Recall: 6 Main Steps to Apply Texture

1. Create texture object
2. Specify the texture
   - Read or generate image
   - assign to texture (hardware) unit
   - enable texturing (turn on)
3. Assign texture (corners) to Object corners
4. Specify texture parameters
   - wrapping, filtering
5. Pass textures to shaders
6. Apply textures in shaders

still haven’t talked about setting texture parameters
Recall: Step 4: Specify Texture Parameters

- Texture parameters control how texture is applied
  - **Wrapping parameters** used if \( s,t \) outside (0,1) range
    - **Clamping:** if \( s,t > 1 \) use 1, if \( s,t < 0 \) use 0
    - **Wrapping:** use \( s,t \) modulo 1

```
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP )
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT )
```
Magnification and Minification

**Magnification:** Stretch small texture to fill many pixels
**Minification:** Shrink large texture to fit few pixels
Step 4: Specify Texture Parameters
Texture Value Lookup

How about coordinates that are not exactly at the intersection (pixel) positions?

A) Nearest neighbor
B) Linear Interpolation
C) Other filters
Example: Texture Magnification

- 48 x 48 image projected (stretched) onto 320 x 320 pixels

- Nearest neighbor filter
- Bilinear filter (avg 4 nearest texels)
- Cubic filter (weighted avg. 5 nearest texels)
Texture mapping parameters

1) Nearest Neighbor (lower image quality)

2) Linear interpolate the neighbors (better quality, slower)

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

Or
GL_TEXTURE_MAX_FILTER
```

```c
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

Dealing with Aliasing

- Point sampling of texture can lead to aliasing errors.
Area Averaging

Better but slower option is area averaging
Other Stuff

- Wrapping texture onto curved surfaces. E.g. cylinder, can, etc
  \[ s = \frac{\theta - \theta_a}{\theta_b - \theta_a} \quad t = \frac{z - z_a}{z_b - z_a} \]

- Wrapping texture onto sphere
  \[ s = \frac{\theta - \theta_a}{\theta_b - \theta_a} \quad t = \frac{\phi - \phi_a}{\phi_b - \phi_a} \]

- Bump mapping: perturb surface normal by a quantity proportional to texture
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, r
- Use 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

  ```glsl
  vec4 texColor = textureCube(mycube, texcoord);
  ```

- Texture coordinates must be 3D (x, y, z)
Creating Cube Map

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

• Compute \( \mathbf{R} = 2(\mathbf{N} \cdot \mathbf{V})\mathbf{N} - \mathbf{V} \)

• Object at origin

• Perform lookup:

\[
\text{vec4 texColor} = \text{textureCube(mycube, } \mathbf{R} \text{);}
\]

• **Largest magnitude component** of \( \mathbf{R} \) \((x, y, z)\) used to determine face of cube

• Other 2 components give texture coordinates

More on this later....
Declaring Cube Maps in OpenGL

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

- Repeat similar for other 5 images (sides)
- Make **1 cubemap texture object from 6 images**
- Parameters apply to all six images. E.g

```c
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAP_WRAP_S, GL_REPEAT)
```

- **Note:** texture coordinates are in 3D space (s, t, r)
Cube Map Example (init)

```c
// 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]);
```

This example generates simple Colors as a texture

You can also just load 6 pictures of environment
Cube Map (init II)

Load 6 different pictures into 1 cube map of environment

```c
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X,
             0, 1, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, red);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
             0, 1, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, green);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y,
             0, 1, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, blue);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,
             0, 1, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, cyan);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z,
             0, 1, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, magenta);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z,
             0, 1, 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)
Adding Normals

```c
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
}
```

Calculate and set quad normals
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
}
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

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

- 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:

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

where

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

Or just use GLSL built-in function \text{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
```
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;
}
References

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