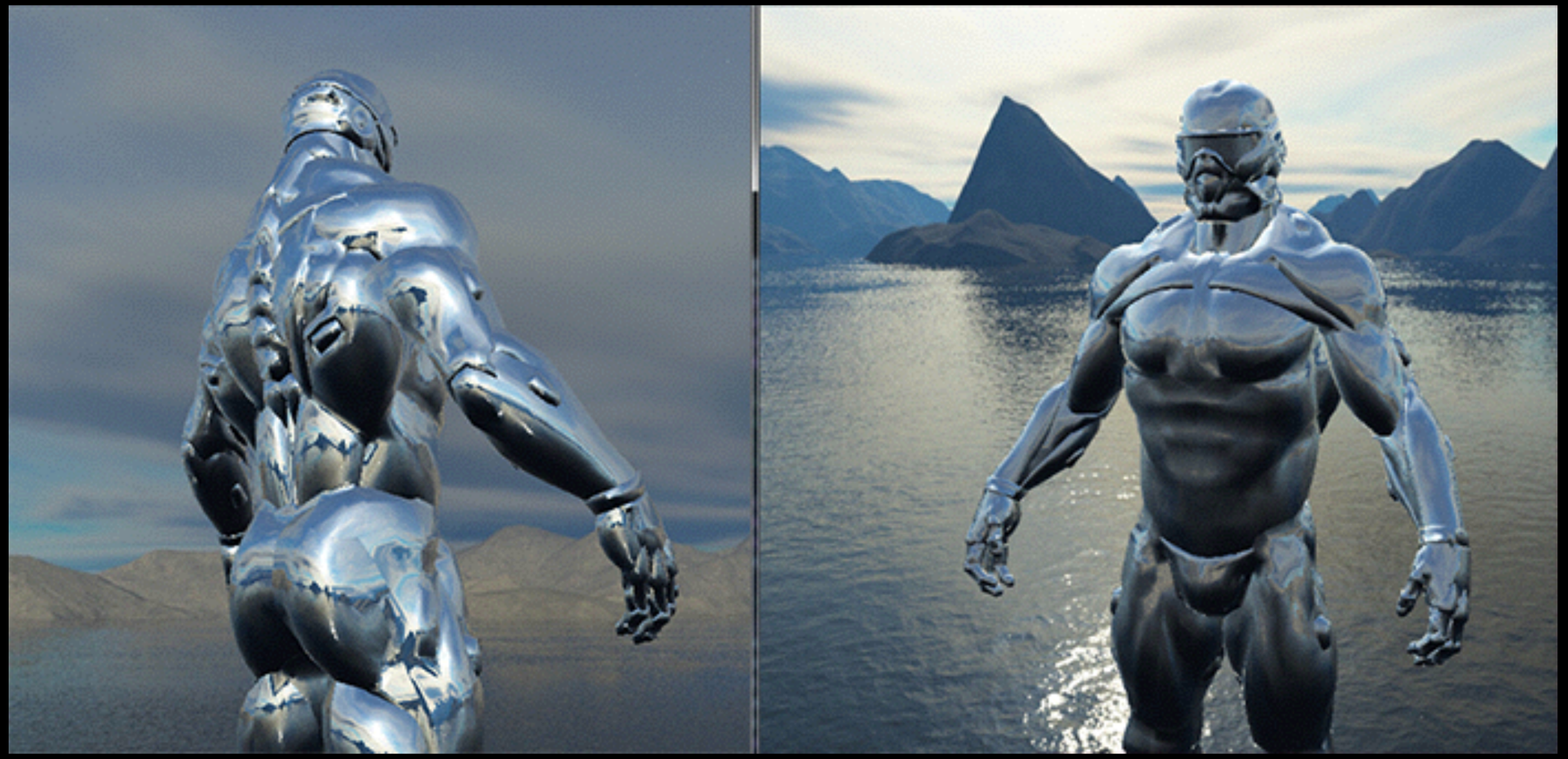
Note: varyingPosition in this case is **not in eye space**!

Make sure that culling is off when rendering
the skybox if rendering a regular outward facing cube!

```
glDisable(GL_CULL_FACE);
```

Cubemap reflection


reflect
i
n

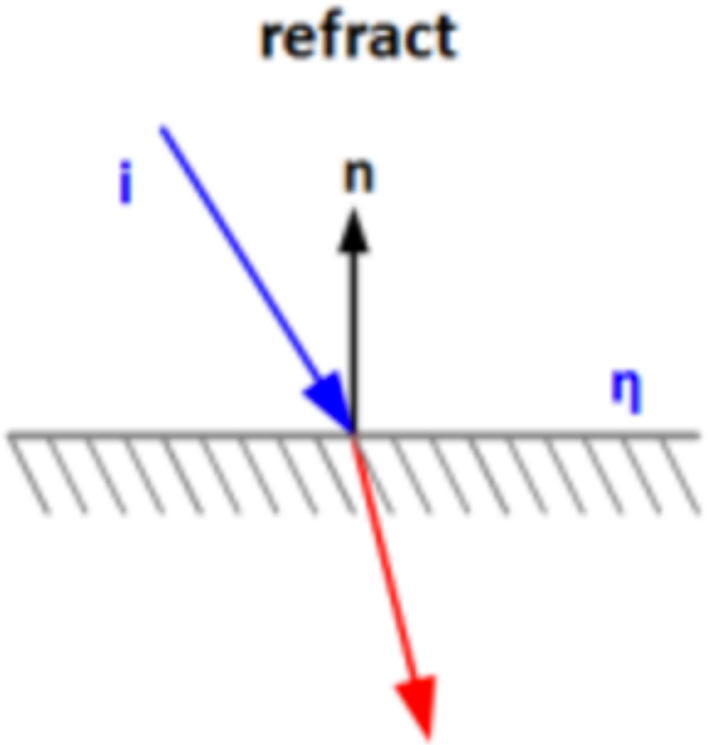


```
varying vec3 varyingNormal;
varying vec3 varyingPosition;

uniform mat4 normalMatrix;

uniform samplerCube environmentMap;

mat3 linearTranspose(mat4 m4) {
  return mat3(
      m4[0][0], m4[1][0], m4[2][0],
      m4[0][1], m4[1][1], m4[2][1],
      m4[0][2], m4[1][2], m4[2][2]);
}

void main() {
    vec3 reflectVector = reflect(normalize(varyingPosition), varyingNormal);
    reflectVector = linearTranspose(normalMatrix) * reflectVector;
    gl_FragColor = textureCube(environmentMap, reflectVector);
}
```

Need to convert reflected vector into world space

(alternatively, we can do this calculation in world space for optimization)

Cubemap refraction

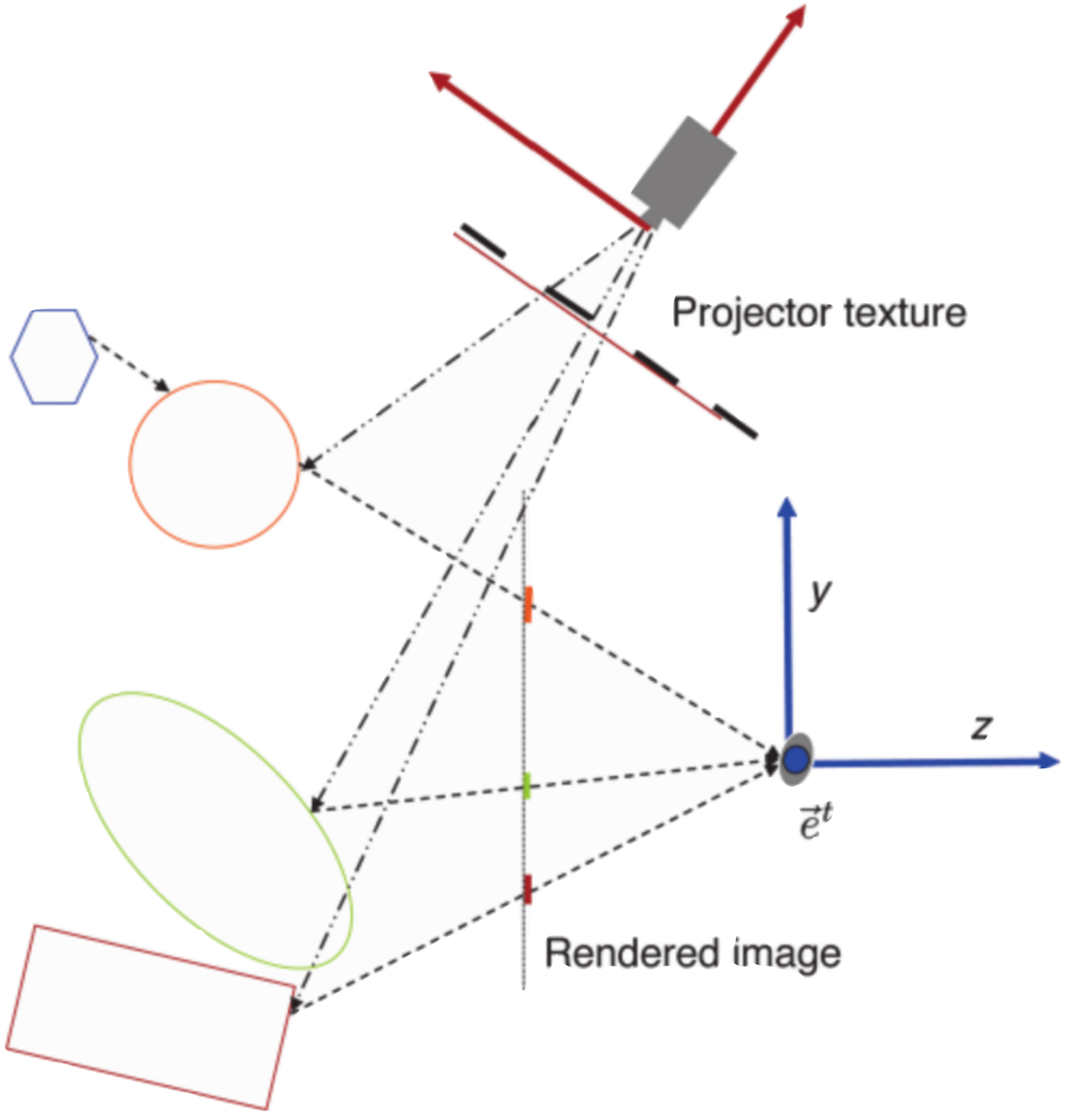

refract
i
n
η

```
varying vec3 varyingNormal;
varying vec3 varyingPosition;

uniform mat4 normalMatrix;

uniform samplerCube environmentMap;

mat3 linearTranspose(mat4 m4) {
  return mat3(
      m4[0][0], m4[1][0], m4[2][0],
      m4[0][1], m4[1][1], m4[2][1],
      m4[0][2], m4[1][2], m4[2][2]);
}

void main() {
    vec3 refractVector = refract(normalize(varyingPosition), varyingNormal, 0.9);
    refractVector = linearTranspose(normalMatrix) * refractVector;
    gl_FragColor = textureCube(environmentMap, refractVector);
}
```

Need to convert reflected vector into world space

(alternatively, we can do this calculation in world space for optimization)

Projection textures



$$\begin{bmatrix} x_t w_t \\ y_t w_t \\ - \\ w_t \end{bmatrix} = P_s M_s \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

$$x_t = \frac{x_t w_t}{w_t} \text{ and } y_t = \frac{y_t w_t}{w_t} .$$

Need to translate projection matrix by (0.5, 0.5, 0.0) and scale by (0.5, 0.5, 1.0) to get into UV coordinate space.

```
attribute vec4 position;
attribute vec2 texCoord;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

uniform mat4 projectorModelviewMatrix;
uniform mat4 projectorProjectionMatrix;

varying vec2 varyingTexCoord;
varying vec4 projectedTexCoord;

void main() {
  varyingTexCoord = texCoord;
  projectedTexCoord = projectorProjectionMatrix * projectorModelviewMatrix * position;
  gl_Position = projectionMatrix * modelViewMatrix * position;
}
```

```
varying vec2 varyingTexCoord;
varying vec4 projectedTexCoord;

uniform sampler2D diffuseTexture;
uniform sampler2D projectedTexture;

void main() {
  vec2 tex2;
  tex2.x = projectedTexCoord.x/projectedTexCoord.w;
  tex2.y = projectedTexCoord.y/projectedTexCoord.w;

  vec4 projectorColor = vec4(0.0, 0.0, 0.0, 0.0);

  if(tex2.x < 1.0 && tex2.y < 1.0 && tex2.x > 0.0 && tex2.y > 0.0) {
       projectorColor = texture2D(projectedTexture, tex2);
  }

     gl_FragColor = texture2D(diffuseTexture, varyingTexCoord) + projectorColor;
}
```

Rendering to texture

1. **Generate** and **bind** a frame buffer .

```
GLuint frameBuffer;
glGenFramebuffers(1, &frameBuffer);
glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);
```

2. Create an **empty texture** to render to.

```
GLuint frameBufferTexture;
glGenTextures(1, &frameBufferTexture);
glBindTexture(GL_TEXTURE_2D, frameBufferTexture);
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, 1024, 1024, 0,GL_RGB,
GL_UNSIGNED_BYTE, NULL);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

3. Bind the texture to the FBO

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

4. Create a **depth buffer texture**.

```
GLuint depthBufferTexture;

glGenTextures(1, &depthBufferTexture);
glBindTexture(GL_TEXTURE_2D, depthBufferTexture);
glTexImage2D(GL_TEXTURE_2D, 0,GL_DEPTH_COMPONENT, 1024, 1024, 0,GL_DEPTH_COMPONENT,
GL_UNSIGNED_BYTE, NULL);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

… and bind it to your frame buffer as a depth buffer attachment.

```
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24, 1024, 1024);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
depthBufferTexture, 0);
```

5. Unbind the created frame buffer

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

Rendering to the FBO

Bind the FBO **before doing your rendering, set your viewport to the size of the texture and clear the buffers.**

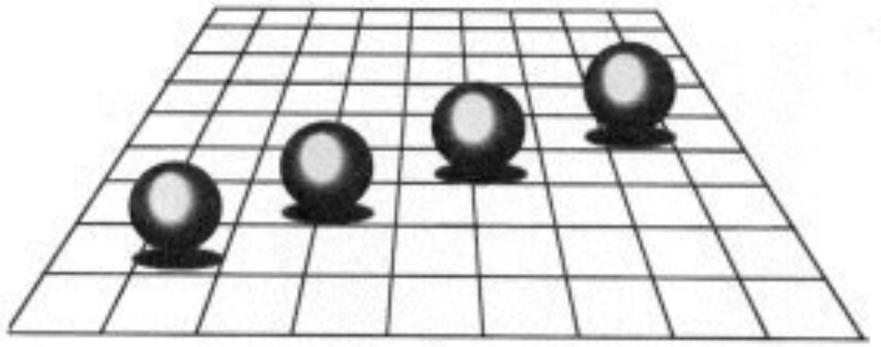
```
glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);
glViewport(0, 0, 1024, 1024);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

Then, do your rendering as usual.

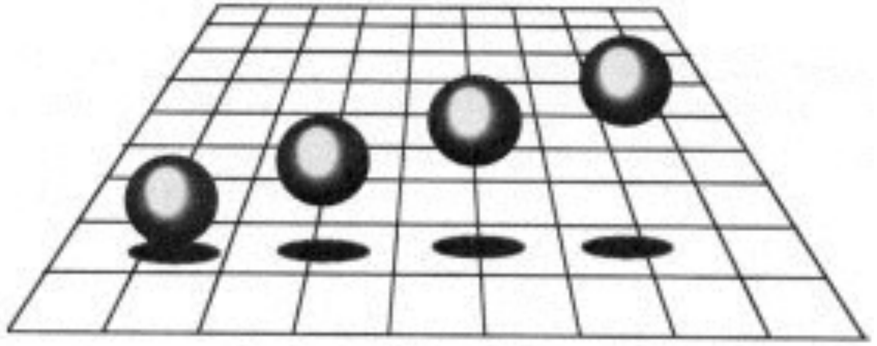
After you’re done, unbind the frame buffer and set viewport to your window size.

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glViewport(0, 0, windowWidth, windowHeight);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

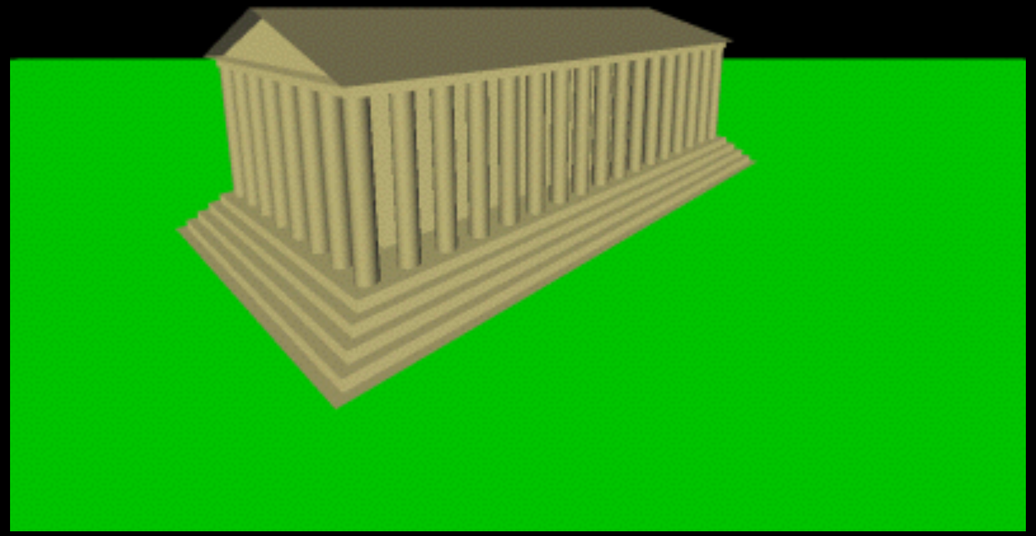
Shadows

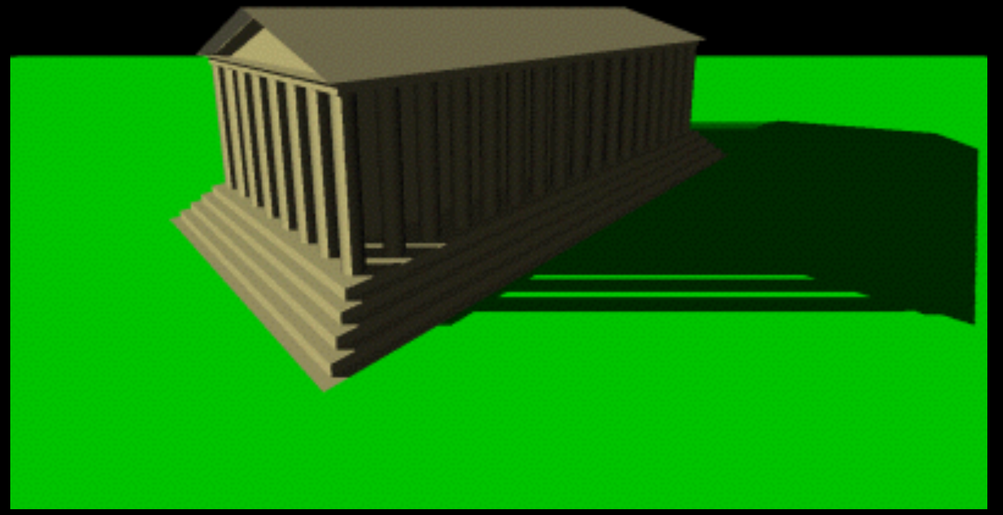


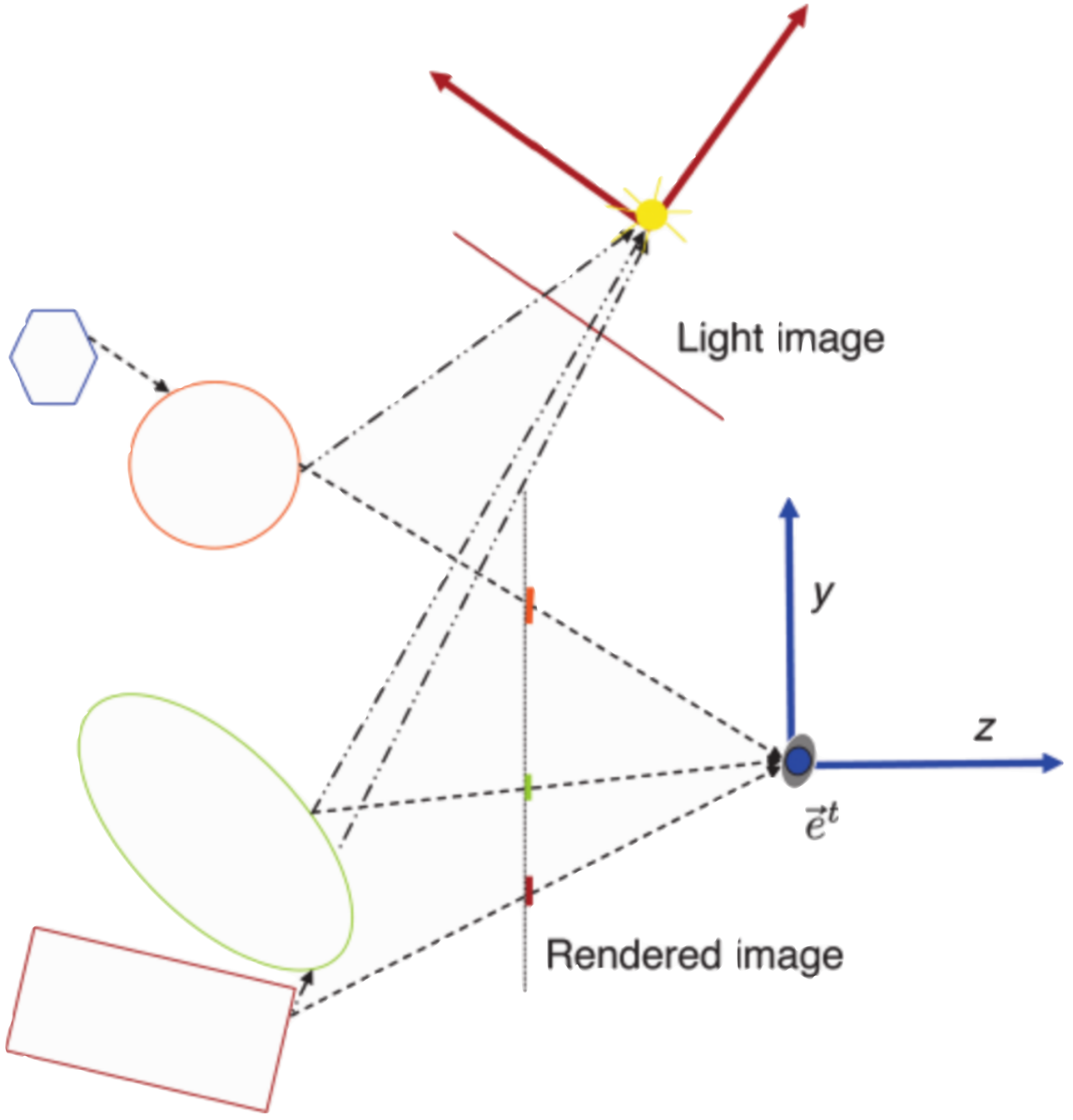
A

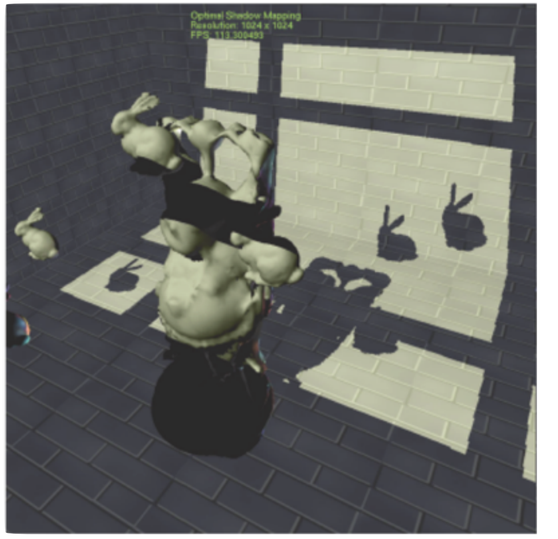


B



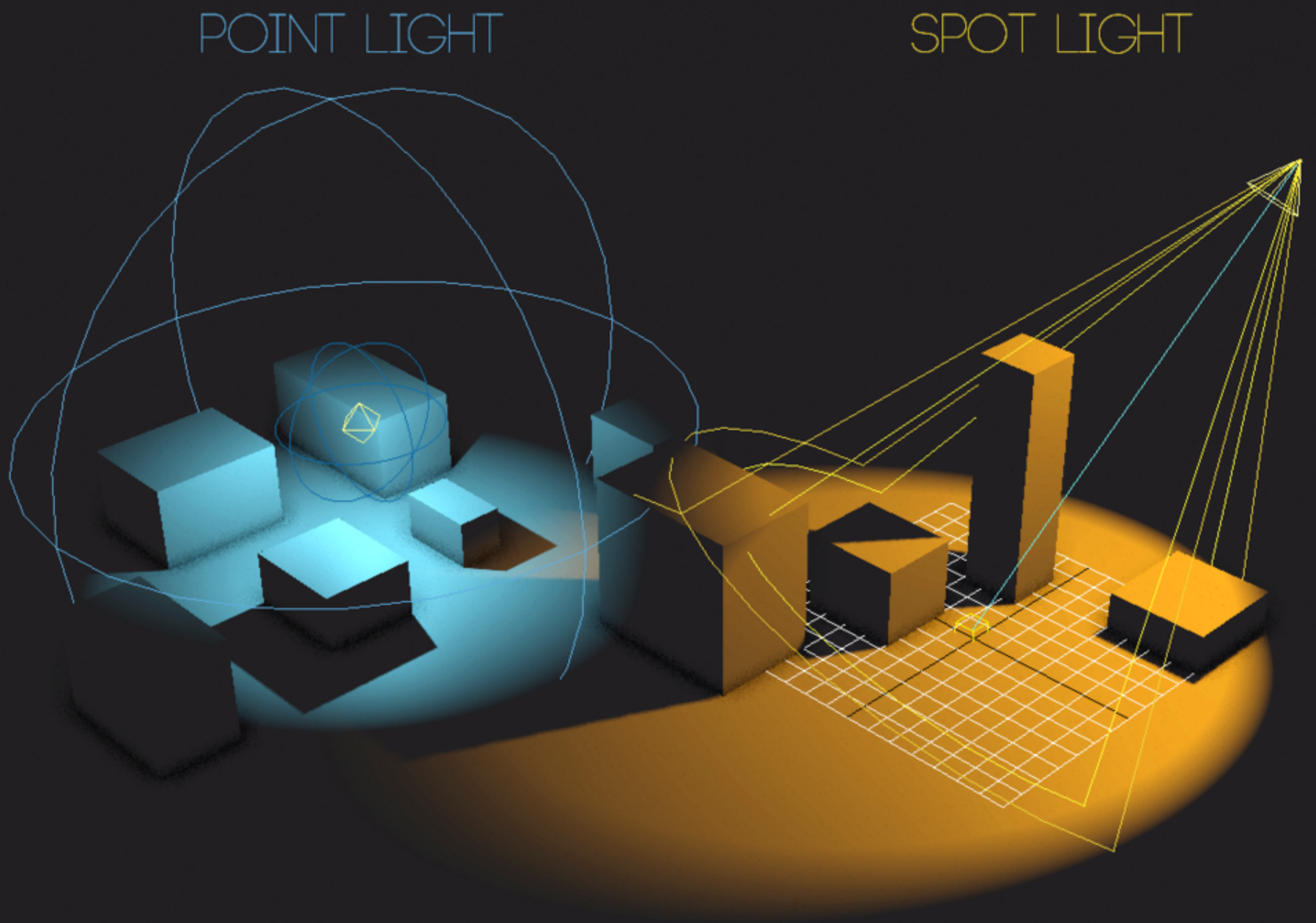


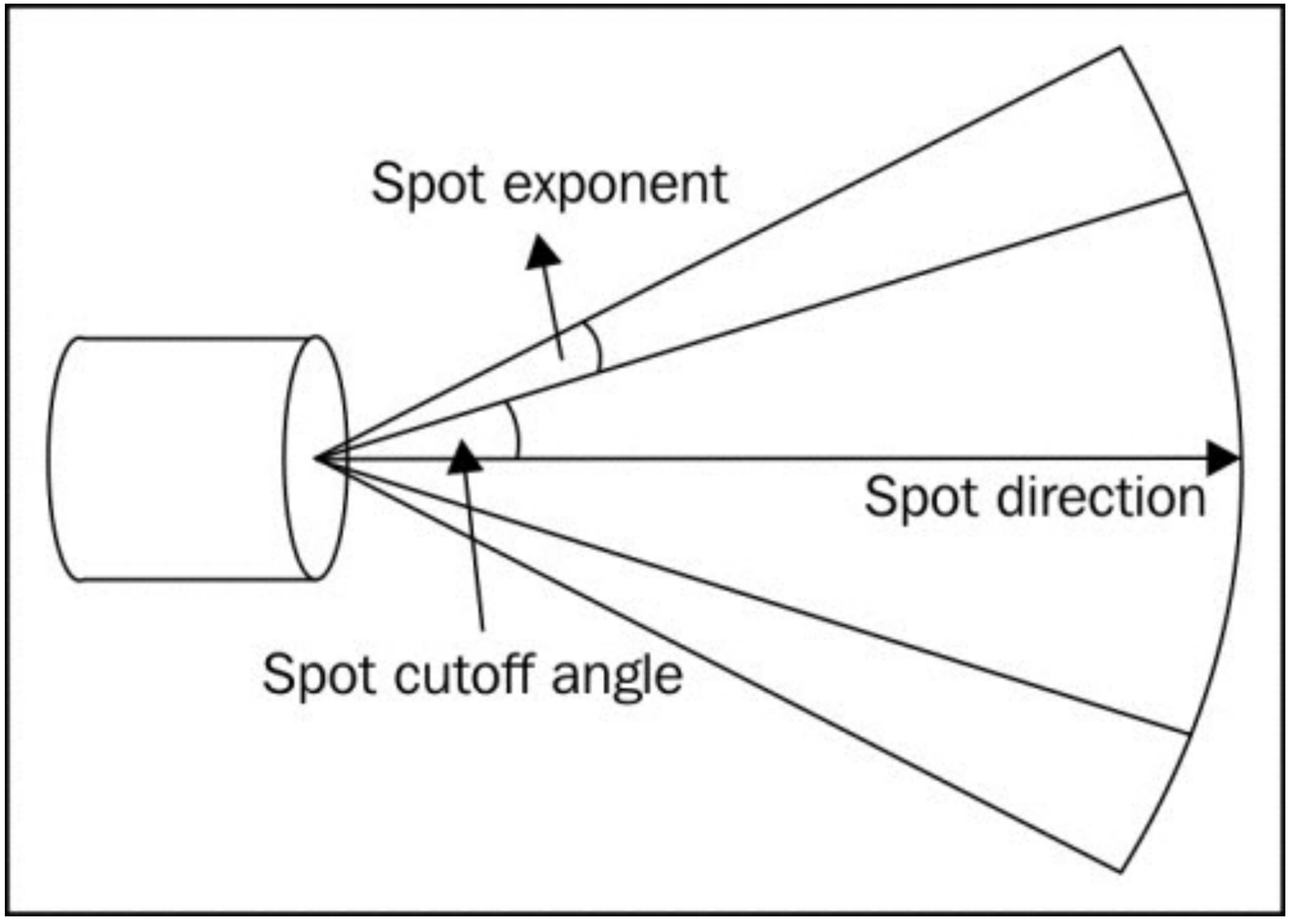

Light image
y
z
$\vec{e}^t$
Rendered image

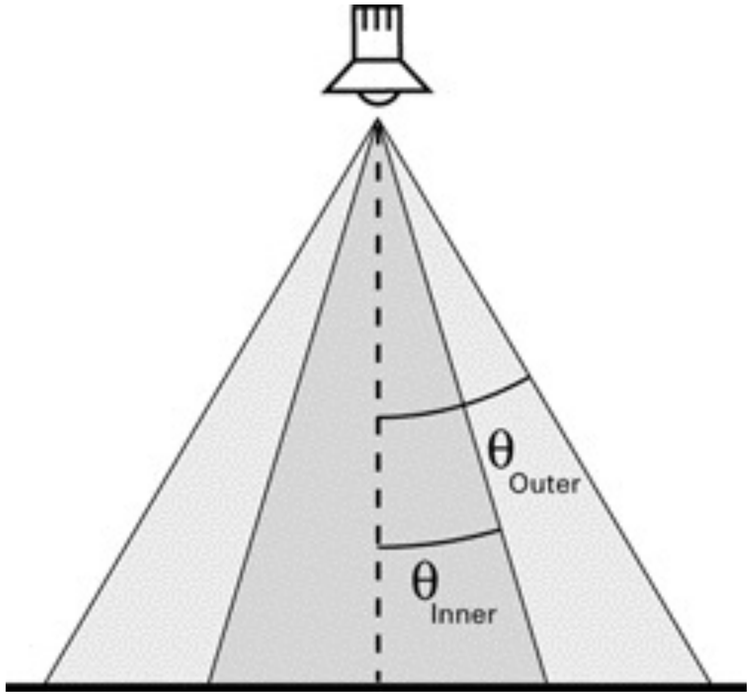


```
varying vec3 varyingNormal;
varying vec3 varyingPosition;
uniform vec3 uColor;

uniform vec3 lightPosition;
uniform vec3 lightColor;
uniform vec3 specularLightColor;

uniform vec3 spotDirection;
uniform float spotExponent;
uniform float spotCosCutoff;

uniform sampler2D shadowTexture;
varying vec4 projectedTexCoord;

float attenuate(float dist, float a, float b) {
    return 1.0 / (1.0 + a*dist + b*dist*dist);
}

void main() {

    vec2 tex2;
    tex2.x = projectedTexCoord.x/projectedTexCoord.w;
    tex2.y = projectedTexCoord.y/projectedTexCoord.w;

    float shadowValue = 1.0;

    if(tex2.x < 1.0 && tex2.y < 1.0 && tex2.x > 0.0 && tex2.y > 0.0) {
        if((texture2D(shadowTexture, tex2).z * 2.0) - 0.000005 > ((projectedTexCoord.z/projectedTexCoord.w))) {
            shadowValue = 0.0;
        }
    }

  vec3 lightDirection = -normalize(varyingPosition-lightPosition);
  float diffuse = max(0.0, dot(varyingNormal, lightDirection));
        float attenuation = attenuate(distance(varyingPosition, lightPosition) / 25.0, 2.7, 5.0);
        vec3 diffuseColor = (lightColor * diffuse) * attenuation;

        vec3 v = normalize(-varyingPosition);
        vec3 h = normalize(v + lightDirection);
        float specular = pow(max(0.0, dot(h, varyingNormal)), 64.0);
        vec3 specularColor = specularLightColor * specular * attenuation;

  vec3 intensity = (uColor * diffuseColor) + specularColor;
    gl_FragColor = vec4(intensity.xyz * shadowValue, 1.0);
}
```

Spotlights




Spot exponent
Spot direction
Spot cutoff angle


θ Outer
θ Inner

Extra credit assignments

- Incorporate OpenVR into a project (https://github.com/ValveSoftware/openvr)

- Incorporate Bullet Physics into a project (http://bulletphysics.org/wordpress/)

- Scan a real-world object and load it into a project (http://www.123dapp.com/catch)