# Bump Maps

The technique of **bump mapping** varies the apparent shape of the surface by perturbing the normal vectors as the surface is rendered; the colors that are generated by shading then show a variation in the surface properties.

We could perturb the normals in many ways; the following procedure for parametric surfaces is an efficient one. Let $\mathbf{p}(u, v)$ be a point on a parametric surface. The unit normal at that point is given by the cross product of the partial derivative vectors:

$$\mathbf{n} = \frac{\mathbf{p}_u \times \mathbf{p}_v}{|\mathbf{p}_u \times \mathbf{p}_v|}$$

where

$$\mathbf{p}_u = \begin{bmatrix} \frac{\partial x}{\partial u} \\ \frac{\partial y}{\partial u} \\ \frac{\partial z}{\partial u} \end{bmatrix} \qquad \mathbf{p}_v = \begin{bmatrix} \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial v} \end{bmatrix}$$

Suppose we display the surface in the normal direction by a given function called the **bump**

**function**, $d(u,v)$, which is assumed known and small $(|d(u,v)| \ll 1)$:

$$\mathbf{p}' = \mathbf{p} + d(u,v)\mathbf{n}.$$

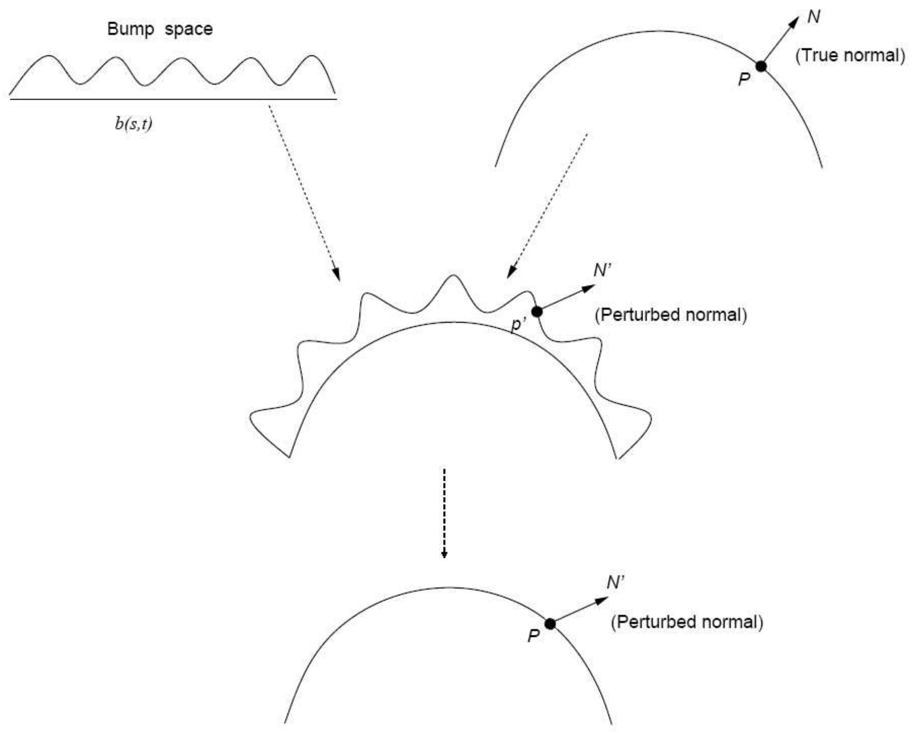
We would prefer not to display the surface; we just want to make it look as though we have displaced it. We can achieve the desired look by altering the normal $\mathbf{n}$, instead of $\mathbf{p}$, and using the perturbed normal in our shading calculations.

The normal at the perturbed point $\mathbf{p}'$ is given by the cross product

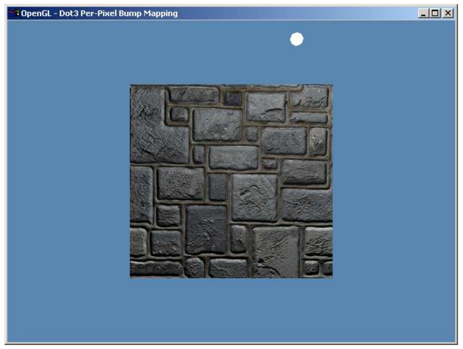
$$\mathbf{n}' = \mathbf{p}'_u \times \mathbf{p}'_v.$$

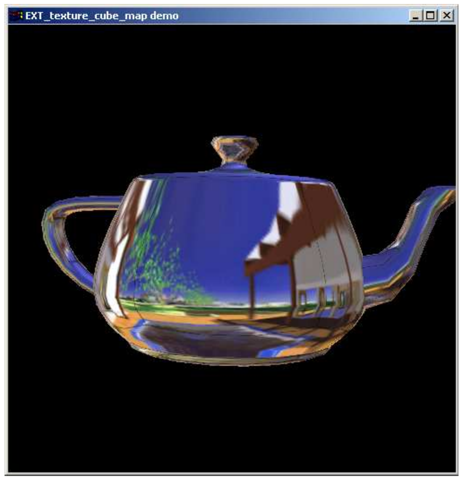
We can compute the two partial derivatives by differentiating the equation for $\mathbf{p}'$, obtaining

$$\mathbf{p}'_u = \mathbf{p}_u + d_u\mathbf{n} + d(u,v)\mathbf{n}_u,$$
$$\mathbf{p}'_v = \mathbf{p}_v + d_v\mathbf{n} + d(u,v)\mathbf{n}_v.$$


Bump space
b(s,t)
N
(True normal)
P
N'
(Perturbed normal)
p'
N'
(Perturbed normal)
P

## Bump and Environment Mapping in OpenGL





## OpenGL Examples for Bump Mapping

**Texture Mapping Set Up Stage 1** Use `GL_DOT3_RGB_EXT` to find the dot-product of (N.L), where N is stored in the normal map, and L is passed in as the `PRIMARY_COLOR` using the standard glColor3f call.

```
// Texture Map Enabling and Normal Map Binding
glEnable(GL_TEXTURE_2D);
glActiveTexture(GL_TEXTURE0);
glBindTexture (GL_TEXTURE_2D, NormalMapTextureID);
// Perform a Dot3 operation
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE_EXT);
glTexEnvf(GL_TEXTURE_ENV, GL_COMBINE_RGB_EXT, GL_DOT3_RGB_EXT);
// between the N (of N.L) which is stored in a normal map texture
glTexEnvf(GL_TEXTURE_ENV, GL_SOURCE0_RGB_EXT, GL_TEXTURE);
glTexEnvf(GL_TEXTURE_ENV, GL_OPERAND0_RGB_EXT, GL_SRC_COLOR);
// with the L (of N.L) which is stored in the vertex's diffuse color
glTexEnvf(GL_TEXTURE_ENV, GL_SOURCE1_RGB_EXT, GL_PRIMARY_COLOR_EXT);
glTexEnvf(GL_TEXTURE_ENV, GL_OPERAND1_RGB_EXT, GL_SRC_COLOR);
```

## Texture Mapping Set Up Stage 2

Modulate the base texture by N.L calculated in Stage 1.

```
glActiveTexture(GL_TEXTURE1);
glBindTexture (GL_TEXTURE_2D, ColorTextureID);
glEnable(GL_TEXTURE_2D);
// Modulate...
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE_EXT);
glTexEnvf(GL_TEXTURE_ENV, GL_COMBINE_RGB_EXT, GL_MODULATE);
// the color argument passed down from the previous stage (Stage 1) with...
glTexEnvf(GL_TEXTURE_ENV, GL_SOURCE0_RGB_EXT, GL_PREVIOUS_EXT);
glTexEnvf(GL_TEXTURE_ENV, GL_OPERAND0_RGB_EXT, GL_SRC_COLOR);
// the texture for this stage with.
glTexEnvf(GL_TEXTURE_ENV, GL_SOURCE1_RGB_EXT, GL_TEXTURE);
glTexEnvf(GL_TEXTURE_ENV, GL_OPERAND1_RGB_EXT, GL_SRC_COLOR);
```

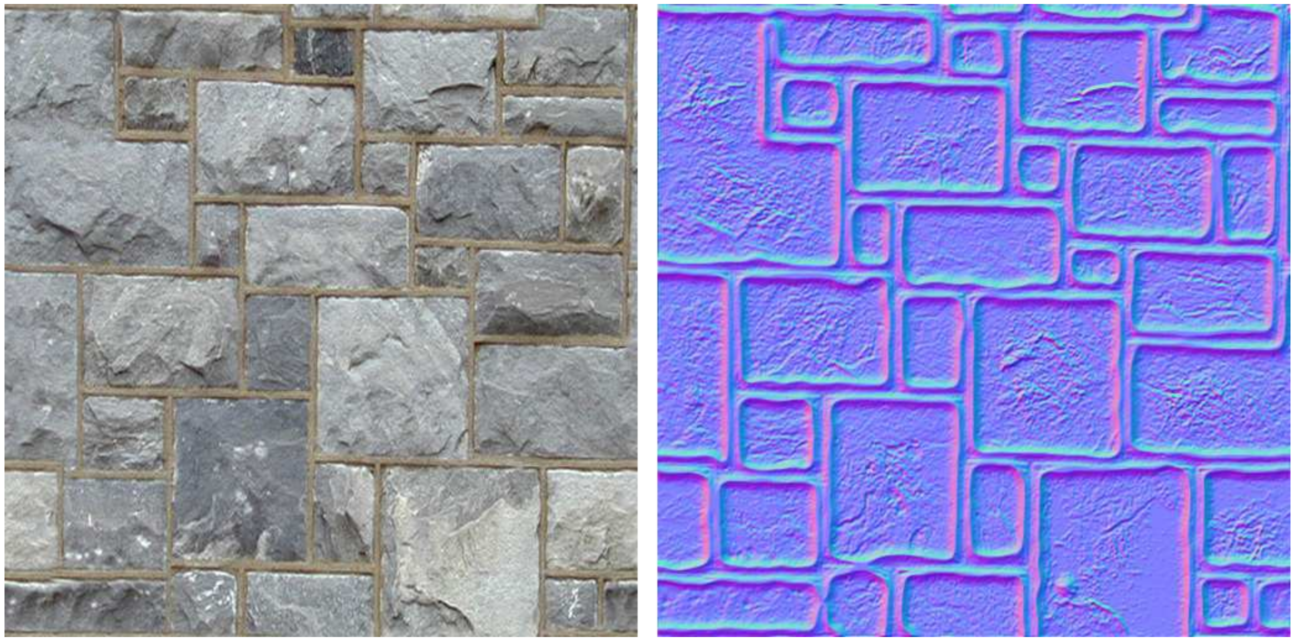
## Drawing Triangles with Textures

```
glBegin(GL_TRIANGLES);
glMultiTexCoord2f( GL_TEXTURE0, TexCoord[0].u, TexCoord[0].v );
glMultiTexCoord2f( GL_TEXTURE1, TexCoord[0].u, TexCoord[0].v );
glColor3f( Light.x, Light.y, Light.z );
glNormal3f( Normal[0].nx, Normal[0].ny, Normal[0].nz );
glVertex3f( Vertex[0].x, Vertex[0].y, Vertex[0].z );

glMultiTexCoord2f( GL_TEXTURE0, TexCoord[1].u, TexCoord[1].v );
glMultiTexCoord2f( GL_TEXTURE1, TexCoord[1].u, TexCoord[1].v );
glNormal3f( Normal[1].nx, Normal[1].ny, Normal[1].nz );
glVertex3f( Vertex[1].x, Vertex[1].y, Vertex[1].z );

glMultiTexCoord2f( GL_TEXTURE0, TexCoord[2].u, TexCoord[2].v );
glMultiTexCoord2f( GL_TEXTURE1, TexCoord[2].u, TexCoord[2].v );
glNormal3f( Normal[2].nx, Normal[2].ny, Normal[2].nz );
glVertex3f( Vertex[2].x, Vertex[2].y, Vertex[2].z );
glEnd();
```

## Equivalent Cg Fragment Program

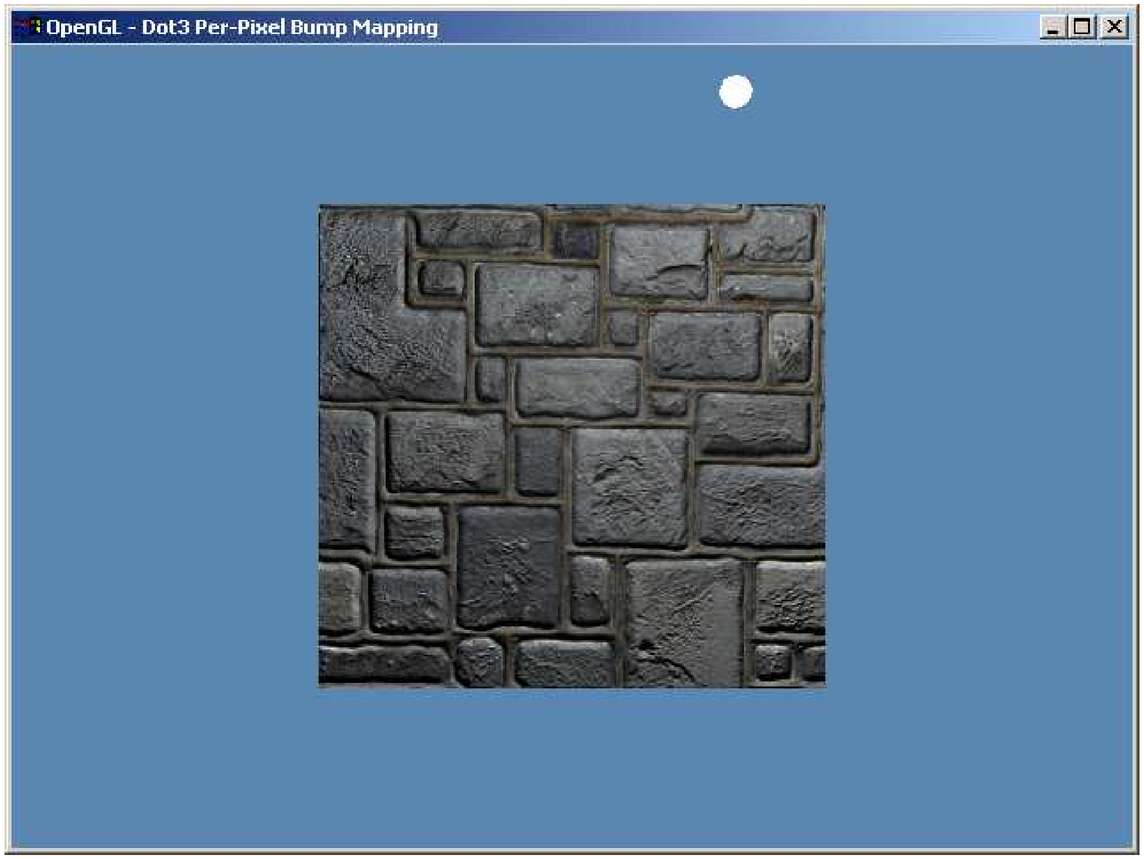
```
float4 main(float2 TexCoords : TEXCOORD0,
            float3 LightVector : COLOR0,
            uniform float3 AmbientColor,
            uniform sampler2D NormalMapTexture : TEXUNIT0,
            uniform sampler2D ColorTexture : TEXUNIT1): COLOR
{
    float3 Color = tex2D(ColorTexture, TexCoords).rgb;

    // Uncompress vectors ([0, 1] -> [-1, 1])
    float3 Light = 2.0 * (LightVector.rgb - 0.5);
    float3 BumpNormal = 2.0 * (tex2D(NormalMapTexture, TexCoords).rgb - 0.5);

    // Compute diffuse factor (N.L)
    float Diffuse = dot(BumpNormal, Light);

    return float4(Diffuse * Color + AmbientColor, 1.0);
}
```

## Bump Mapping Images



Color Texture and Normal Map

## Rendered Image



## Interaction Between Texture and Shading

For RGB colors, there are two options. The texture can modulate the shade that we would have assigned without texture mapping by multiplying the color components of the texture by the color components from the shader. Modulation is the default mode; it can be set by

```
glTexEnv(GL_TEX_ENV, GL_TEX_ENV_MODE, GL_MODULATE);
```

If we replace `GL_MODULATE` by `GL_DECAL`, the color of the texture determines the color of the object completely — a technique called **decaling**.

A growing technique for modulating the shade of textures (e.g. shadow on textured objects) is to use **multi-texture** units available on most current graphics cards.

Environment mapping is a form of texture mapping where the texture is derived from the environment. Once we obtain the required texture—either by scanning an image or through projecting a scene—OpenGL can automatically generate the tangent coordinates for a spherical mapping.

## OpenGL Examples for Environment Mapping

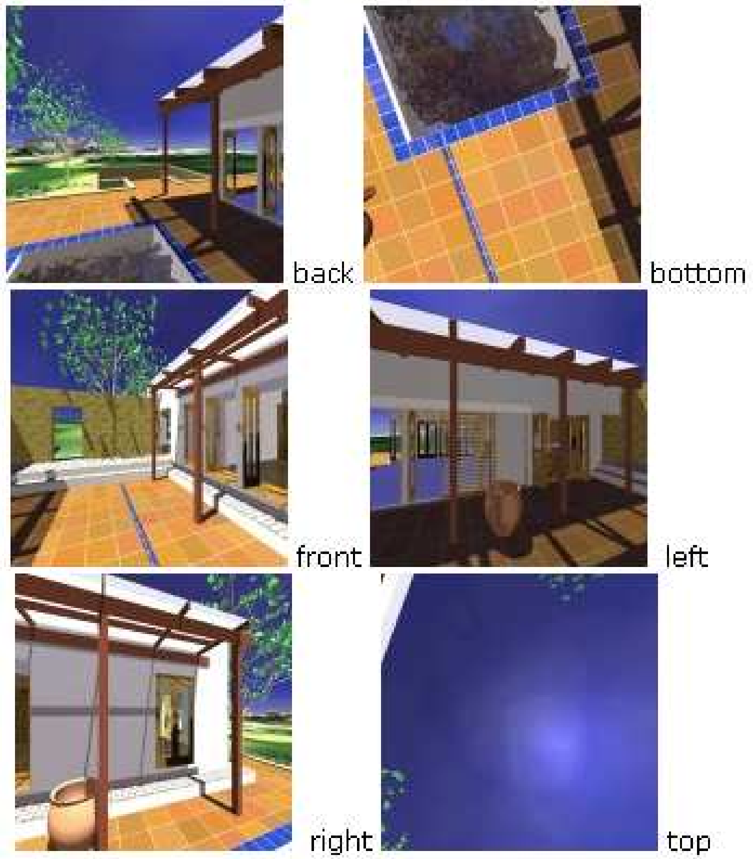
**Cube map texture set up**

```
static GLenum FaceTarget[6] = {
  GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT,  GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT,
  GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT,  GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT,
  GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT,  GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT
};

// Loading the cube map textures
// image[i] = left, right, top, bottom, back, and front images
for (i=0; i<6; i++) {
    glTexImage2D(FaceTarget[i], 0, GL_RGB, width[i], height[i], 0,
                 GL_RGB, GL_UNSIGNED_BYTE, image[i]);
}
glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_WRAP_T, GL_CLAMP);

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);

glEnable(GL_TEXTURE_CUBE_MAP_EXT);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
```

## Drawing Triangles with Textures

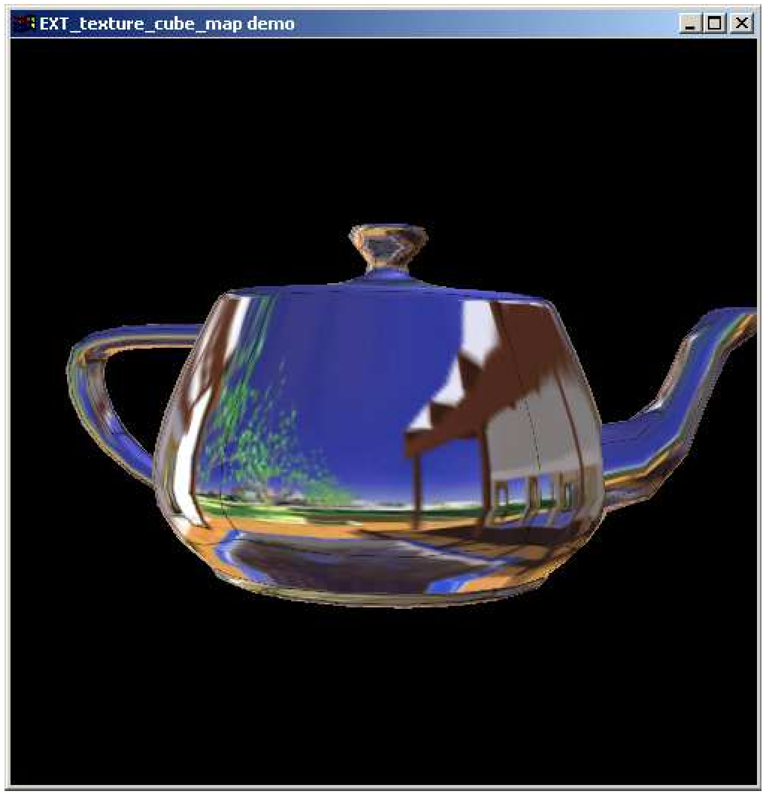
```
glBegin(GL_TRIANGLES);
glMultiTexCoord2f( GL_TEXTURE0, TexCoord[0].u, TexCoord[0].v );
glNormal3f( Normal[0].nx, Normal[0].ny, Normal[0].nz );
glVertex3f( Vertex[0].x, Vertex[0].y, Vertex[0].z );

glMultiTexCoord2f( GL_TEXTURE0, TexCoord[1].u, TexCoord[1].v );
glNormal3f( Normal[1].nx, Normal[1].ny, Normal[1].nz );
glVertex3f( Vertex[1].x, Vertex[1].y, Vertex[1].z );

glMultiTexCoord2f( GL_TEXTURE0, TexCoord[2].u, TexCoord[2].v );
glNormal3f( Normal[2].nx, Normal[2].ny, Normal[2].nz );
glVertex3f( Vertex[2].x, Vertex[2].y, Vertex[2].z );
glEnd();
```

## Cube Map Texture Images

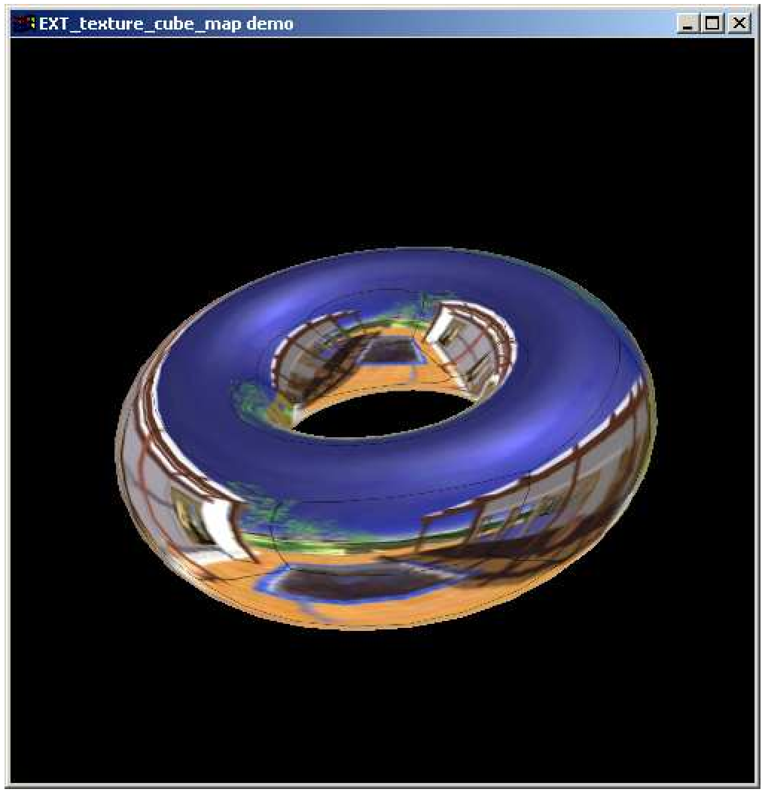


## Rendered Images

### Teapot



## Torus



## Reading Assignment and News

Please review the appropriate sections related to this lecture in chapter 7, and associated exercises, of the recommended text.

(Recommended Text: Interactive Computer Graphics, by Edward Angel, Dave Shreiner, 6th edition, Addison-Wesley)

Please track Blackboard for the most recent Announcements and Project postings related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics2012/cs354/)