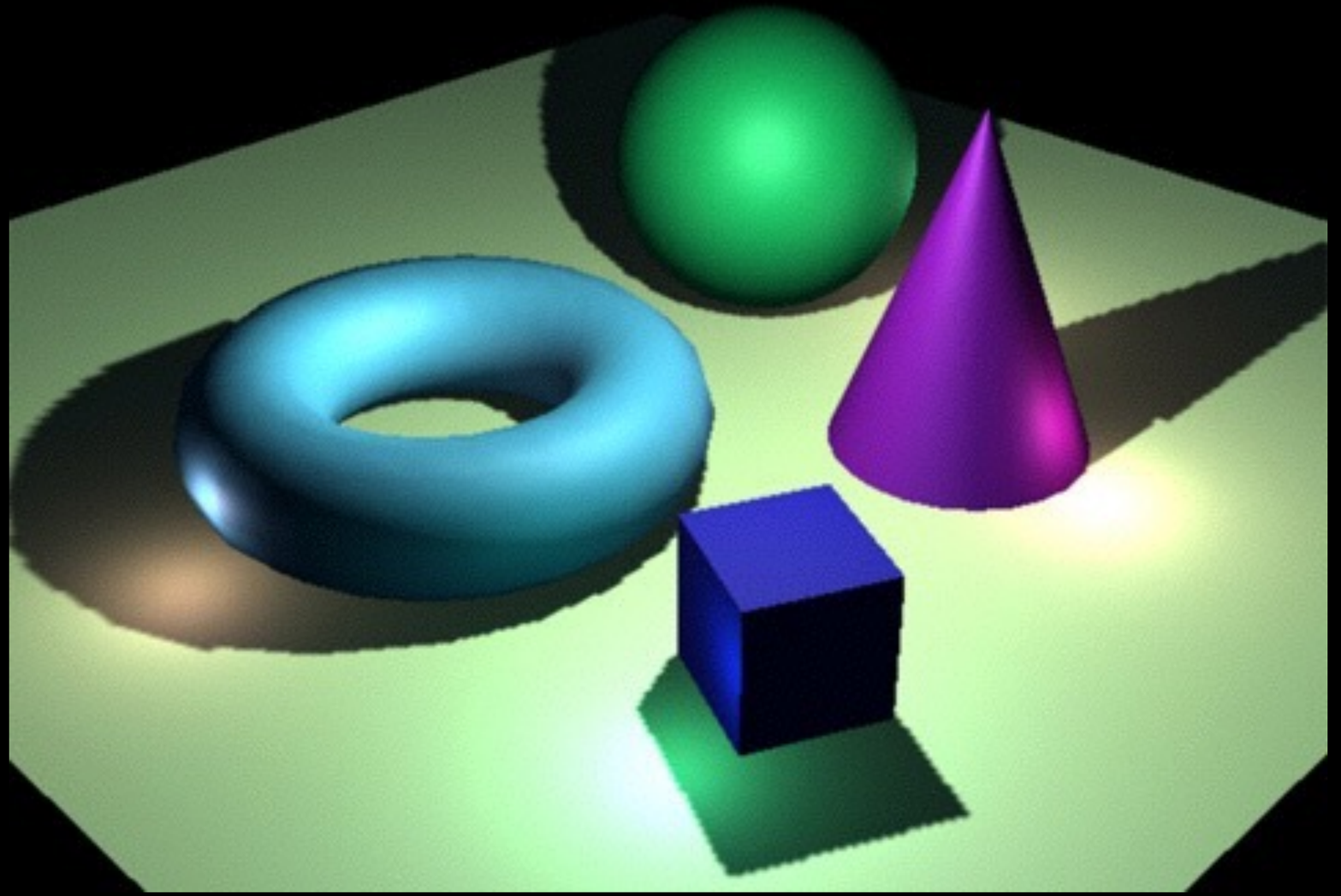
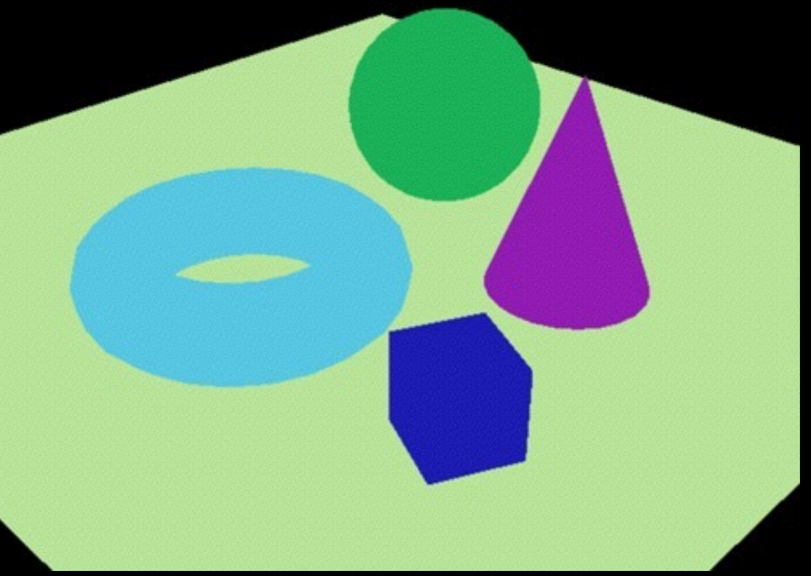
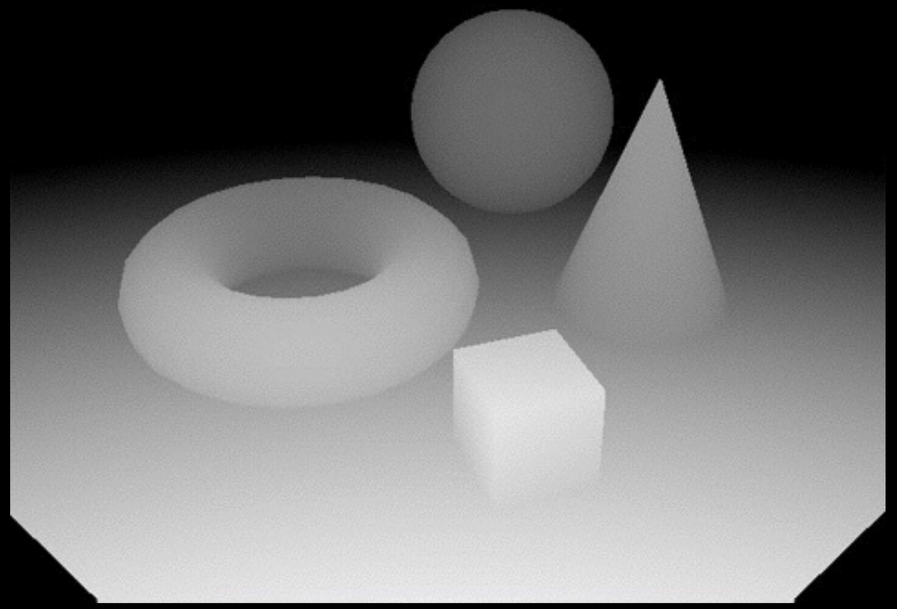
Deferred shading.

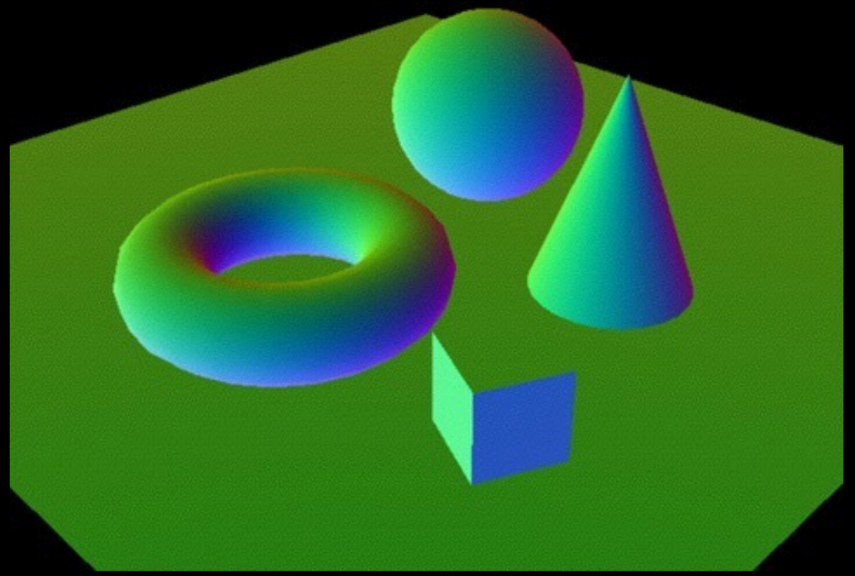
CS GY-6533 / UY-4533

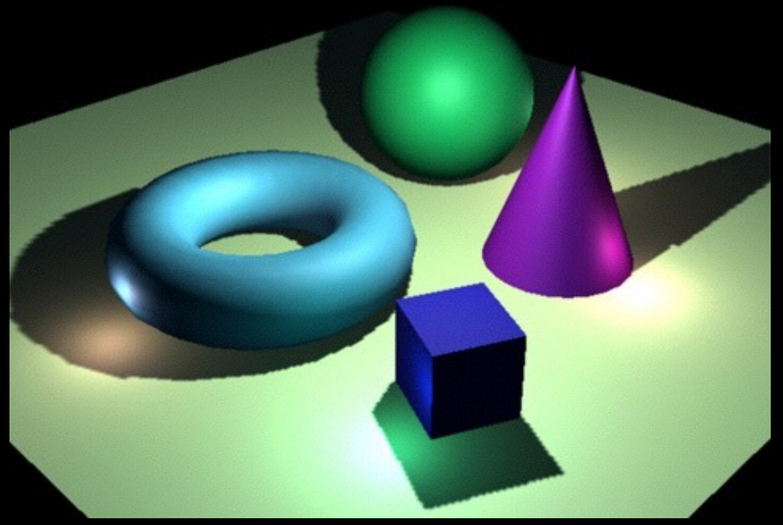
Forward shading vs. deferred shading.












DEPTH
PIXEL'S DISTANCE TO THE CAMERA
NORMALS
PIXEL'S ORIENTATION AND BUMP
ALBEDO
DIFFUSE COLOR AND TEXTURE
SHININESS
INTENSITY OF REFLECTIVE MATERIALS

Rendering the G-Buffers

MRT: Multiple Render Targets.

```
glGenTextures(1, &texture0);
glBindTexture(GL_TEXTURE_2D, texture0);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 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);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texture0, 0);

glGenTextures(1, &texture1);
glBindTexture(GL_TEXTURE_2D, texture1);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 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);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT1, GL_TEXTURE_2D, texture1, 0);

glGenTextures(1, &texture2);
glBindTexture(GL_TEXTURE_2D, texture2);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 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);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT2, GL_TEXTURE_2D, texture2, 0);
```

In OpenGL

```
GLenum buffers[] = { GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1, GL_COLOR_ATTACHMENT2};
glDrawBuffers(3, buffers);
```

In GLSL

```
gl_FragData[0] = // render to texture 0
gl_FragData[1] = // render to texture 1
gl_FragData[2] = // render to texture 2
```

Color G-Buffer

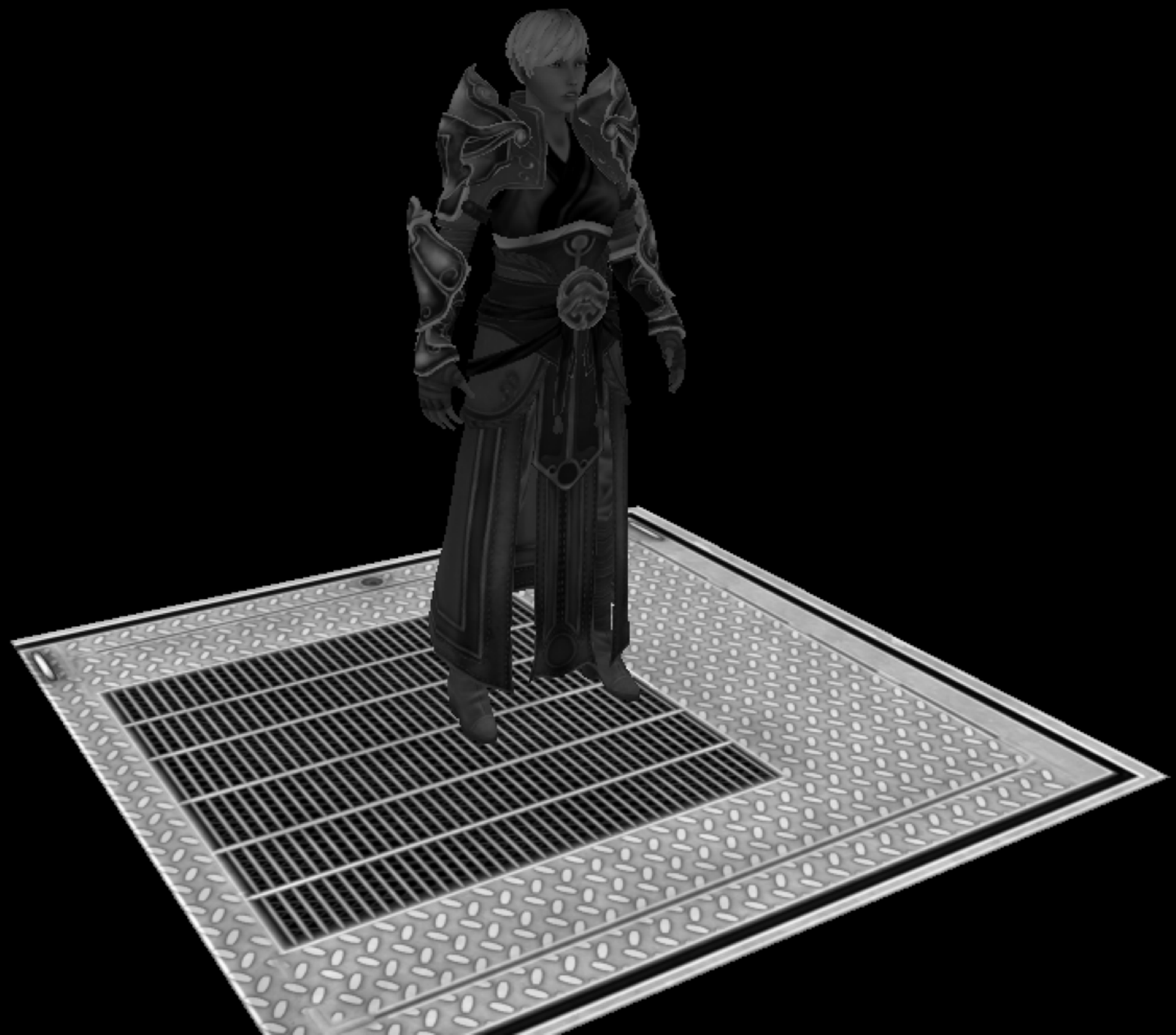
```
varying vec3 varyingNormal;
varying vec2 varyingTexCoord;

uniform sampler2D normalTexture;
uniform sampler2D diffuseTexture;
uniform sampler2D specularTexture;

varying mat3 varyingTBNMatrix;

void main() {
    gl_FragData[0] = texture2D(diffuseTexture, varyingTexCoord);
}
```

Specular G-Buffer



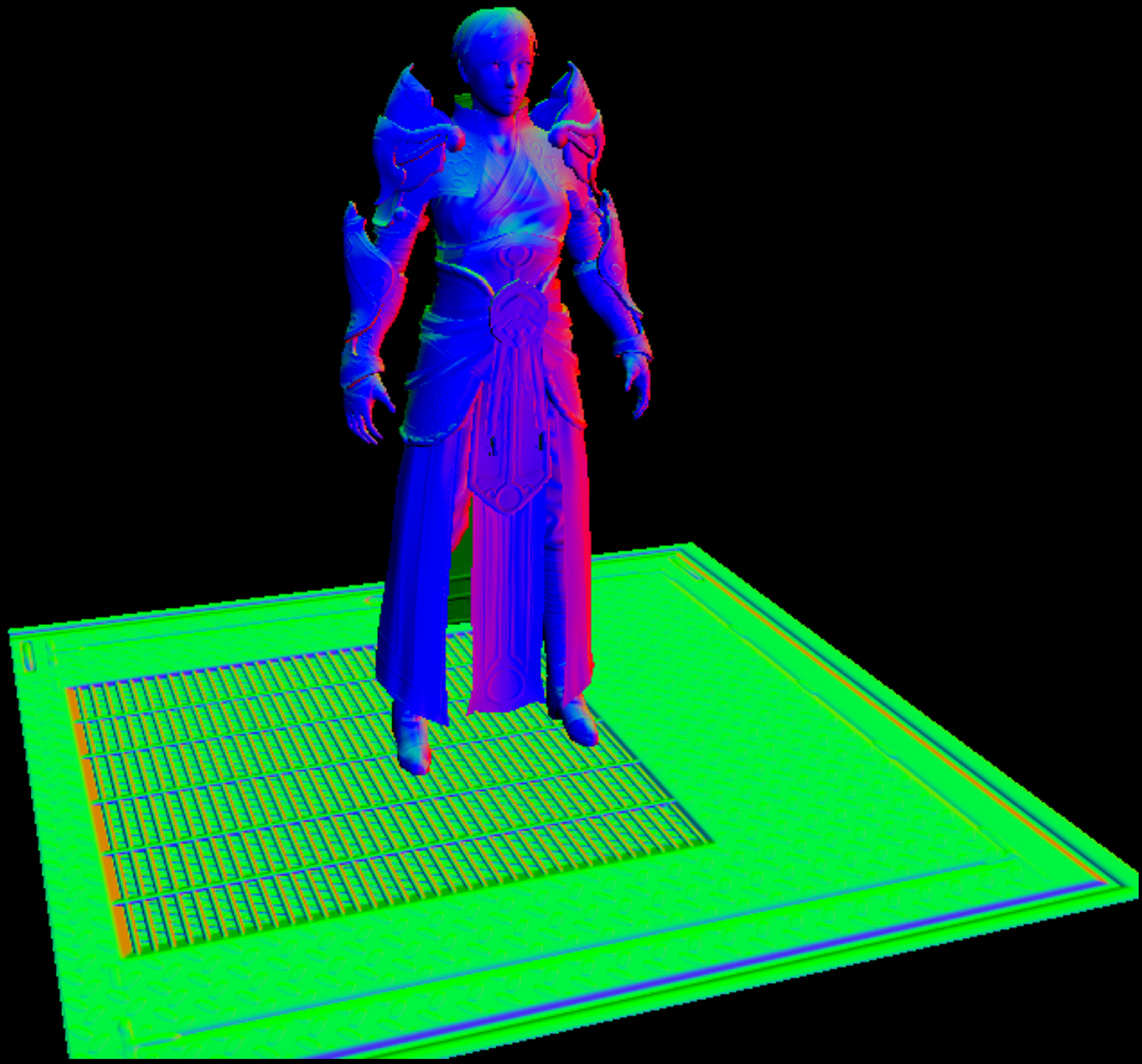
```
varying vec3 varyingNormal;
varying vec2 varyingTexCoord;

uniform sampler2D normalTexture;
uniform sampler2D diffuseTexture;
uniform sampler2D specularTexture;

varying mat3 varyingTBNMatrix;

void main() {
    gl_FragData[0] = texture2D(diffuseTexture, varyingTexCoord);
    gl_FragData[1] = texture2D(specularTexture, varyingTexCoord);
}
```

Normal G-Buffer



```
varying vec3 varyingNormal;
varying vec2 varyingTexCoord;

uniform sampler2D normalTexture;
uniform sampler2D diffuseTexture;
uniform sampler2D specularTexture;

varying mat3 varyingTBNMatrix;

void main() {

    vec3 textureNormal = normalize((texture2D(normalTexture, varyingTexCoord).xyz * 2.0) -1.0);
    textureNormal = normalize(varyingTBNMatrix * textureNormal);

    gl_FragData[0] = texture2D(diffuseTexture, varyingTexCoord);
    gl_FragData[1] = texture2D(specularTexture, varyingTexCoord);
    gl_FragData[2] = vec4(((textureNormal.xyz + 1.0) * 0.5), 1.0);
}
```

Normals are unit vectors (going from -1.0 to 1.0 on xyz).

Need to convert to 0.0 - 1.0 for texture storage.

Final Shader

```
varying vec3 varyingNormal;
varying vec2 varyingTexCoord;

uniform sampler2D normalTexture;
uniform sampler2D diffuseTexture;
uniform sampler2D specularTexture;

varying mat3 varyingTBNMatrix;

void main() {

    vec3 textureNormal = normalize((texture2D(normalTexture, varyingTexCoord).xyz * 2.0) -1.0);
    textureNormal = normalize(varyingTBNMatrix * textureNormal);

    gl_FragData[0] = texture2D(diffuseTexture, varyingTexCoord);
    gl_FragData[1] = texture2D(specularTexture, varyingTexCoord);
    gl_FragData[2] = vec4(((textureNormal.xyz + 1.0) * 0.5), 1.0);
}
```

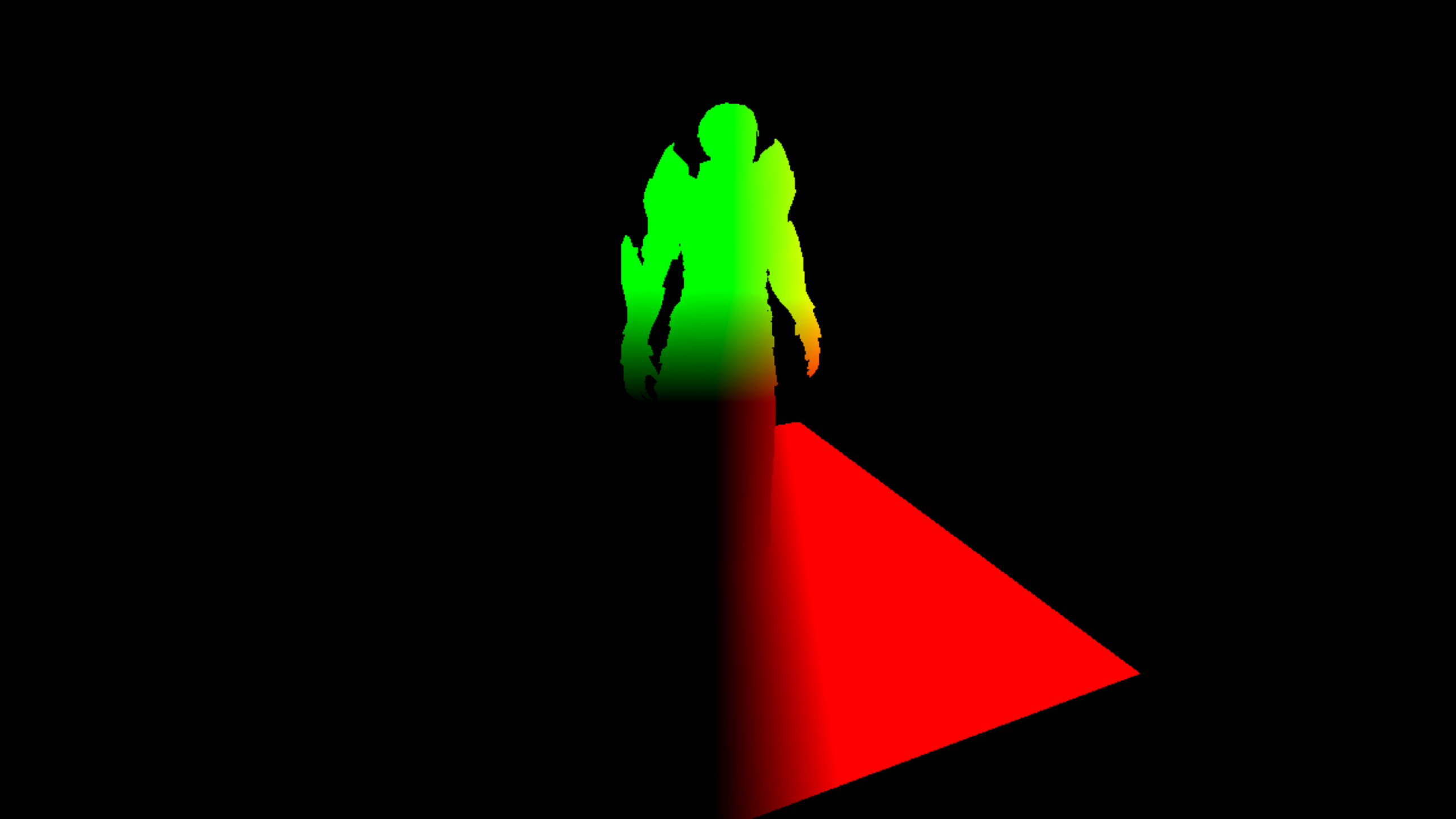
Normals are unit vectors (going from -1.0 to 1.0 on xyz).

Need to convert to 0.0 - 1.0 for texture storage.

Screen Shader

Position

Position G-Buffer?



Position from depth buffer.

$$\vec{W} = \begin{bmatrix} gl\_FragCoord.x \\ gl\_FragCoord.y \\ fromDepthTexture \end{bmatrix} \qquad \vec{N} = \begin{bmatrix} \frac{(2*W_x)-(2*V_x)}{V_w} - 1 \\ \frac{(2*W_y)-(2*V_y)}{V_h} - 1 \\ \frac{(2*W_z)-D_f-D_n}{D_f-D_n} - 1 \end{bmatrix}$$

```
[ xx   xx   xx   xx ]
[ xx   xx   xx   xx ]
[ 0    0    T1   T2 ]
[ 0    0    E1    0 ]
```

Original projection matrix

$$C_w = \frac{T2}{N_z - \frac{T1}{E1}}$$

$$\vec{C}_{xyz} = \vec{N} * C_w$$

Multiply the final clip coordinate by the inverse of the projection matrix.

```
// need to convert Z-depth from 0.0/1.0 texture storage to -1.0/1.0 NDC space
// we can use the screen texture coordinate for our device X and Y, but need to
vec3 ndcspace = vec3(texCoordVar * 2.0 - 1.0, (texture2D(depthFramebuffer, texCoordVar).x * 2.0) - 1.0);

float T2 = projectionMatrix[3][2];
float T1 = projectionMatrix[2][2];
float E1 = projectionMatrix[2][3];

float clipSpaceW = (T2 / (ndcspace.z - T1/E1));
vec4 clipSpace = vec4(ndcspace * clipSpaceW, clipSpaceW);

vec4 cameraSpace = inverseProjectionMatrix * clipSpace;
```

Final screen shader

```
uniform sampler2D normalFramebuffer;
uniform sampler2D diffuseFramebuffer;
uniform sampler2D specularFramebuffer;
uniform sampler2D depthFramebuffer;

uniform mat4 inverseProjectionMatrix;
uniform mat4 projectionMatrix;

varying vec2 texCoordVar;

struct Light {
    vec3 lightPosition;
    vec3 lightColor;
    vec3 specularLightColor;
};

uniform Light lights[2];

void main() {

  vec3 ndcspace = vec3(texCoordVar * 2.0 - 1.0, (texture2D(depthFramebuffer, texCoordVar).x * 2.0) - 1.0);

  float T2 = projectionMatrix[3][2];
  float T1 = projectionMatrix[2][2];
  float E1 = projectionMatrix[2][3];

  float clipSpaceW = (T2 / (ndcspace.z - T1/E1));
  vec4 clipSpace = vec4(ndcspace * clipSpaceW, clipSpaceW);

  vec4 cameraSpace = inverseProjectionMatrix * clipSpace;

  vec3 deferredPosition = cameraSpace.xyz;
  vec3 deferredNormal = normalize((  texture2D( normalFramebuffer, texCoordVar).xyz * 2.0) -1.0);
  vec3 deferredDiffuse = texture2D(diffuseFramebuffer, texCoordVar).xyz;
  vec3 deferredSpecular = texture2D(specularFramebuffer, texCoordVar).xyz;

  // calculate intensity as usual using above values for position, normal and diffuse and specular texture values
  gl_FragColor = vec4(intensity , 1.0);

}
```

Problems with deferred shading.

- Transparent/blended objects are difficult to render (can be drawn with an additional step).

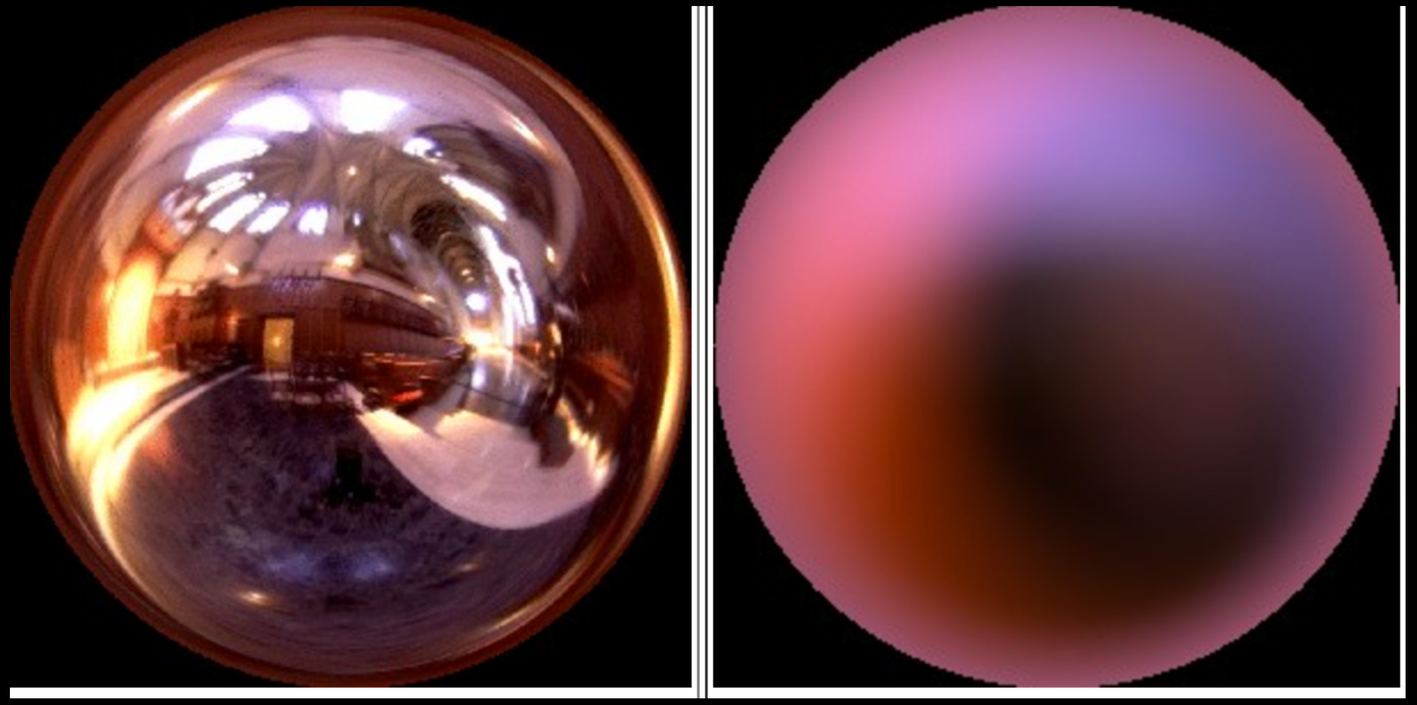
- Everything is rendered with the same shader material (can separate using a material ID g-buffer).

- Cannot use hardware anti-aliasing (Can substitute with shader-based anti-aliasing).

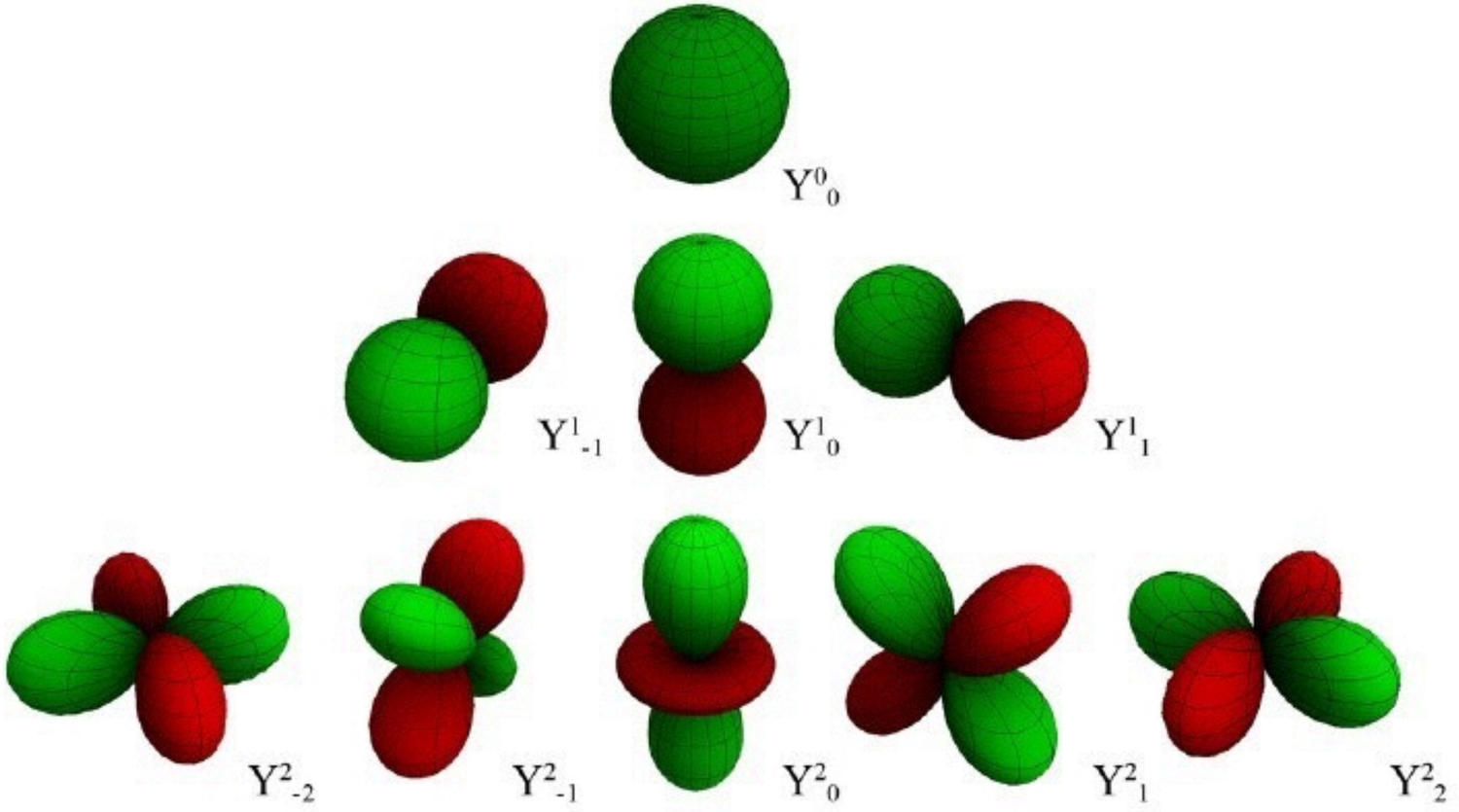
Optimizations: Combining g-buffers

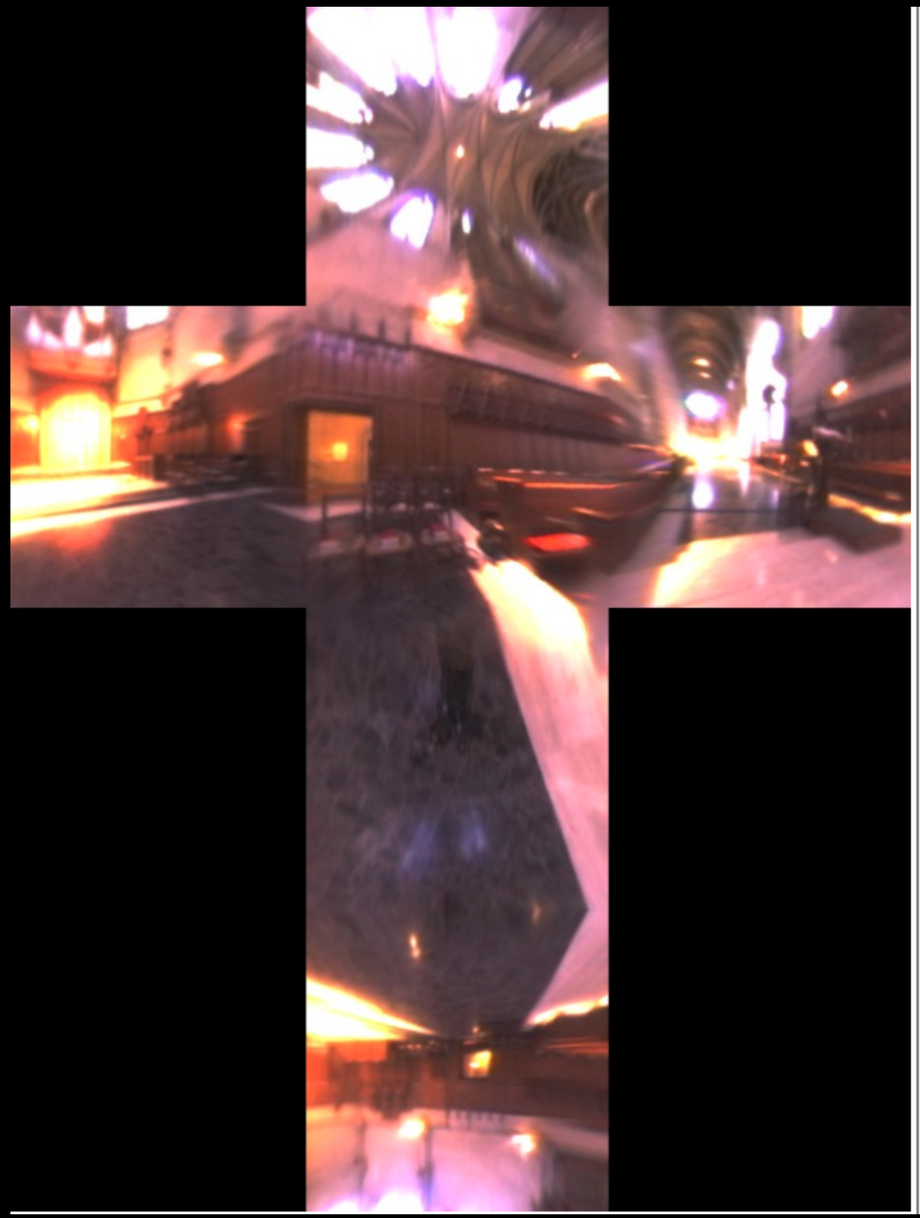
Image based lighting.

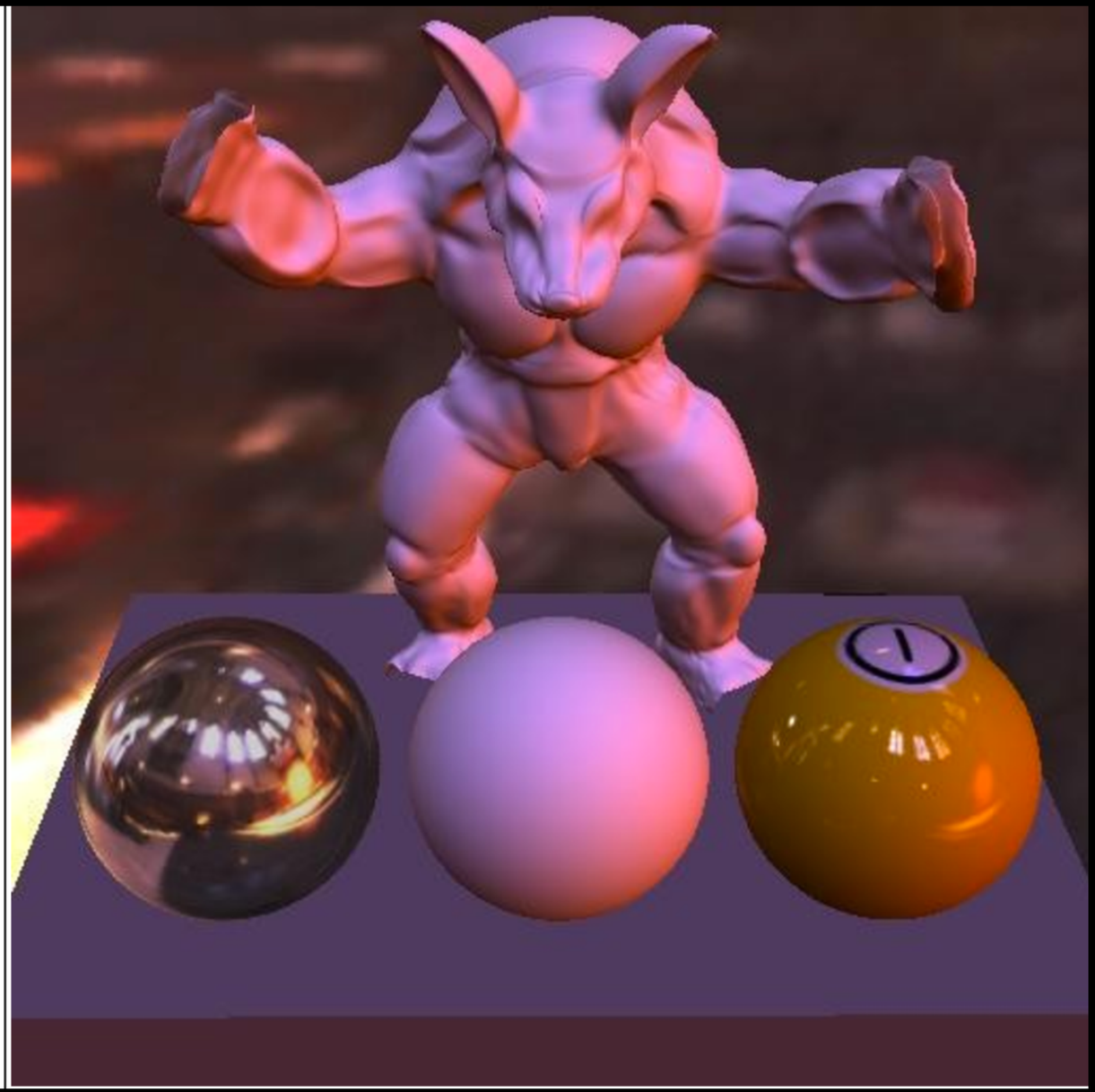
Radiance maps.



Spherical Harmonics


Y^0_0
Y^1_{-1}
Y^1_0
Y^1_1
Y^2_{-2}
Y^2_{-1}
Y^2_0
Y^2_1
Y^2_2





https://youtu.be/FQMbxzTUuSg?t=2257