Cube Mapping
What is Cube Mapping?

Cube Mapping is the process of creating a representation of an object’s surrounding environment as a collection of 6 images, grouped together as a single “cube map texture”.

Think of it as a folding box. (BTW, I have this box on a 2-sided PowerPoint slide if you want to print and cutout your own.)

Note: as the scene observer, you are inside the box.
BTW, if you would like your very own Nvidia Lobby 3D Cube, print this page and the next page on a single piece of paper.

In the **Printer Properties** be sure that the **Print on both sides** setting says:
**Yes, flip over**
or
**Flip pages on Short Edge**
Using Cube Mapping to Model a 3D Environment

Take 6 photos in all directions. Warning! It is tricky to do this and get the seams to match correctly.

Go here: https://www.humus.name/index.php?page=Textures&start=0 to find lots of cool cube map textures.

You can find lots more cube map textures just by Googling: cube map textures
Cube Map of the Kelley Engineering Center Atrium
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 \(L\):
  \(s = a/L, t = b/L\).

\[
\text{reflect( ) and refract( ) are built-in GLSL functions}
\]

\[
\text{vec3 ReflectVector} = \text{reflect( vec3 eyeDir, vec3 normal );}
\]
\[
\text{vec3 RefractVector} = \text{refract( vec3 eyeDir, vec3 normal, float Eta );}
\]
Remember Angle-of-Reflection-Equals-Angle-of-Incidence from Lighting?

That's what the built-in reflect() function does.
Using the Cube Map for Reflection
Using the Cube Map for Reflection

Vertex shader

```glsl
out vec3 vNormal;
out vec3 vEyeDir;
out vec3 vMC;

void main( )
{
    vec4 newVertex = gl_Vertex;
    // could possibly apply displacements to newVertex here
    vMC = newVertex.xyz;
    vec3 ECposition = ( gl_ModelViewMatrix * newVertex ).xyz;
    vEyeDir = ECposition - vec3(0.,0.,0.); // vector from eye to pt
    vNormal = normalize( gl_NormalMatrix * gl_Normal );
    // or newNormal if you have displaced vertices
    gl_Position = gl_ModelViewProjectionMatrix * newVertex;
}
```
Using the Cube Map for Reflection

Fragment shader

```glsl
in vec3 vNormal;
in vec3 vEyeDir;
in vec3 vMC;
uniform samplerCube uReflectUnit;

void main( )
{
    vec3 normal = vNormal;
    // if you are bump-mapping, apply noise to normal here using vMC
    vec3 reflectVector = reflect( vEyeDir, normal );
    vec4 reflectColor = texture( uReflectUnit, reflectVector ); // on Macs, use textureCube( )
    gl_FragColor = vec4( reflectColor.rgb, 1. )
}
```

Oregon State University
Computer Graphics
The Index of Refraction, η (eta)

The Index of Refraction (IOR) is a measure of how much light slows down as it passes through a particular material. The larger the IOR, the slower the speed of light in that material.

Snell’s Law of Refraction says that:

\[ \frac{\sin \theta_2}{\sin \theta_1} = \frac{\eta_1}{\eta_2} \]

Or:

\[ \sin \theta_2 = \sin \theta_1 \frac{\eta_1}{\eta_2} \]

That's what the built-in refract( ) function does.

Notice that there are certain combinations of the η’s that require \( \sin \Theta_2 \) to be outside the range \(-1. \rightarrow +1\)., which is not possible. This indicates that the refraction has actually become a **Total Internal Reflection**.

https://en.wikipedia.org/wiki/Snell’s_law
## Common Indices of Refraction

<table>
<thead>
<tr>
<th>Material</th>
<th>$\eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air</td>
<td>1.000237</td>
</tr>
<tr>
<td>Ice</td>
<td>1.31</td>
</tr>
<tr>
<td>Water</td>
<td>1.33</td>
</tr>
<tr>
<td>Pyrex</td>
<td>1.47</td>
</tr>
<tr>
<td>Window Glass</td>
<td>1.52</td>
</tr>
<tr>
<td>Quartz</td>
<td>1.54</td>
</tr>
<tr>
<td>Cubic Zirconia</td>
<td>2.16</td>
</tr>
<tr>
<td>Diamond</td>
<td>2.42</td>
</tr>
<tr>
<td>Moissanite</td>
<td>2.69</td>
</tr>
</tbody>
</table>


Using the Cube Map for Refraction
Using the Cube Map for Refraction

Vertex shader

```glsl
out vec3 vNormal;
out vec3 vEyeDir;
out vec3 vMC;

void main( )
{
    vec4 newVertex = gl_Vertex;
    // could possibly apply displacements to newVertex here
    vMC = newVertex.xyz;
    vec3 ECposition = ( gl_ModelViewMatrix * newVertex).xyz;
    vEyeDir = ECposition – vec3(0.,0.,0.); // vector from eye to pt
    vNormal = normalize( gl_NormalMatrix * gl_Normal );
    // or newNormal if you have displaced vertices
    gl_Position = gl_ModelViewProjectionMatrix * newVertex;
}
```

Same as for reflection...
Using the Cube Map for Refraction

Fragment shader

```glsl
in vec3 vNormal;
in vec3 vEyeDir;
in vec3 vMC;

uniform float uEta;
uniform samplerCube uReflectUnit;
uniform samplerCube uRefractUnit;
uniform float uMix, uWhiteMix;

const vec3 WHITE = vec3(1.,1.,1.);

void main()
{
    vec3 normal = vNormal;  // if you are bump-mapping, apply noise to normal here using vMC
    vec3 reflectVector = reflect(vEyeDir, normal);
    vec3 reflectColor = texture(uReflectUnit, reflectVector).rgb;  // on Macs, use textureCube()

    vec3 refractVector = refract(vEyeDir, normal, uEta);
    vec3 refractColor;
    if( all( equal( refractVector, vec3(0.,0.,0.)) ) )  // like saying “if all elements of the refractVector are == 0.0 . . .”
    {
        refractColor = reflectColor;
        // . . . then treat this as a total internal reflection
    }
    else
    {
        refractColor = texture(uRefractUnit, refractVector).rgb;  // on Macs, use textureCube()
        refractColor = mix( refractColor, WHITE, uWhiteMix );
    }

    vec3 color = mix( refractColor, reflectColor, uMix );
    color = mix( color, WHITE, uWhiteMix );
gl_FragColor = vec4(color, 1.);
}
```
## Cube Mapping in glman

These must be listed in the order: 
+X, -X, +Y, -Y, +Z, -Z

These have nothing to do with the cube mapping. They are here to create the six walls, without which the cube mapping looks ridiculous.
Cube Mapping in a C/C++ Program

GLSLProgram Pattern;
GLuint CubeName;
char * FaceFiles[6]
{
    "kec.posx.bmp",
    "kec.negx.bmp",
    "kec.posy.bmp",
    "kec.negy.bmp",
    "kec.posz.bmp",
    "kec.negz.bmp"
};
void
InitGraphics( )
{
    // open the window . . .
    // setup the callbacks . . .
    // initialize glew . . .
    // create and compile the shader . . .

    glGenTextures( 1, &CubeName );
    glBindTexture( GL_TEXTURE_CUBE_MAP, CubeName );
    glTexParameteri( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_REPEAT );
    glTexParameteri( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_REPEAT );
    glTexParameteri( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_REPEAT );
    glTexParameteri( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
    glTexParameteri( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR );

    for( int file = 0; file < 6; file++ )
    {
        int nums, numt;
        unsigned char * texture2d = BmpToTexture( FaceFiles[file], &nums, &numt );
        if( texture2d == NULL )
            fprintf( stderr, "Could not open BMP 2D texture '%s', FaceFiles[file] \n", FaceFiles[file] );
        else
            fprintf( stderr, "BMP 2D texture '%s' read -- nums = %d, numt = %d\n", FaceFiles[file], nums, numt );

        glTexImage2D( GL_TEXTURE_CUBE_MAP_POSITIVE_X + file, 0, 3, nums, numt, 0,
                     GL_RGB, GL_UNSIGNED_BYTE, texture2d );

        delete [ ] texture2d;
    }
}
void Display( )
{
    
    int uReflectUnit = 5;
    int uRefractUnit = 6;
    float uAd = 0.1f;
    float uBd = 0.1f;
    float uEta = 1.4f;
    float uTol = 0.f;
    float uMix = 0.4f;

    Pattern.Use( );
    glBindTexture(GL_TEXTURE_CUBE_MAP, CubeName);
    glBindTexture(GL_TEXTURE_CUBE_MAP, CubeName);
    Pattern.SetUniformVariable( "uReflectUnit", uReflectUnit );
    Pattern.SetUniformVariable( "uRefractUnit", uRefractUnit );
    Pattern.SetUniformVariable( "uMix", uMix );
    Pattern.SetUniformVariable( "uEta", uEta )
    glEndList( SphereList );
    Pattern.UnUse;
}
Sidebar: You Can Also Use Cube Mapping to "Surround" an Object with a Texture: A Cube Map of the World
Sidebar: You Can Also Use Cube Mapping to "Surround" an Object with a Texture: A Cube Map of the World

Use the normal \((n_x, n_y, n_z)\) as the \((s, t, p)\) for the 3D lookup

(Some shapes map better than others…)
Sidebar: You Can Also Use Cube Mapping to "Surround" an Object with a Texture: A Cube Map of the World

Vertex shader

```glsl
out vec3 vNormal;

void main() {
    vNormal = normalize( gl_Normal );
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Fragment shader

```glsl
uniform samplerCube uTexUnit;
in vec3 vNormal;

void main() {
    vec4 newcolor = texture( uTexUnit, vNormal );
    gl_FragColor = vec4( newcolor.rgb, 1. );
}
```

Use the normal \((n_x,n_y,n_z)\) as the \((s,t,p)\) for the 3D lookup.