CS559: Computer Graphics

Lecture 27: Texture Mapping

Li Zhang

Spring 2008

Many slides from Ravi Ramamoorthi, Columbia Univ, Greg Humphreys, UVA and Rosalee Wolfe, DePaul tutorial teaching texture mapping visually, Jingyi Yu, U Kentucky.

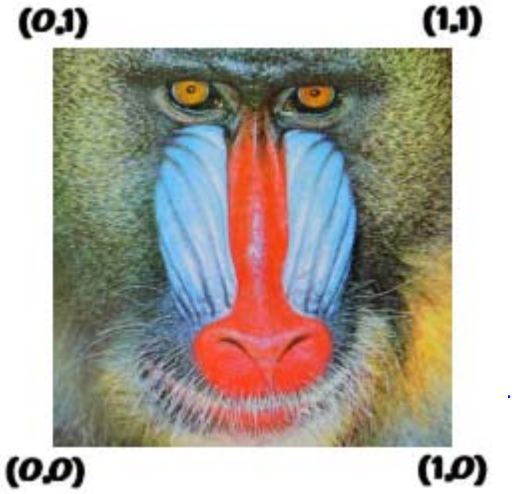
Today

- Continue on Texture mapping

- Reading
  - Redbook: Ch 9
  - (highly recommended) Moller and Haines: *Real-Time Rendering, 3e*, Ch 6
    - Linux: /p/course/cs559-lizhang/public/readings/6_texture.pdf
    - Windows: P:\course\cs559-lizhang\public\readings\6_texture.pdf
  - (optional) Shirley: Ch 11.4 – 11.8

Simple OpenGL Example

- Specify a texture coordinate at each vertex *(s, t)*
- Canonical coordinates where *s* and *t* are between 0 and 1




```
public void Draw() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslated(centerx, centery, depth);
    glMultMatrixf(Rotation);
        :
    // Draw Front of the Cube
    glEnable(GL_TEXTURE_2D);
    glBegin(GL_QUADS);
        glTexCoord2d(0, 1);
        glVertex3d( 1.0, 1.0, 1.0);
        glTexCoord2d(1, 1);
        glVertex3d(-1.0, 1.0, 1.0);
        glTexCoord2d(1, 0);
        glVertex3d(-1.0,-1.0, 1.0);
        glTexCoord2d(0, 0);
        glVertex3d( 1.0,-1.0, 1.0);
    glEnd();
    glDisable(GL_TEXTURE_2D);
        :
    glFlush();
```

glTexCoord works like glColor

Slides from Jingyi Yu

Initializing Texture Mapping

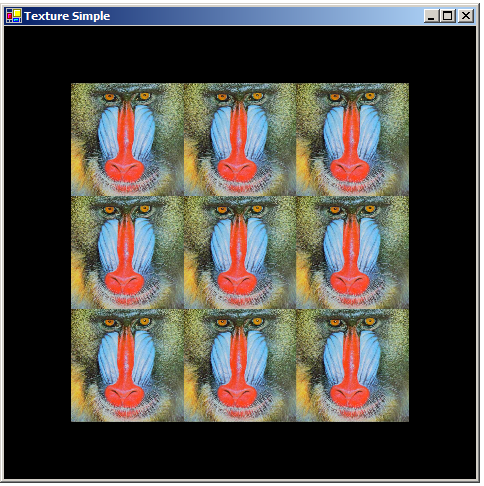
```
static GLubyte image[64][64][4];
static GLuint  texname;

void init(void) {
        glClearColor (0.0, 0.0, 0.0, 0.0);
        glShadeModel(GL_FLAT);
        glEnable(GL_DEPTH_TEST);
        //load in or generate image;
        …
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

        glGenTextures(1, &texName);
        glBindTexture(GL_TEXTURE_2D, texName);

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, checkImageWidth, checkImageHeight, 0,
                GL_RGBA, GL_UNSIGNED_BYTE, image);
}
```

Level index in the Pyramid

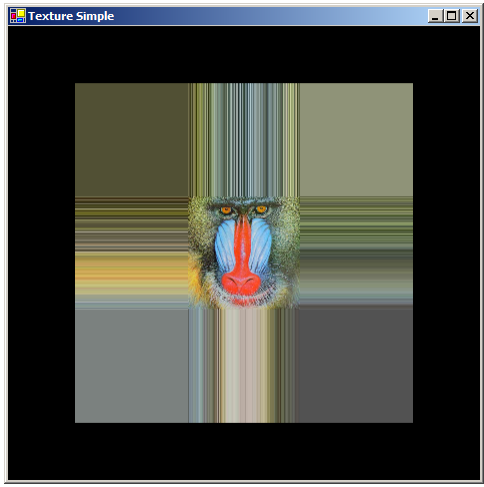
OpenGL Texture Peculiarities

- The width and height of Textures in OpenGL must be powers of 2
- The parameter space of each dimension of a texture ranges from [0,1) regardless of the texture's actual size.
- The behavior of texture indices outside of the range [0,1) is determined by the texture wrap options.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```



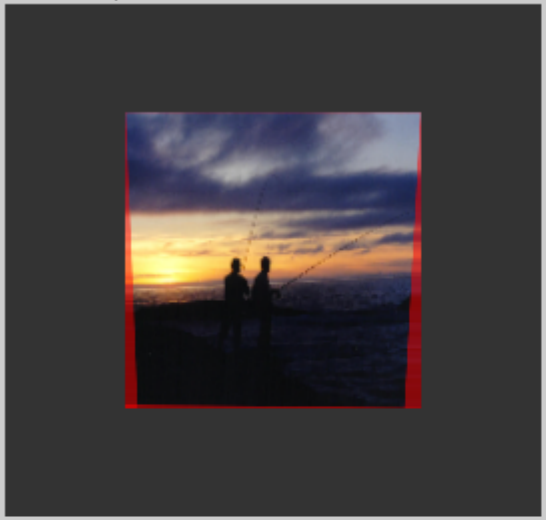

```
// Draw Front of the Cube
glEnable(GL_TEXTURE_2D);
glBegin(GL_QUADS);
glTexCoord2d(-1, 2);  glVertex3d( 1.0, 1.0, 1.0);
glTexCoord2d(2, 2);  glVertex3d(-1.0, 1.0, 1.0);
glTexCoord2d(2, -1);  glVertex3d(-1.0,-1.0, 1.0);
glTexCoord2d(-1, -1);  glVertex3d( 1.0,-1.0, 1.0);
glEnd();
glDisable(GL_TEXTURE_2D);
```



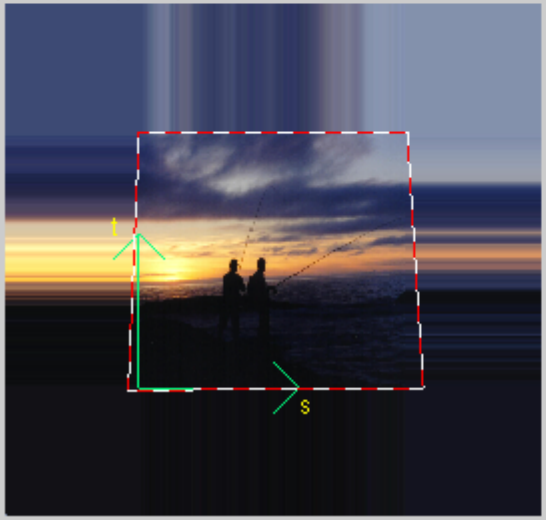

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

Texture

Screen-space view



Texture-space view



Command manipulation window

```
GLfloat border_color[ ] = {  1.00 ,  0.00 ,  0.00 ,  1.00 };
GLfloat env_color[ ] = {     0.00 ,  1.00 ,  0.00 ,  1.00 };

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, env_color);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,  GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,  GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,  GL_CLAMP);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,  GL_REPLACE);

glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);
glColor4f(  0.60  , 0.60  , 0.60  , 1.00  );
glBegin(GL_POLYGON);
glTexCoord2f( -0.0 , -0.0 ); glVertex3f( -1.0 , -1.0 , 0.0  );
glTexCoord2f( 1.1  , 0.0  ); glVertex3f( 1.0  , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 1.0  ); glVertex3f( 1.0  , 1.0  , 0.0  );
glTexCoord2f( 0.0  , 1.0  ); glVertex3f( -1.0 , 1.0  , 0.0  );
glEnd();
```

Click on the arguments and move the mouse to modify values.

http://www.xmission.com/~nate/tutors.html

Texture Function

glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE)

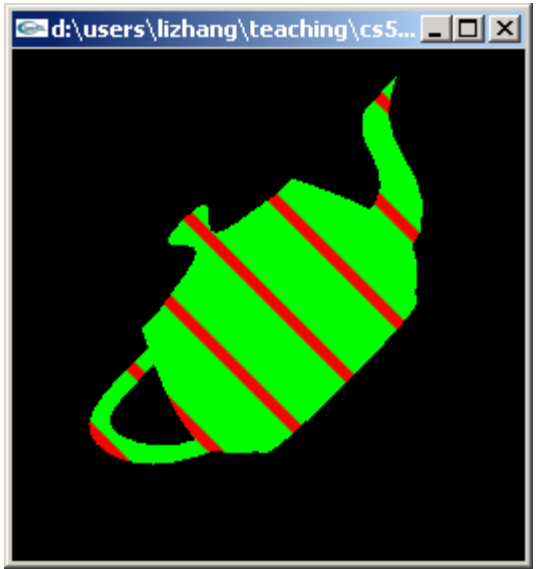
C = Ct, A = Af

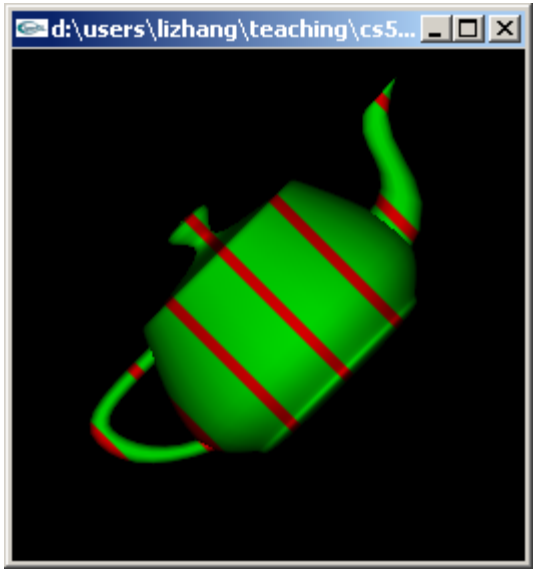
t: texture f:fragment

glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)

C = Cf*Ct, A = Af

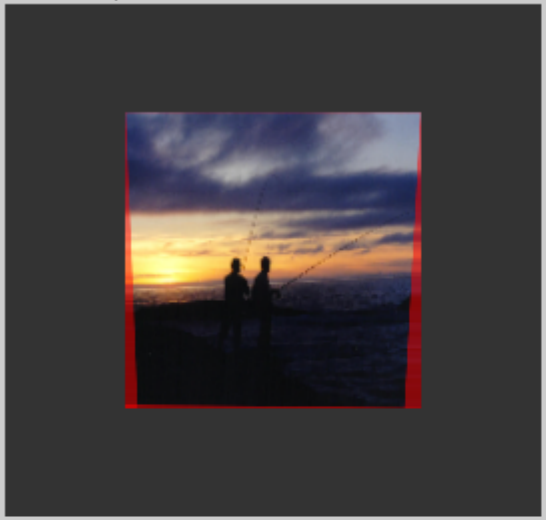
t: texture f:fragment



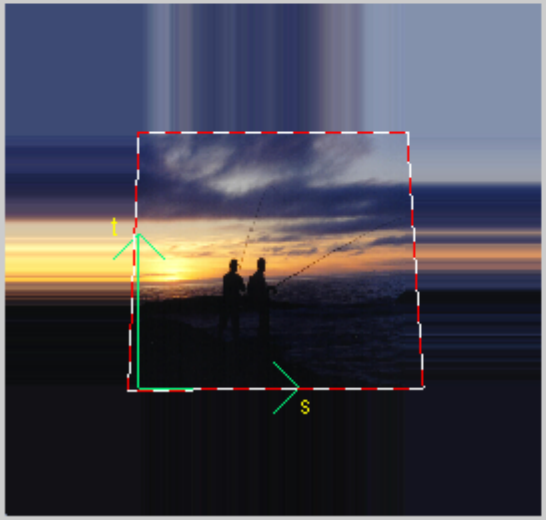


Texture

Screen-space view



Texture-space view



Command manipulation window

```
GLfloat border_color[ ] = {  1.00 ,  0.00 ,  0.00 ,  1.00 };
GLfloat env_color[ ] = {     0.00 ,  1.00 ,  0.00 ,  1.00 };

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, env_color);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,  GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,  GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,  GL_CLAMP);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,  GL_REPLACE);

glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);
glColor4f(  0.60  , 0.60  , 0.60  , 1.00  );
glBegin(GL_POLYGON);
glTexCoord2f( -0.0 , -0.0 ); glVertex3f( -1.0 , -1.0 , 0.0  );
glTexCoord2f( 1.1  , 0.0  ); glVertex3f( 1.0  , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 1.0  ); glVertex3f( 1.0  , 1.0  , 0.0  );
glTexCoord2f( 0.0  , 1.0  ); glVertex3f( -1.0 , 1.0  , 0.0  );
glEnd();
```

Click on the arguments and move the mouse to modify values.

http://www.xmission.com/~nate/tutors.html

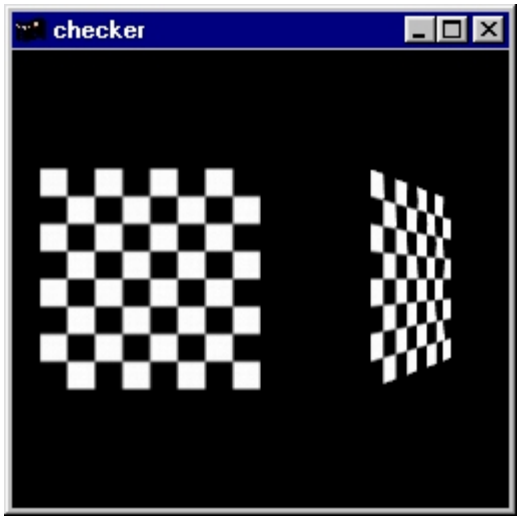
More texture function options

Base Internal Format	Replace Texture Function	Modulate Texture Function
GL_ALPHA	C = Cf, A = At	C = Cf, A = AfAt
GL_LUMINANCE	C = Lt, A = Af	C = CfLt, A = Af
GL_LUMINANCE_ALPHA	C = Lt, A = At	C = CfLt, A = AfAt
GL_INTENSITY	C = It, A = It	C = CfIt, A = AfIt
GL_RGB	C = Ct, A = Af	C = CfCt, A = Af
GL_RGBA	C = Ct, A = At	C = CfCt, A = AfAt

Table 9-3 : Decal and Blend Texture Function

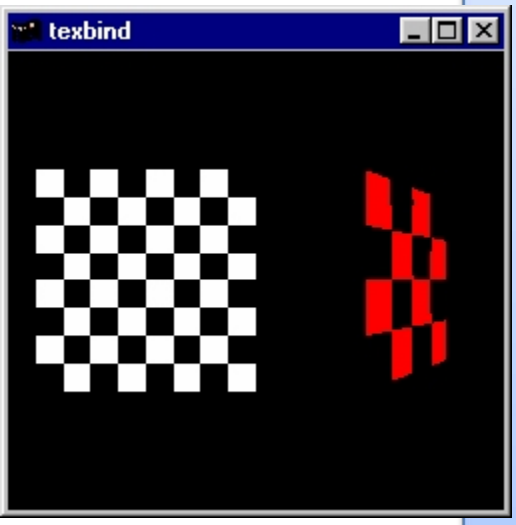
Base Internal Format	Decal Texture Function	Blend Texture Function
GL_ALPHA	undefined	C = Cf, A = AfAt
GL_LUMINANCE	undefined	C = Cf(1-Lt) + CcLt, A = Af
GL_LUMINANCE_ALPHA	undefined	C = Cf(1-Lt) + CcLt, A = AfAt
GL_INTENSITY	undefined	C = Cf(1-It) + CcIt, A = Af(1-It) + AcIt,
GL_RGB	C = Ct, A = Af	C = Cf(1-Ct) + CcCt, A = Af
GL_RGBA	C = Cf(1-At) + CtAt, A = Af	C = Cf(1-Ct) + CcCt, A = AfAt

Using one texture for different objects




```
static GLubyte image[64][64][4];
static GLuint  texname;

void display(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texName);

        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
        glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
        glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);

        glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
        glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
        glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
        glEnd();
        glFlush();

        glDisable(GL_TEXTURE_2D);
}
```

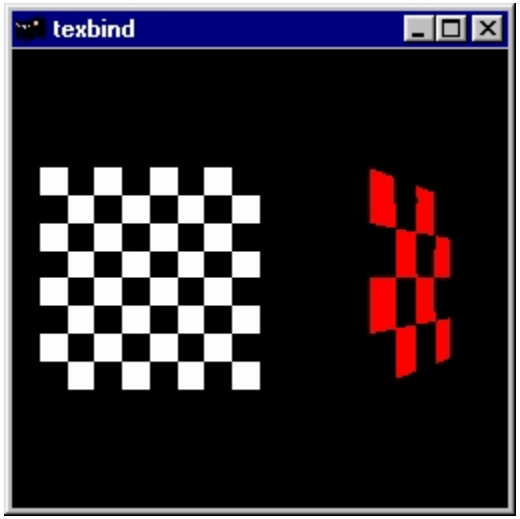
Using several texture images




```
static GLubyte image0[64][64][4], image1[64][64][4];
static GLuint  texname[2];

void init(void) {
…
        glGenTextures(2, texName);

        glBindTexture(GL_TEXTURE_2D, texName[0]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, checkImageWidth, checkImageHeight, 0,
                GL_RGBA, GL_UNSIGNED_BYTE, image0);

        glBindTexture(GL_TEXTURE_2D, texName[1]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, checkImageWidth, checkImageHeight, 0,
                GL_RGBA, GL_UNSIGNED_BYTE, image1);
}
```

Using several texture images

– Swithcing using glBindTexture(GL_TEXTURE_2D, texName);

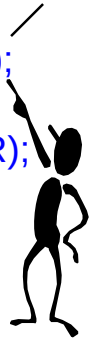



```
void display(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glBindTexture(GL_TEXTURE_2D, texName[0]);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
        glTexCoord2f(0.0, 1.0);  glVertex3f(-2.0, 1.0, 0.0);
        glTexCoord2f(1.0, 1.0);  glVertex3f(0.0, 1.0, 0.0);
        glTexCoord2f(1.0, 0.0);  glVertex3f(0.0, -1.0, 0.0);
        glEnd();

        glBindTexture(GL_TEXTURE_2D, texName[1]);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
        glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
        glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
        glEnd();

        glFlush();
}
```

OpenGL Mipmap

Incorporating MIPmapping into OpenGL applications is surprisingly easy.

```
// Boilerplate Texture setup code

glTexImage2D(GL_TEXTURE_2D, 0, 4, texWidth, texHeight, 0,
GL_RGBA,GL_UNSIGNED_BYTE, data);
gluBuild2DMipmaps(GL_TEXTURE_2D, 4, texWidth, texHeight, GL_RGBA,
GL_UNSIGNED_BYTE, data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
                                                GL_LINEAR_MIPMAP_LINEAR
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```



OpenGL also provides a facility for specifying the MIPmap image at each level using multiple calls to the glTexImage*D() function. This approach provides more control over filtering in the MIPmap construction.

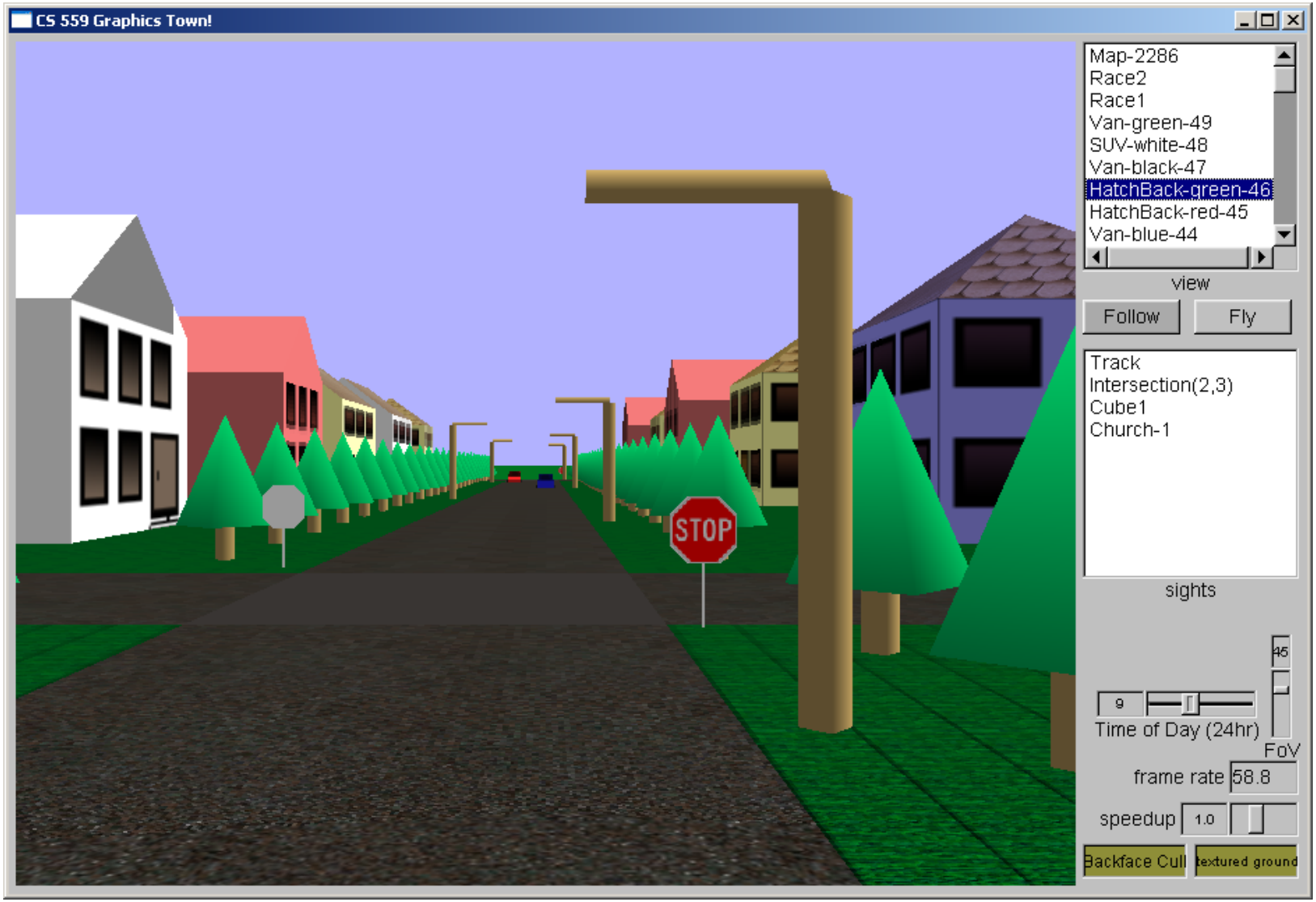
The gluBuildMipmaps() utility routine will automatically construct a mipmap from a given texture buffer. It will filter the texture using a simple box filter and then subsample it by a factor of 2 in each dimension. It repeats this process until one of the texture's dimensions is 1. Each texture ID, can have multiple levels associated with it. GL_LINEAR_MIPMAP_LINEAR trilinearly interploates between texture indices and MIPmap levels. Other options include GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, and GL_LINEAR_MIPMAP_NEAREST.

Slides from Jingyi Yu

Initializing Texture Mapping

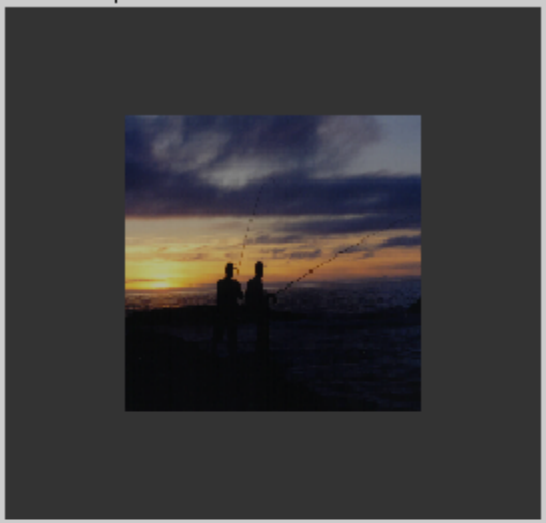
- Red book chapter on Texture mapping, Example 9-1
- All Red book example source code can be found at http://www.opengl.org/resources/code/samples/redbook/
- Course Tutorial 10, http://pages.cs.wisc.edu/~cs559-1/Tutorial10.htm

Project 3


CS 559 Graphics Town!
Map-2286
Race2
Race1
Van-green-49
SUV-white-48
Van-black-47
HatchBack-green-46
HatchBack-red-45
Van-blue-44
view
Follow
Fly
Track
Intersection(2,3)
Cube1
Church-1
sights
45
9
Time of Day (24hr)
FoV
frame rate
58.8
speedup
1.0
Backface Cull
textured ground
STOP

Texture

Screen-space view



Texture-space view

Command manipulation window

```
glMatrixMode(GL_TEXTURE);
glTranslatef( 0.00  , 0.00  , 0.00  );
   glRotatef( 0.0   , 0.00  , 0.00  , 1.00  );
    glScalef( 1.00  , 1.00  , 1.00  );
glMatrixMode(GL_MODELVIEW);


glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);
glColor4f(    0.60  , 0.60  , 0.60  , 0.60  );
glBegin(GL_POLYGON);
glTexCoord2f( 0.0   , 0.0   ); glVertex3f( -1.0 , -1.0 , 0.0   );
glTexCoord2f( 1.0   , 0.0   ); glVertex3f( 1.0  , -1.0 , 0.0   );
glTexCoord2f( 1.0   , 1.0   ); glVertex3f( 1.0  , 1.0  , 0.0   );
glTexCoord2f( 0.0   , 1.0   ); glVertex3f( -1.0 , 1.0  , 0.0   );
glEnd();
```

Click on the arguments and move the mouse to modify values.

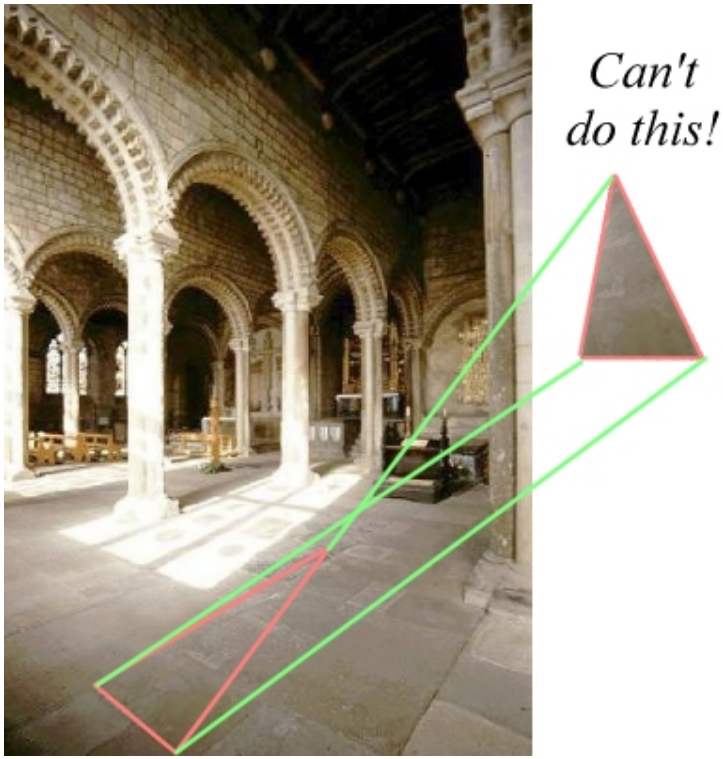
http://www.xmission.com/~nate/tutors.html

Texture animation

- Basic idea: treat texture coordinate like color
  - Moving water texture to simulate flow
  - zoom, rotation, and shearing image on a surface

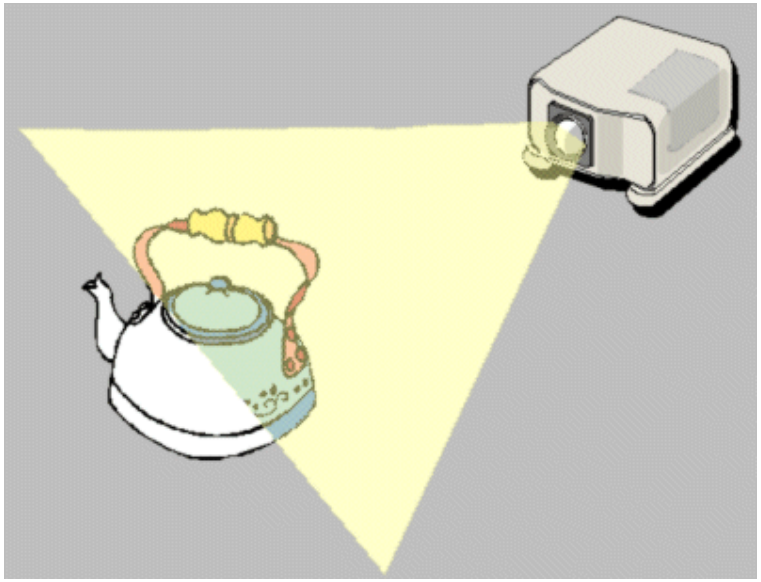
Other Issues with Textures

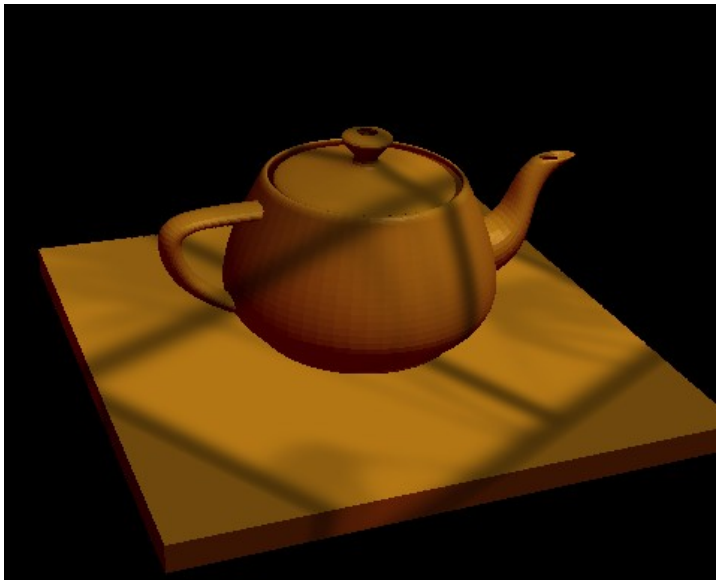
- Tedious to specify texture coordinates for every triangle
- Textures are attached to the geometry
- Can't use just any image as a texture
- **The "texture" can't have projective distortions**
  Reminder: linear interpolation in image space is not equivalent to linear interpolation in 3-space (This is why we need "perspective-correct" texturing).
  The converse is also true.
- Makes it hard to use pictures as textures



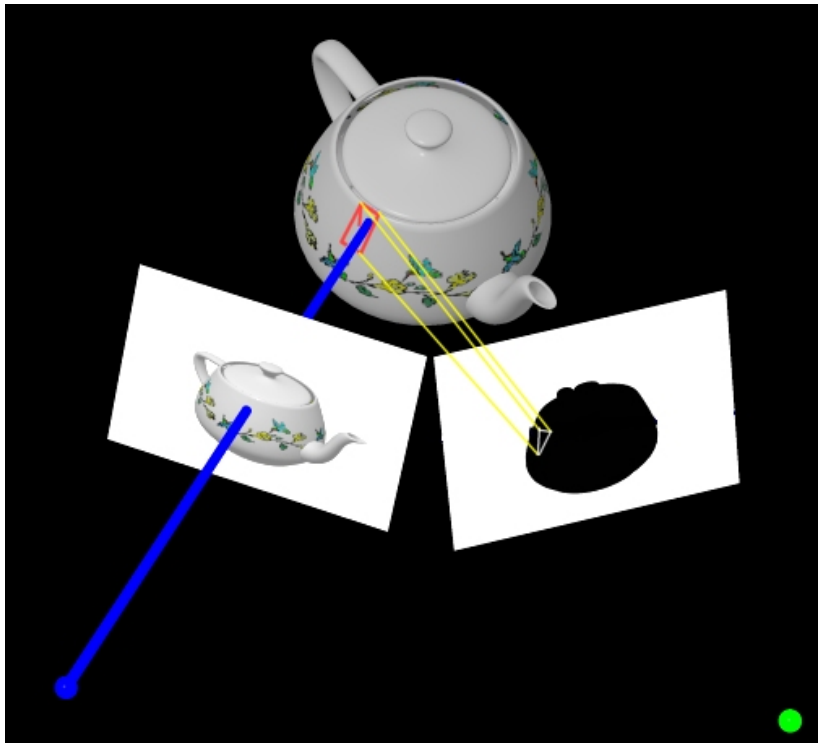
Projective Textures

- Treat the texture as a light source (like a slide projector)
- No need to specify texture coordinates explicitly
- A good model for shading variations due to illumination (cool spotlights)
- A fair model for view-dependent reflectance (can use pictures)





The Mapping Process



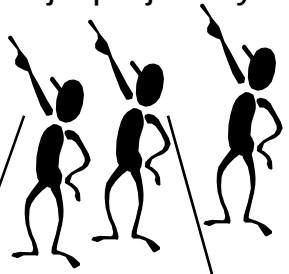
This is the same process, albeit with an additional transform, as perspective correct texture mapping. Thus, we can do it for free! Almost.

What transformation do we need?

- OpenGL is able to insert this extra projection transformation for textures by including another matrix stack, called GL_TEXTURE.
- Also we can use 3d vertex coordinate as texture coordinates
- The transform we want is:

$$T_{eye} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} P_{proj} V_{proj} M_{eye-to-world}$$

This extra matrix maps from normalized device coordinates ranging from [-1,1] to valid texture coordinates ranging from [0,1].



This matrix undoes the world-to-eye transform on the MODEL_VIEW matrix stack., so the projective texture can be specified in world coordinates. Note: If you specify your lights in eye-space then this matrix is identity.

This matrix specifies the frustum of the projector . It is a non-affine, projection matrix. You can use any of the projection transformations to establish it, such as glFrustum(), glOrtho() or gluPerspective () .

This matrix positions the projector in the world, much like the viewing matrix positions the eye within the world. (HINT, you can use gluLookAt() to set this up if you want.

OpenGL Example

Here is a code fragment implementing projective textures in OpenGL

```
// The following information is associated with the current active texture
// Basically, the first group of setting says that we will not be supplying texture coordinates.
// Instead, they will be automatically established based on the vertex coordinates in “EYE-SPACE”
// (after application of the MODEL_VIEW matrix).

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);

// These calls initialize the TEXTURE_MAPPING function to identity. We will be using
// the Texture matrix stack to establish this mapping indirectly.

float [] eyePlaneS = { 1.0f, 0.0f, 0.0f, 0.0f };
float [] eyePlaneT = { 0.0f, 1.0f, 0.0f, 0.0f };
float [] eyePlaneR = { 0.0f, 0.0f, 1.0f, 0.0f };
float [] eyePlaneQ = { 0.0f, 0.0f, 0.0f, 1.0f };

glTexGenfv(GL_S, GL_EYE_PLANE, eyePlaneS);
glTexGenfv(GL_T, GL_EYE_PLANE, eyePlaneT);
glTexGenfv(GL_R, GL_EYE_PLANE, eyePlaneR);
glTexGenfv(GL_Q, GL_EYE_PLANE, eyePlaneQ);
```

GL_OBJECT_LINEAR vs GL_EYE_LINEAR

When the mode is GL_OBJECT_LINEAR, the texture coordinate is calculated using the the plane-equation coefficients that are passed via `glTexGenfv(GL_S, GL_OBJECT_PLANE, planeCoefficients)`.

The coordinate (S in this case) is computed as:

```
S = Ax + By + Cz + D
```

using the [x y z] coordinates of the vertex (the values passed to glVertex).

If [A B C] is a unit vector, this means that the texture coordinate S is equal to the distance of the vertex from the texgen plane.

GL_EYE_LINEAR works similarly, with the coefficients passed via `glTexGenfv(GL_S, GL_EYE_PLANE, planeCoefficients)`.

The difference is that GL_OBJECT_LINEAR operates in "object coordinates", while GL_EYE_LINEAR works in "eye coordinates".

This means that the texture coordinates computed with GL_OBJECT_LINEAR depend solely on the values passed to glVertex, and do not change as the object (or camera) moves via transformations.

The texture coordinates computed with GL_EYE_LINEAR use the positions of vertices after all modeling & viewing transformations - i.e. they use the positions of the vertices on the screen.

http://resumbrae.com/ub/dms424_s05/10/print.html

OpenGL Example (cont)

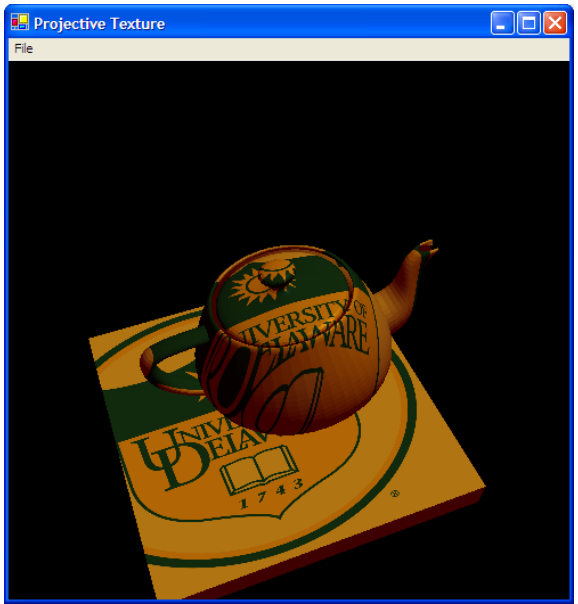
The following code fragment is inserted into Draw( ) or Display( )

```
if (projTexture) {
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_GEN_R);
    glEnable(GL_TEXTURE_GEN_Q);
    projectTexture();
}

// … draw everything that the texture is projected onto

if (projTexture) {
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);
    glDisable(GL_TEXTURE_GEN_R);
    glDisable(GL_TEXTURE_GEN_Q);
 }
```

OpenGL Example (cont)

Here is where the extra “Texture” transformation on the vertices is inserted.

```
private void projectTexture() {

    glMatrixMode(GL_TEXTURE);
    glLoadIdentity();
    glTranslated(0.5, 0.5, 0.5);    // Scale and bias the [-1,1] NDC values
    glScaled(0.5, 0.5, 0.5);        // to the [0,1] range of the texture map
    gluPerspective(15, 1, 5, 7);      // projector "projection" and view matrices
    gluLookAt(lightPosition[0],lightPosition[1],lightPosition[2], 0,0,0, 0,1,0);
    glMatrixMode(GL_MODELVIEW);

}
```



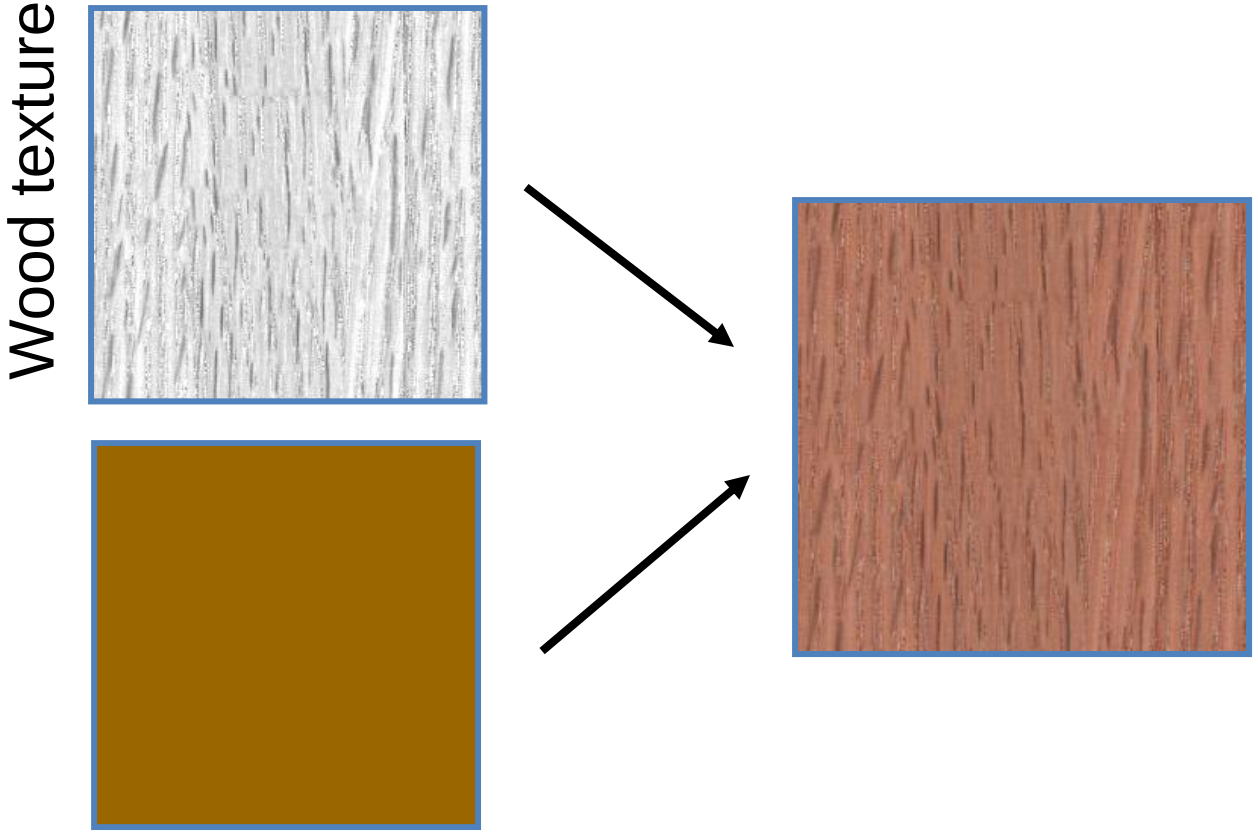
How to know where the light is to project an captured image?

Outline

- *Types of mappings*
- Interpolating texture coordinates
- Texture Resampling
- Texture mapping in OpenGL
- *Broader use of textures*

Modulation textures

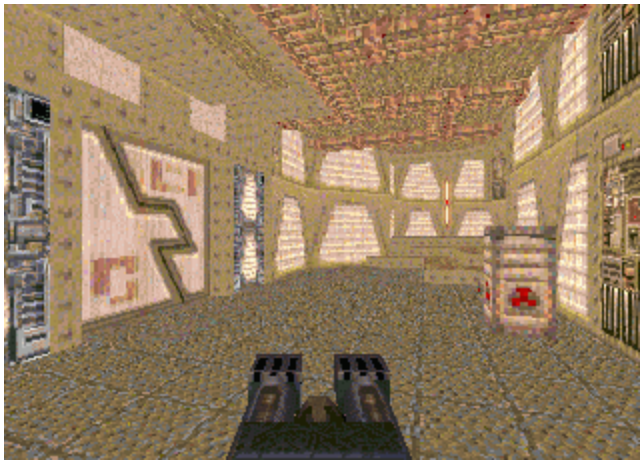
Map texture values to scale factor

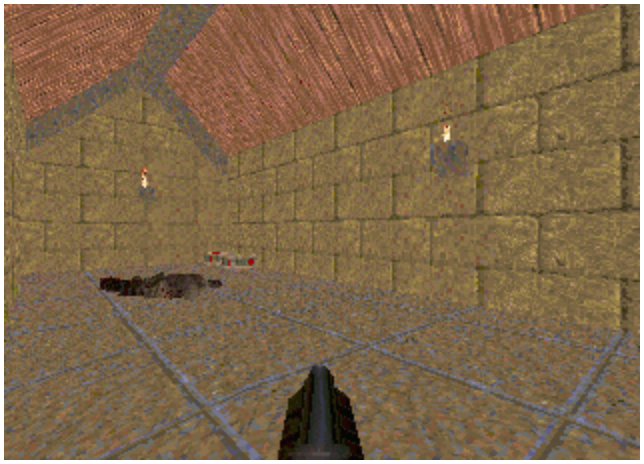


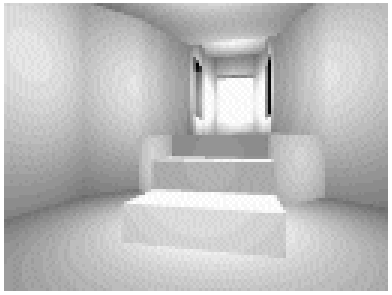
Illumination Maps

- Quake introduced *illumination maps* or *light maps* to capture lighting effects in video games

Texture map:





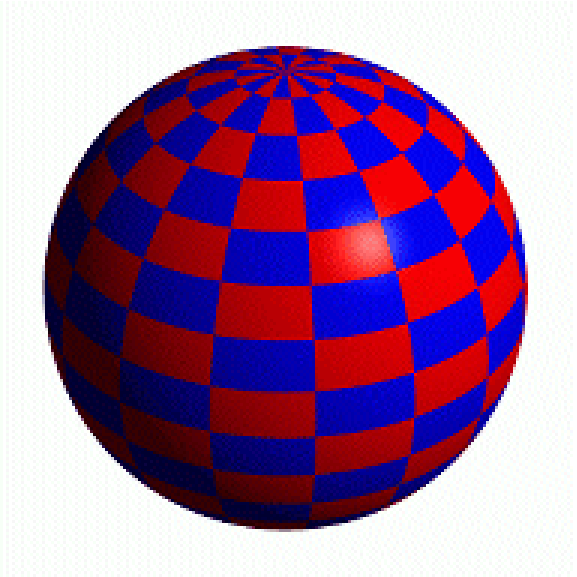


Light map

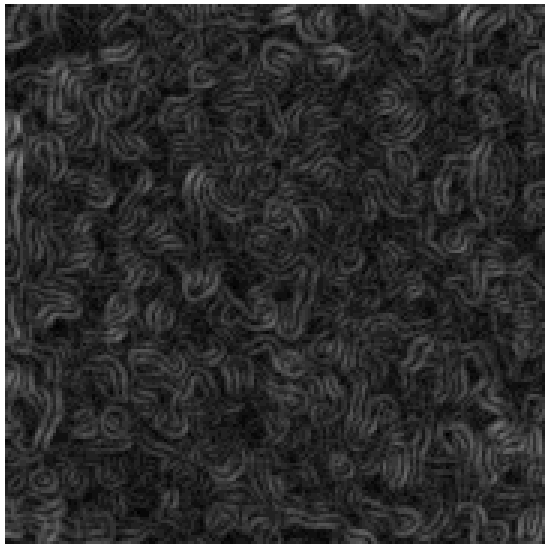
Texture map + light map:

Bump Mapping

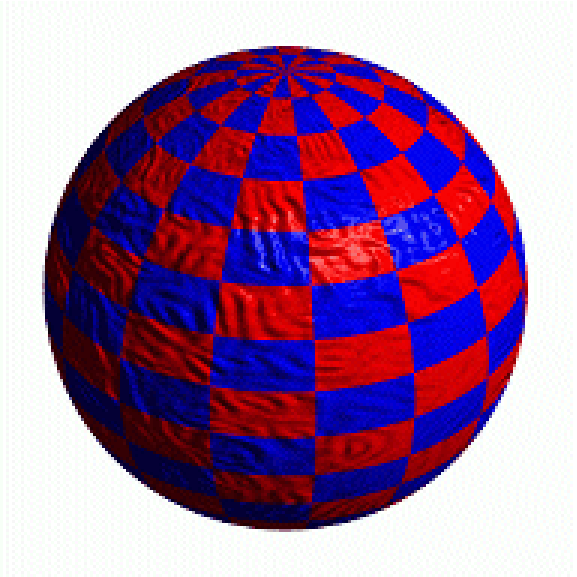
- Texture = change in surface normal!



Sphere w/ diffuse texture

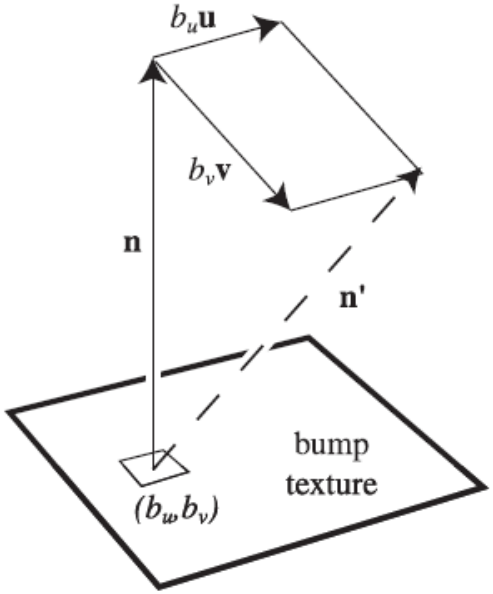


Swirly bump map

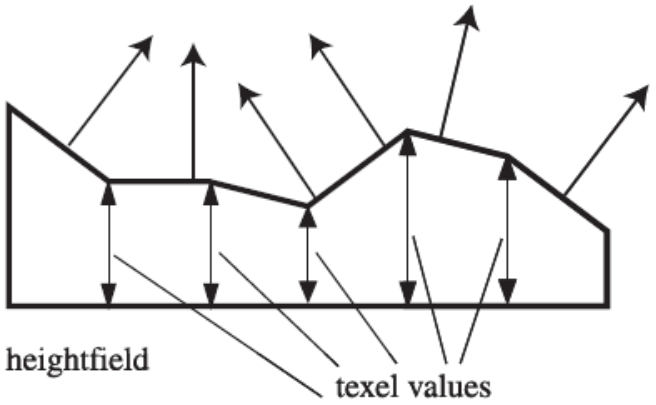


Sphere w/ diffuse texture and swirly bump map

Bump map respresentation



(bu, bv) as texel values



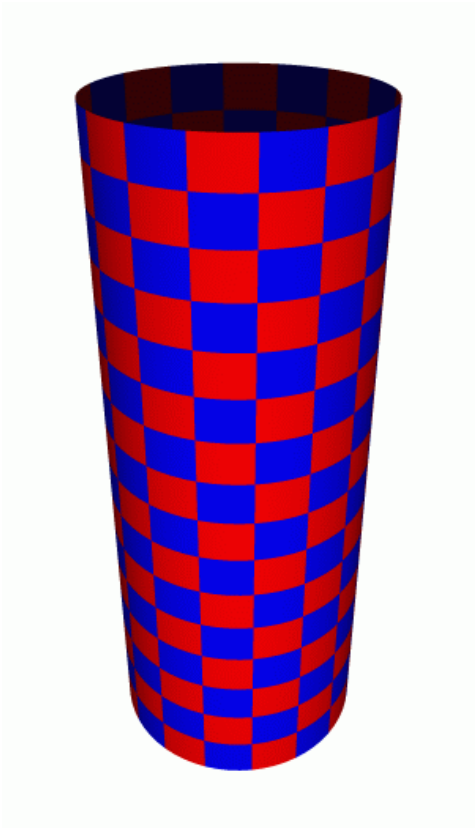
High values as texel values need to converted to normal

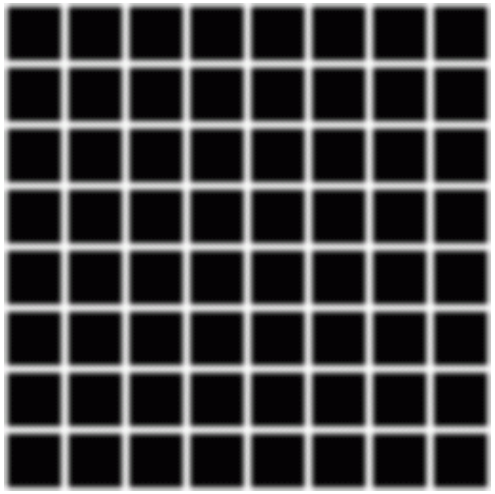
What happens if the light is head on?

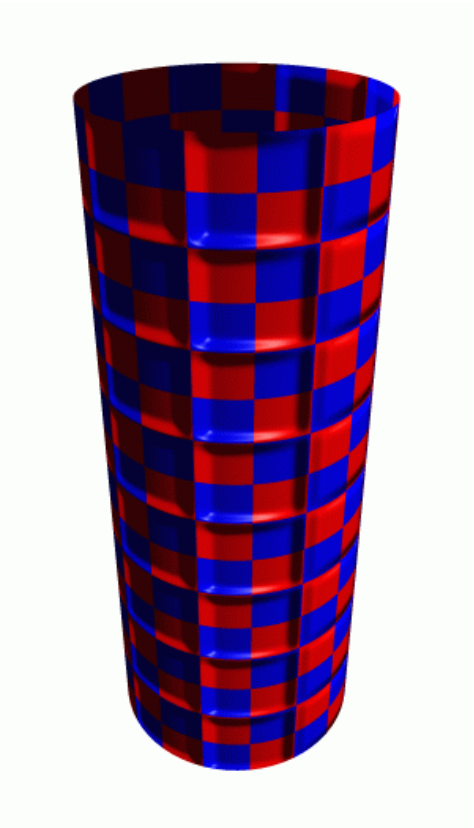
Figure 6.22 from RTR book

More Bump Map Examples

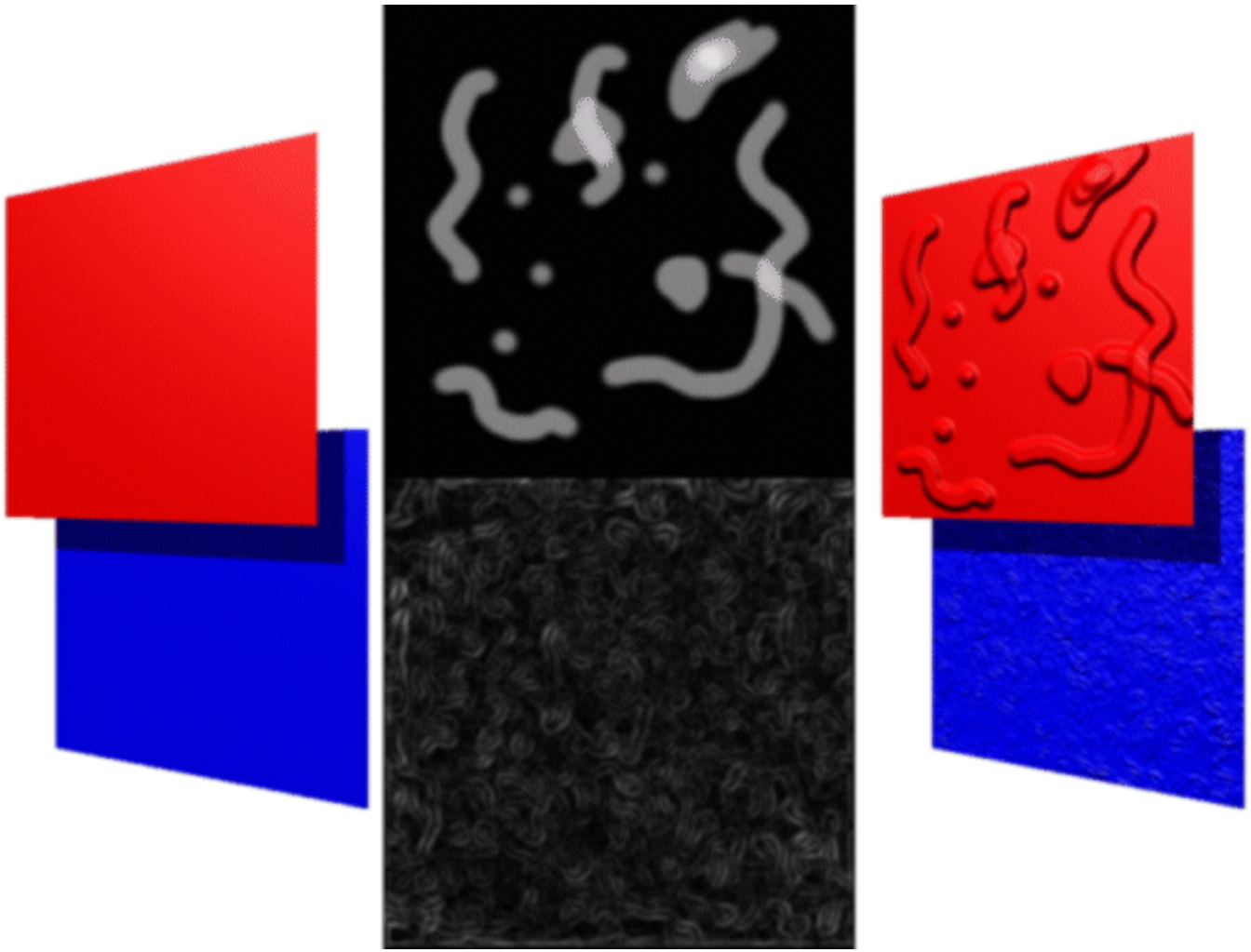
Since the actual shape of the object does not change, the silhouette edge of the object will not change. Bump Mapping also assumes that the Illumination model is applied at every pixel (as in Phong Shading or ray tracing).



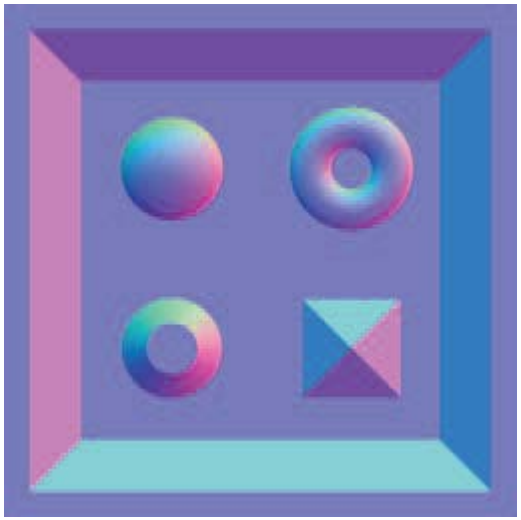




One More Bump Map Example



Dot product normal map



(r,g,b) encodes (nx,ny,nz)

Figure 6.24 from RTR book

Bump map in shading

$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_s \cdot \max(0, \mathbf{V} \cdot \mathbf{R})^{n_s}$$

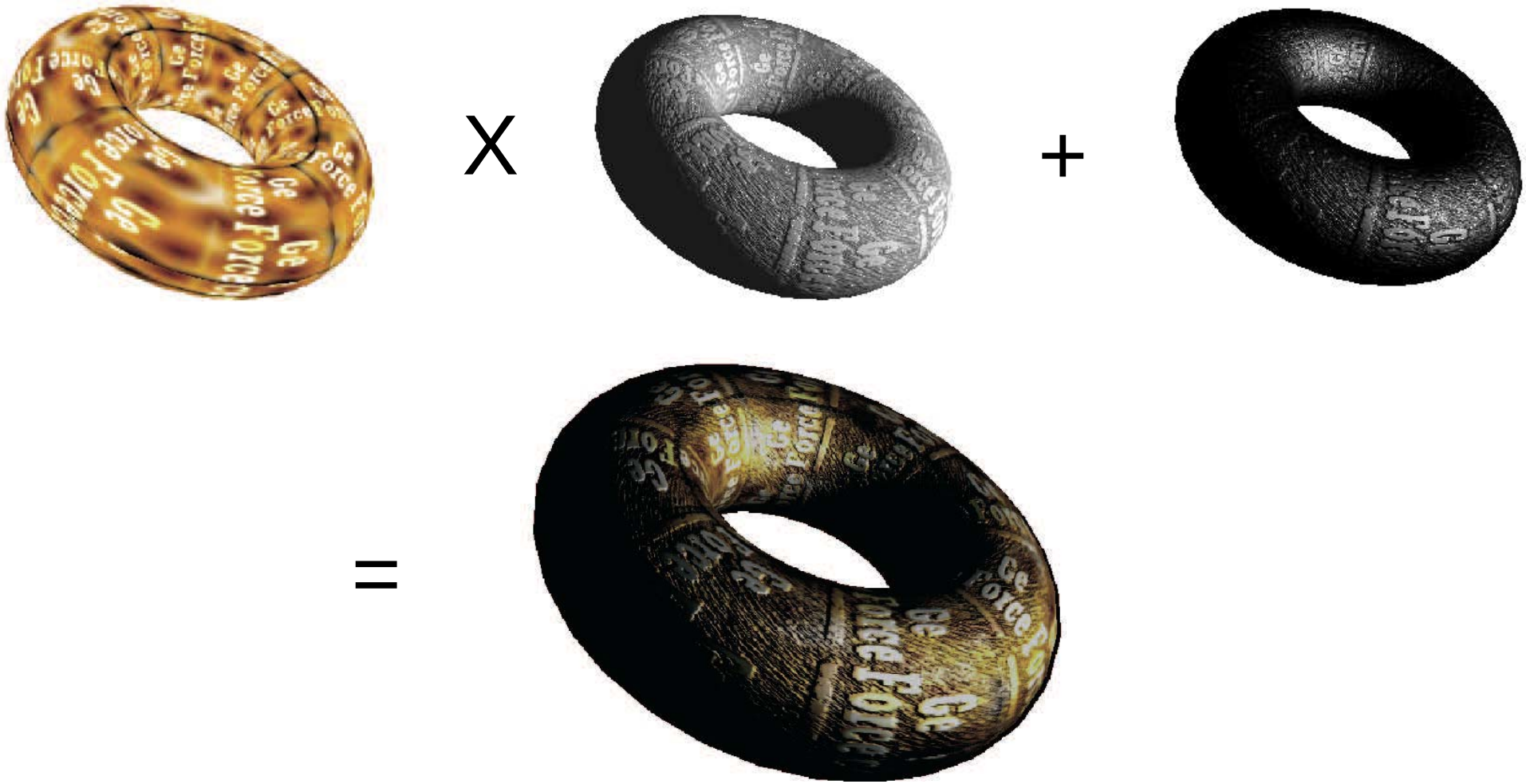
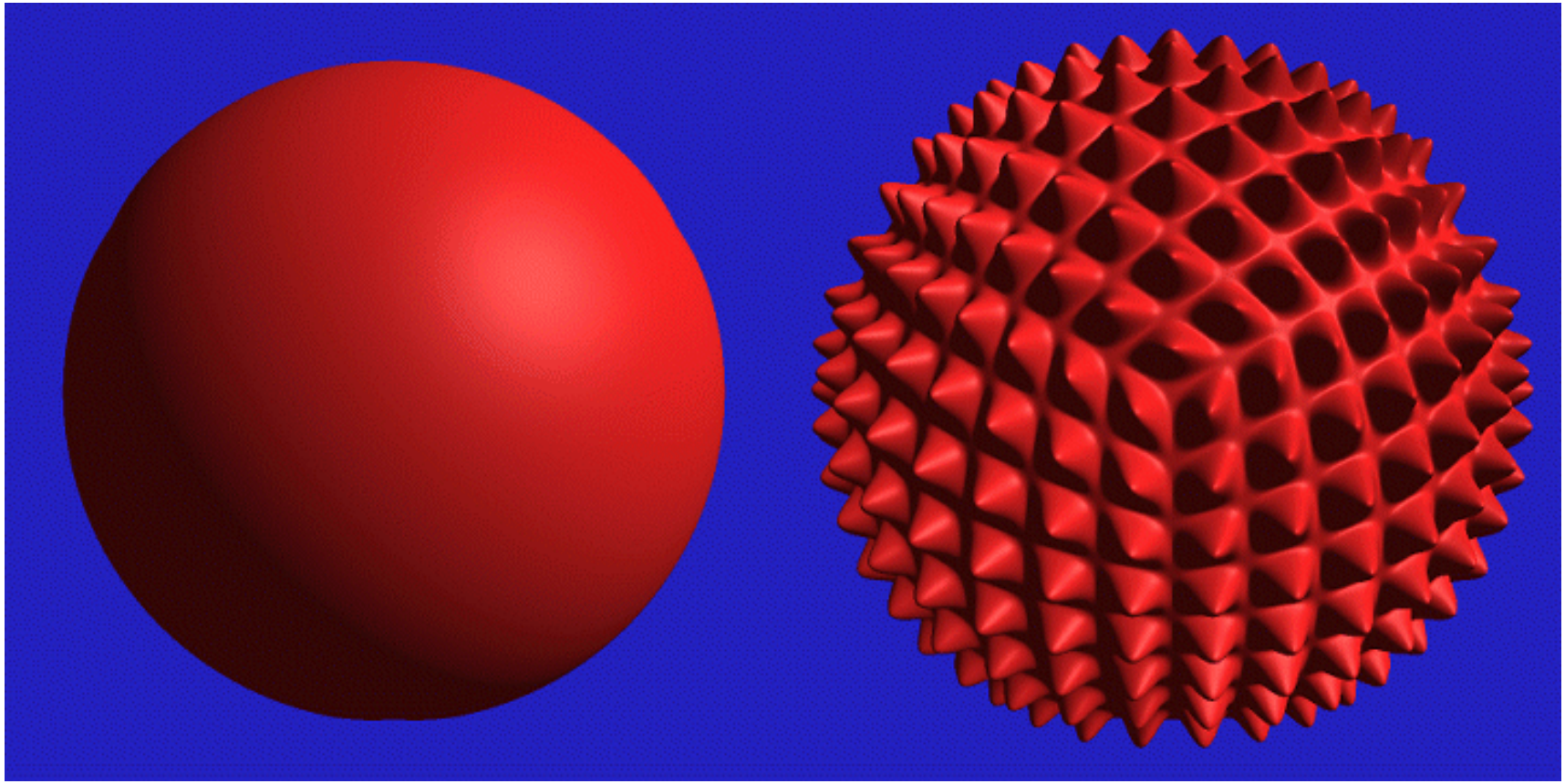


Figure 6.26 from RTR book

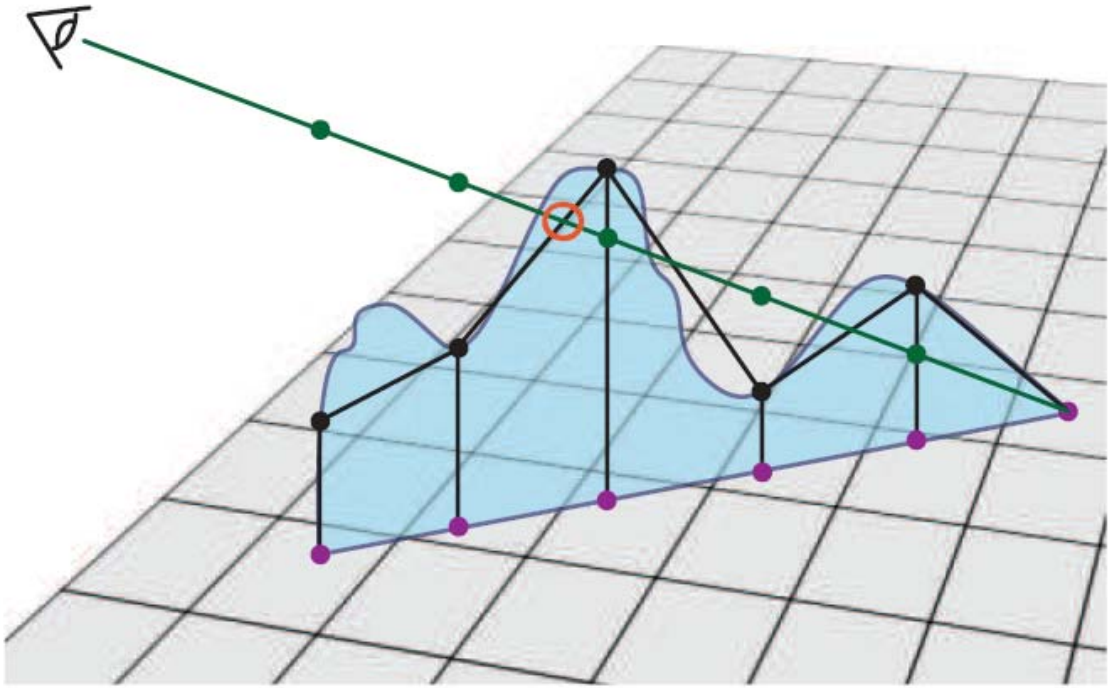
Displacement Mapping

Texture maps can be used to actually move surface points. This is called *displacement mapping*. How is this fundamentally different than bump mapping?



Bump map does not have occlusion effects.

Rendering Displacement map is tricky



How to find the intersection?
See RTR book

Figure 6.29 from RTR book

Bump map vs Displacement map

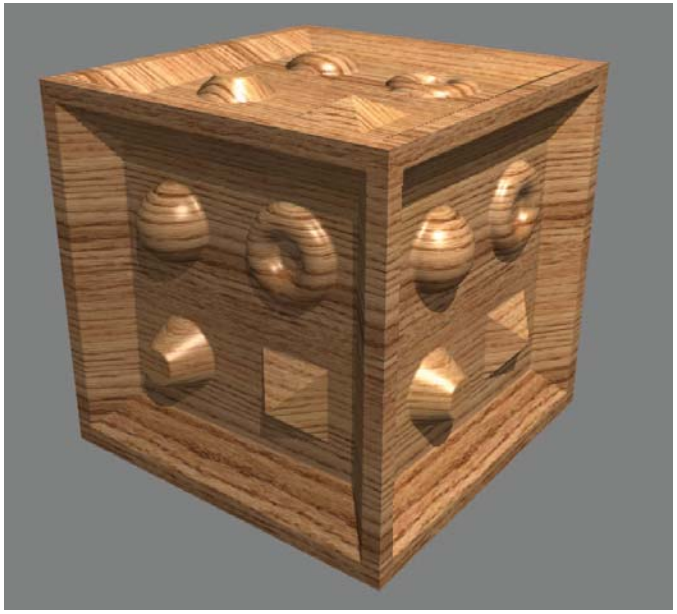


Figure 6.24 and 30 from RTR book