COMP3320 Introduction to OpenGL

Alex Biddulph

The University of Newcastle, Australia

Based on the work provided at www.learnopengl.com

Semester 2, 2021

Blinn-Phong Lighting

- Specular lighting breaks down when the angle between the view and light direction vectors exceeds 90°
- Harsh cutoff appears when the angle exceeds 90°

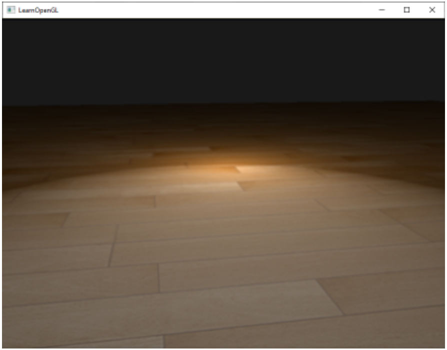

Figure: Image sourced from `learnopengl.com/Advanced-Lighting/Advanced-Lighting`

Blinn-Phong Lighting

- Rather than using the reflected vector use the halfway vector, the vector exactly halfway between the view and lighting directions

$$\vec{H} = \frac{\vec{L} + \vec{V}}{\|\vec{L} + \vec{V}\|}$$

- Specular calculation then reduces to

```
vec3  norm       = normalize(frag_normal);
vec3  light_dir  = normalize(frag_pos - light_pos);
vec3  view_dir   = normalize(view_pos - frag_pos);
vec3  halfway    = normalize(view_dir + light_dir);
float strength   = pow(max(dot(norm, halfway), 0.0f), 32.0f);
vec3  specular   = 0.5f * strength * light_colour;
vec3  result     = specular * object_colour;
```

Normal Mapping

- We are modelling "bumpy" surfaces with flat triangles
- Vertex normals are useful for creating some nice lighting effects, but are not fine-grained enough to show shadows from bumps in the objects surface
- Textures fail to show the more detailed bumps in the object though

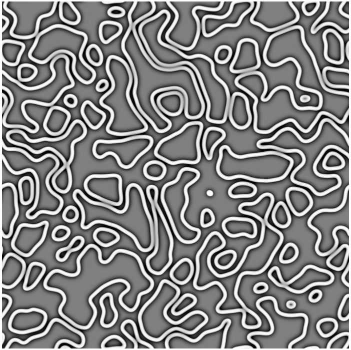

Figure: Image sourced from `https://3dtextures.me/2017/12/28/abstract-005/`

Normal Mapping

- Bake normal vectors into a texture, a "normal map"
- Sample normal map to get normal vector for the current fragment
- Use this normal for lighting equations

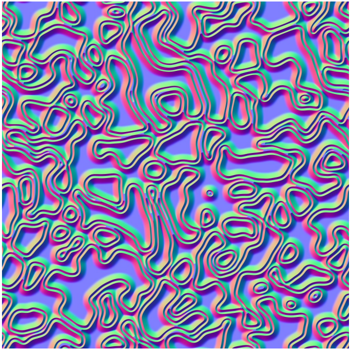

Figure: Image sourced from `https://3dtextures.me/2017/12/28/abstract-005/`

Normal Mapping

- Creating the texture maps the normals to the range $[0, 255]$
- Texture sampling maps the normals to the range $[0, 1]$
- Need to convert the samples back to the range $[-1, 1]$

```
vec3 normal = vec3(texture(normal_map, texCoords));
normal      = normalize(normal * 2.0 - 1.0);
```

Normal Mapping

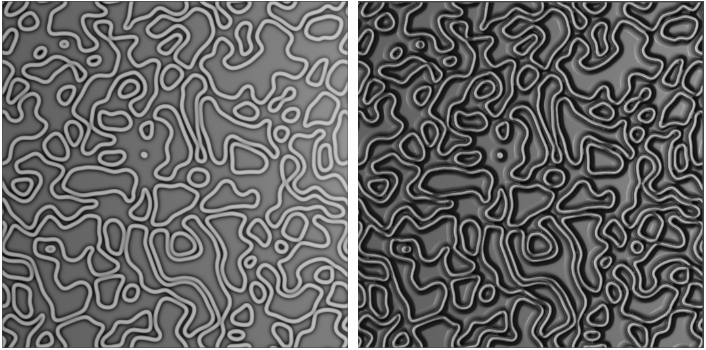

Figure: Left: Diffuse texture only. Right: Diffuse texture with normal map. Image sourced from `https://3dtextures.me/2017/12/28/abstract-005/`

Tangent Space

- Normal vectors in normal maps are expressed in tangent space
- Tangent space normals always point in the positive z direction (roughly)
- Need to account for this in lighting calculations
- Two options
  1. Map normal vectors from tangent space to world space
  2. Map light positions from world space to tangent space
- Option one needs to happen in the fragment shader (happens once per fragment)
- Option two can be done in the vertex shader (happens once per vertex)
- There are typically a lot less vertices than there are fragments

Tangent Space

- The *TBN* matrix will transform vectors from tangent space to world space
- The inverse of this matrix will transform world space vectors to tangent space
- $TBN = [\mathrm{Tangent}|\mathrm{Bitanget}|\mathrm{Normal}]$
- Tangent, bitanget, and normal vectors need to be calculated per-vertex
- Can calculate all of these manually, or you can ask assimp to do it for you

Examples

See ☞ learnopengl.com for a more detailed explanation of tangent space and the tangent and bitangent vectors

Tangent Space

- Provide tangent, bitangent, and normal vectors to your vertex shader as uniforms
- These vectors will be in model space, need to convert them to world space

```
vec3 T = normalize(vec3(Hwm * vec4(aTangent, 0.0)));
vec3 B = normalize(vec3(Hwm * vec4(aBitangent, 0.0)));
vec3 N = normalize(vec3(Hwm * vec4(aNormal, 0.0)));
mat3 TBN = mat3(T, B, N);
```

- Strictly speaking you don't need to provide the bitangent vector
- $\vec{T}$, $\vec{B}$, and $\vec{N}$ are all orthogonal to each other, so given $\vec{T}$ and $\vec{N}$ we can calculate $\vec{B}$ as

$$\vec{B} = cross\left(\vec{N}, \vec{T}\right)$$

Tangent Space

- With complex models with lots of vertices being shared between triangles it is possible to end up with $\vec{T}$, $\vec{B}$, and $\vec{N}$ not being mutually orthogonal
- If you think this is causing your normal mapping to be slightly off the Gram-Schmidt process can be used to re-orthogonalise the vectors

```
vec3 T = normalize(vec3(Hwm * vec4(aTangent, 0.0)));
vec3 N = normalize(vec3(Hwm * vec4(aNormal, 0.0)));

// Ensure T is orthogonal to N
// If T is already orthogonal to N then
// dot(T, N) = 0, so T = T
T = normalize(T - dot(T, N) * N);

// Now calculate B to ensure all
// three vectors are mutually orthogonal
vec3 B = cross(N, T);

mat3 TBN = mat3(T, B, N);
```

Displacement Mapping

- Like normal mapping, displacement mapping is used to significantly increase the detail in our rendered objects
- Sample a texture and use the sampled value to shift the texture coordinates used for any other texture sampling

```
float height = texture(dispMap, texCoords);
vec2 p       = viewDir.xy / viewDir.z * (height * dispScale);
vec2 coords  = texCoords - p;

vec3 diffuse = texture(diffuseMap, coords);
vec3 normal  = texture(normalMap, coords);
normal       = normalize(normal * 2.0 - 1.0);
```

- *dispScale* is an extra scaling factor to attenuate the effect of the displacement map
- Best to use a displacement map in conjuction with a normal map for best results

Displacement Mapping

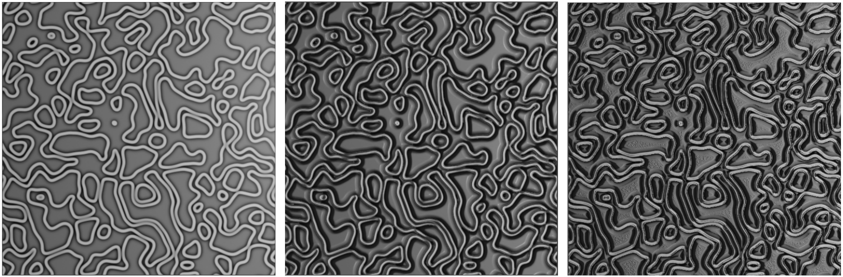

Figure: Left: Diffuse texture only. Middle: Diffuse texture with normal map. Right: Diffuse texture with normal map and displacement map. Image sourced from `https://3dtextures.me/2017/12/28/abstract-005/`