Environment Mapping

- Environmental mapping is a way to create the appearance of highly **reflective** and **refractive** surfaces without ray tracing.
Types of Environment Maps

- Assumes environment infinitely far away
- Options: Store “object’s environment as

  a) Sphere around object (sphere map)

  b) Cube around object (cube map)

- OpenGL supports **cube maps** and **sphere maps**
Cube mapping

- Need to compute reflection vector, \( \mathbf{r} \)
- Use \( \mathbf{r} \) by for environment map lookup
Cube Map: How to Store

- Stores “environment” around objects as 6 sides of a cube (1 texture)
- Load 6 textures separately into 1 OpenGL cubemap
Cube Maps

- Loaded cube map texture can be accessed in GLSL through `cubemap sampler`
- Compute reflection vector $\mathbf{R} = 2(\mathbf{N} \cdot \mathbf{V})\mathbf{N} - \mathbf{V}$
- Perform cubemap lookup using $\mathbf{R}$ vector (texcoord)

```cpp
textColor = textureCube(mycube, R);
```

- $\mathbf{R}$ is 3D vector, so texture coordinates must be 3D $(x, y, z)$
- OpenGL figures out which face of cube $\mathbf{R}$ hits, to look up
- More details on lookup later
Creating Cube Map

- Use 6 cameras directions from scene center
  - each with a 90 degree angle of view
Cube Map Layout

Make 1 cubemap texture object from 6 images
Declaring Cube Maps in OpenGL

- Declare each of 6 sides of cube map separately.

  - E.g. to declare +X image

    ```
glTextureMap2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, level, rows, columns, border, GL_RGBA, GL_UNSIGNED_BYTE, image1)
    ```

- Repeat similar for other 5 images (sides)

- Parameters apply to all six images. E.g

  ```
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAP_WRAP_S, GL_REPEAT)
  ```
Cube Map Example (init)

// colors for sides of cube
GLubyte red[3] = {255, 0, 0};
GLubyte green[3] = {0, 255, 0};
GLubyte blue[3] = {0, 0, 255};
GLubyte cyan[3] = {0, 255, 255};
GLubyte magenta[3] = {255, 0, 255};
GLubyte yellow[3] = {255, 255, 0};

glEnable(GL_TEXTURE_CUBE_MAP);

// Create texture object
glGenTextures(1, tex);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_CUBE_MAP, tex[0]);
Cube Map (init II)

Load 6 different pictures into 1 cube map of environment

```c
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X ,
            0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, red);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X ,
            0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, green);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y ,
            0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, blue);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y ,
            0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, cyan);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z ,
            0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, magenta);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z ,
            0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, yellow);
glTexParameteri(GL_TEXTURE_CUBE_MAP,
            GL_TEXTURE_MAG_FILTER,GL_NEAREST);
```
Cube Map (init III)

```c
GLuint texMapLocation;
GLuint tex[1];

texMapLocation = glGetUniformLocation(program, "texMap");
glUniform1i(texMapLocation, tex[0]);
```

Connect texture map (tex[0]) to variable texMap in fragment shader (texture mapping done in frag shader)
void quad(int a, int b, int c, int d)
{
    static int i = 0;

    normal = normalize(cross(vertices[b] - vertices[a], vertices[c] - vertices[b]));

    normals[i] = normal;
    points[i] = vertices[a];
    i++;

    // rest of data
Vertex Shader

out vec3 R;
in vec4 vPosition;
in vec4 Normal;
uniform mat4 ModelView;
uniform mat4 Projection;

void main() {
    gl_Position = Projection*ModelView*vPosition;
    vec4 eyePos  = vPosition; // calculate view vector V
    vec4 NN = ModelView*Normal; // transform normal
    vec3 N = normalize(NN.xyz); // normalize normal
    R = reflect(eyePos.xyz, N); // calculate reflection vector R
}
in vec3 R;
uniform samplerCube texMap;

void main()
{
    vec4 texColor = textureCube(texMap, R); // look up texture map using R

    gl_FragColor = texColor;
}
Refraction using Cube Map

- Can also use cube map for refraction (transparent)
Reflection and Refraction

- At each vertex

\[ I = I_{\text{amb}} + I_{\text{diff}} + I_{\text{spec}} + I_{\text{refl}} + I_{\text{tran}} \]

- Refracted component \( I_T \) is along transmitted direction \( \mathbf{t} \)
Finding Transmitted (Refracted) Direction

- Transmitted direction obeys **Snell’s law**
- Snell’s law: relationship holds in diagram below

\[
\frac{\sin(\theta_2)}{c_2} = \frac{\sin(\theta_1)}{c_1}
\]

\(c_1, c_2\) are speeds of light in medium 1 and 2
Finding Transmitted Direction

- If ray goes from faster to slower medium (e.g. air to glass), ray is bent towards normal
- If ray goes from slower to faster medium (e.g. glass to air), ray is bent away from normal
- $c_1/c_2$ is important. Usually measured for medium-to-vacuum. E.g water to vacuum
- Some measured relative $c_1/c_2$ are:
  - Air: 99.97%
  - Glass: 52.2% to 59%
  - Water: 75.19%
  - Sapphire: 56.50%
  - Diamond: 41.33%
Transmission Angle

- Vector for transmission angle can be found as

\[ t = \frac{c_2}{c_1} \text{dir} + \left( \frac{c_2}{c_1} (m \cdot \text{dir}) - \cos(\theta_2) \right) m \]

where

\[ \cos(\theta_2) = \sqrt{1 - \left( \frac{c_2}{c_1} \right)^2 \left( 1 - (m \cdot \text{dir})^2 \right)} \]

Or just use GLSL built-in function `refract` to get \( T \)
Refraction Vertex Shader

```cpp
out vec3 T;
in vec4 vPosition;
in vec4 Normal;
uniform mat4 ModelView;
uniform mat4 Projection;

void main() {
    gl_Position = Projection*ModelView*vPosition;
    vec4 eyePos = vPosition; // calculate view vector V
    vec4 NN = ModelView*Normal; // transform normal
    vec3 N = normalize(NN.xyz); // normalize normal
    T = refract(eyePos.xyz, N, iorefr); // calculate refracted vector T
}
```

Was previously

```cpp
R = reflect(eyePos.xyz, N);
```
Refraction Fragment Shader

in vec3 T;
uniform samplerCube RefMap;

void main()
{
    vec4 refractColor = textureCube(RefMap, T);  // look up texture map using T
    refractcolor = mix(refractColor, WHITE, 0.3);  // mix pure color with 0.3 white

    gl_FragColor = refractcolor;
}
Caustics occur when light is focussed on diffuse surface.
Cube Map Layout
Indexing into Cube Map: How?

- Compute $\mathbf{R} = 2(\mathbf{N} \cdot \mathbf{V})\mathbf{N} - \mathbf{V}$
- Object at origin
- Use largest magnitude component of $\mathbf{R}$ to determine face of cube
- Other 2 components give texture coordinates
Cube Map Texture Lookup:
Given an \((s,t,p)\) direction vector, what \((r,g,b)\) does that correspond to?

- Let \(L\) be the texture coordinate of \((s, t, \text{ and } p)\) with the largest magnitude
- \(L\) determines which of the 6 2D texture “walls” is being hit by the vector \((-X\) in this case)
- The texture coordinates in that texture are the remaining two texture coordinates divided by \(L\):
  \((a/L, b/L)\)

Built-in GLSL functions

```glsl
vec3 ReflectVector = reflect( vec3 eyeDir, vec3 normal );
vec3 RefractVector = refract( vec3 eyeDir, vec3 normal, float Eta );
```
Example

- $\mathbf{R} = (-4, 3, -1)$
- Same as $\mathbf{R} = (-1, 0.75, -0.25)$
- Look up face $x = -1$ and $[y = 0.75, z = -0.25]$ as tex coords
- Not quite right since cube defined by $x, y, z = \pm 1$ rather than $[0, 1]$ range needed for texture coordinates
- Remap by from $[-1,1]$ to $[0,1]$ range
  - $s = \frac{1}{2} + \frac{1}{2} y$, $t = \frac{1}{2} + \frac{1}{2} z$
- Hence, $s = 0.875$, $t = 0.375$
References

- Real Time Rendering by Akenine-Moller, Haines and Hoffman