159.709 Computer Graphics

Environment Mapping & Reflections

Daniel Playne
d.p.playne@massey.ac.nz

Environment Maps

Environment Mapping is a technique in computer graphics for approximating reflections on a surface.

It is an image-based lighting technique as the environment surrounding the objects is represented by an image.

Rather that trying to calculate the reflection all of the objects in a scene (a rather complex process), surfaces can instead just reflect the environment map which just requires a simple texture lookup.

Environment Maps

Environment maps provide a simple and efficient way of approximating reflections in a scene.

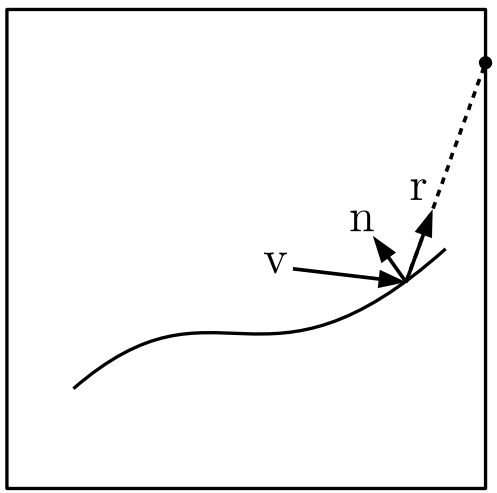
The correctness of this method relies on two major assumptions (which are rarely true).

- All reflected objects are very far away and there is no discernible parallax effect.
- The object has no self-reflection.

Environment Map - Reflections

When the viewer looks at a reflective surface, the view vector $\mathbf{v}$ can be reflected on the surface (according to the normal $\mathbf{n}$).

The reflected vector $\mathbf{r}$ is then used to access the environment map.



Environment Map - Reflections

To reflect other objects in the scene would require each object to have its own environment map.

A simple approximation is to only reflect objects outside the scene (the skybox).

This is a common method of implementing reflections due to the relatively low cost and reasonably convincing effect.

Environment Map - Reflections

To implement environment map reflections we can render our objects as usual but also pass in a cubemap texture for the skybox.

The shader program will calculate the reflected view vector and use this to access the skybox texture to find the reflected light.

To do this it must convert the reflected vector back into ***world space*** to access the correct texture coordinates.

Reflections - Vertex Shader

```
1  // OpenGL 4.0
2  #version 400
3
4  // Input to Vertex Shader
5  in vec4 vert_Position;
6  in vec4 vert_Normal;
7
8  // Transform Matrices
9  uniform mat4 u_Model;
10 uniform mat4 u_View;
11 uniform mat4 u_Projection;
12 uniform mat4 u_IView;
13
14 // Output to Fragment Shader
15 out vec4 frag_Position;
16 out vec4 frag_Normal;
17
18 void main() {
19    // -------------------- Output to Fragment Shader --------------------
20     // Frag Position
21    frag_Position = u_View * u_Model * vert_Position;
22
23    // Frag Normal
24    frag_Normal = u_View * u_Model * vert_Normal;
25
26    // -------------------- Vertex Position --------------------
27    gl_Position = u_Projection * u_View * u_Model * vert_Position;
28 }
```

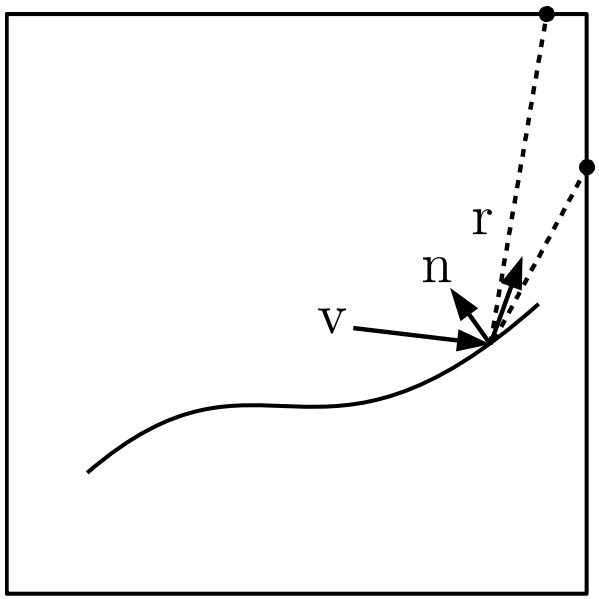
Reflections - Fragment Shader

```
// OpenGL 4.0
#version 400

// Input from Vertex Shader
in vec4 frag_Position;
in vec4 frag_Normal;

// Transform Matrices
uniform mat4 u_IView;

// Texture
uniform samplerCube u_texture_Map;

// Output from Fragment Shader
out vec4 pixel_Colour;

void main () {
   // -------------------- Calculate Vectors --------------------
   // View, Normal, Reflected (eye space)
   vec3 v = normalize(frag_Position.xyz);
   vec3 n = normalize(frag_Normal.xyz);
   vec3 r = reflect(v, n);

   // STR Vector (world space)
   vec3 str = (u_IView * vec4(r, 0.0f)).xyz * vec3(-1.0, -1.0, 1.0);

   // -------------------- Fragment Colour --------------------
   pixel_Colour = texture(u_texture_Map, str);
}
```

Environment Map - Reflections

These shaders are sufficient for rendering perfectly reflective surfaces (such as a mirror or chrome finish).

However, most reflective surfaces are not quite so perfect. These surface reflect light from a range of angles.



Environment Map - Reflections

One method would be to sample many different texture coordinates in the appropriate range and combine them together according to a function describing the surface.

Taking many texture sample can be an expensive process so another commonly-used method is to pre-blur the environment map and use this instead.

Environment Map - Reflections

This would require a different environment map for each *level* of roughness of reflective surfaces in the scene. It is also not so straightforward to blur the textures without introducing artefacts at the corners.

A simple solution is to instead access the cubemap at different mip-map levels. These mipmap levels are bi-linearly interpolated from the original full-resolution image and provide a good approximation of a more rough surface.

Environment Map - Reflections

To use this method of reflection, mipmaps must first be enabled and generated for the cubemap:

```
// Set storage - log_2(image size)
glTexStorage2D(GL_TEXTURE_CUBE_MAP, max_levels, GL_RGBA8, width, height);

// Load six faces
for(int i = 0; i < 6; i++) {
   // Load image from file
   unsigned char *image = loadImage(filename[i], width, height, n, true);
   // Copy image data into texture
   glTexSubImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, 0, 0, width, height,
                   GL_RGBA, GL_UNSIGNED_BYTE, image);
   // Delete image data
   delete[] image;
}
// Configure Texture
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_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);

// Generate Mipmaps
glGenerateMipmap(GL_TEXTURE_CUBE_MAP);
```

Environment Map - Reflections

In standard filtering for cubemaps, texels are not filtered across different faces of the cube which can result in a *seam* in the reflection running across the edges of the environment map.

Modern hardware is able to perform filtering across different faces by enabling the following flag:

```
1 glEnable(GL_TEXTURE_CUBE_MAP_SEAMLESS);
```

Environment Map - Reflections

With these settings enabled we are able to access the environment map at a specific level-of-detail to produce a *blurred* reflection and approximate a rough surface.

```
1 // -------------------- Fragment Colour ----------
2 pixel_Colour = textureLod(u_texture_Map, str, 7);
```

The function `textureLod` works the same as `texture` except that it also provides a specified mipmap level to access.

Environment Map - Reflections

Reflections using level-of-detail $0, 3, 6, 9$.

Environment Map - Reflections

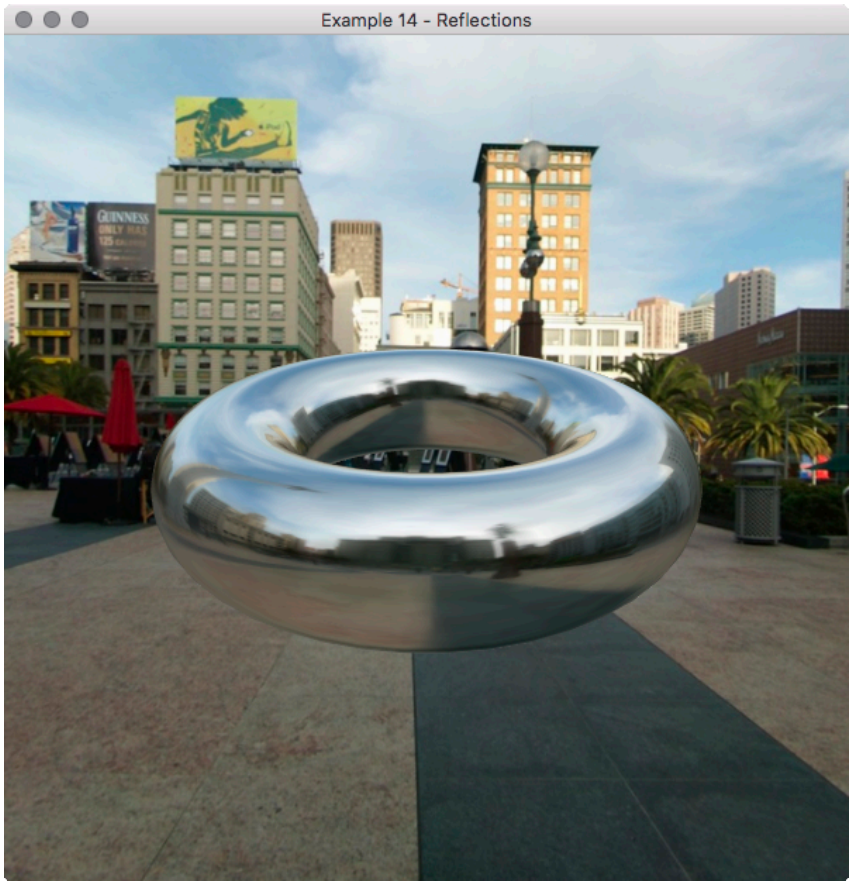
Another approach that can used to tune the appearance of a reflective surface is to combine reflection values from two (or more) mipmap levels from the environment map.

One simple way to do this is to interpolate between two different levels using a *roughness* value.

```
1 // -------------------- Fragment Colour ----------
2 pixel_Colour = (texture   (u_texture_Map, str)    * (1.0f - roughness) +
3                 textureLod(u_texture_Map, str, 7) * (       roughness));
```

Environment Map - Reflections

This approach allows more range in the reflectance functions approximated.



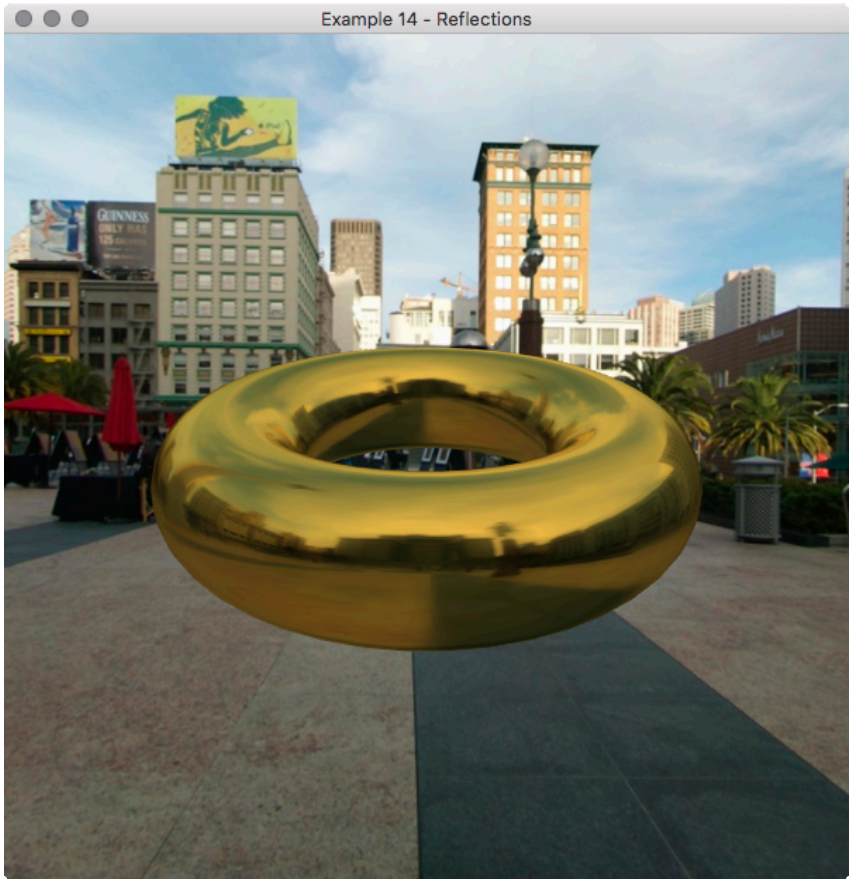
Environment Map - Reflections

Finally we could also introduce a material coefficient to control the colours that the material absorbs the colours that are reflected.

```
1 // -------------------- Fragment Colour ----------
2 pixel_Colour = Kr * (texture   (u_texture_Map, str)    * (1.0f - roughness) +
3                      textureLod(u_texture_Map, str, 7) * (       roughness));
```

Environment Map - Reflections

This allows different materials (e.g. different metals) to be rendered:



Environment Map - Example

Example 12 - Environment Map