

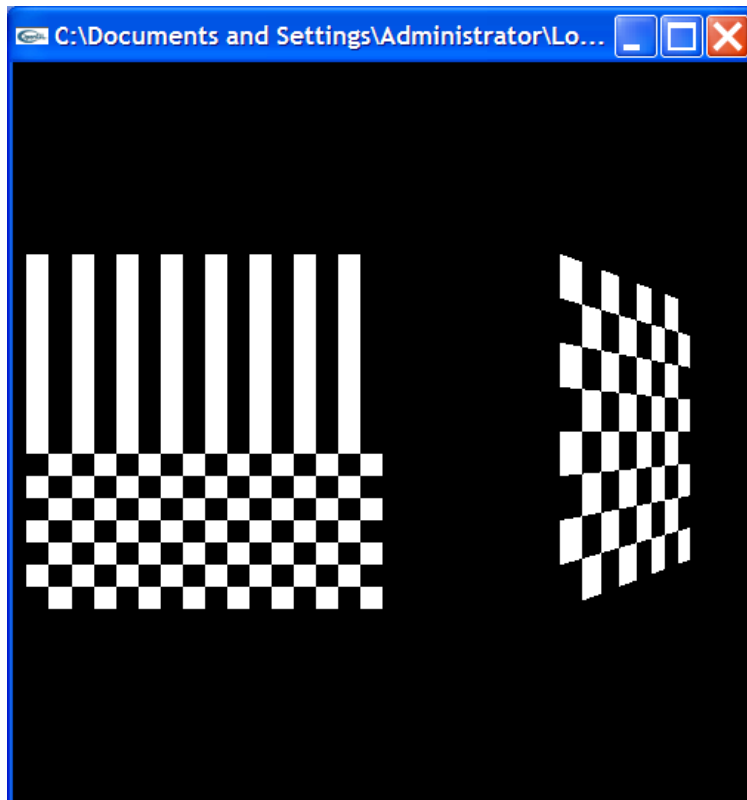
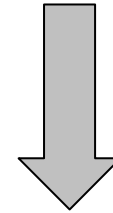
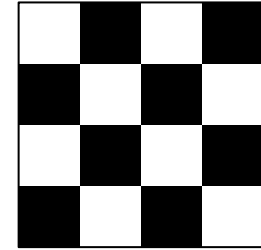
# COMP 371 – Winter 2012

# Computer Graphics

- Texture Mapping

# What is Texture Mapping?

- Allows you to attach an image to an object (i.e., polygon).
- Supports all object transformations.
- Texture is a rectangular array of data (colors).
- The individual values in a texture are called texels.



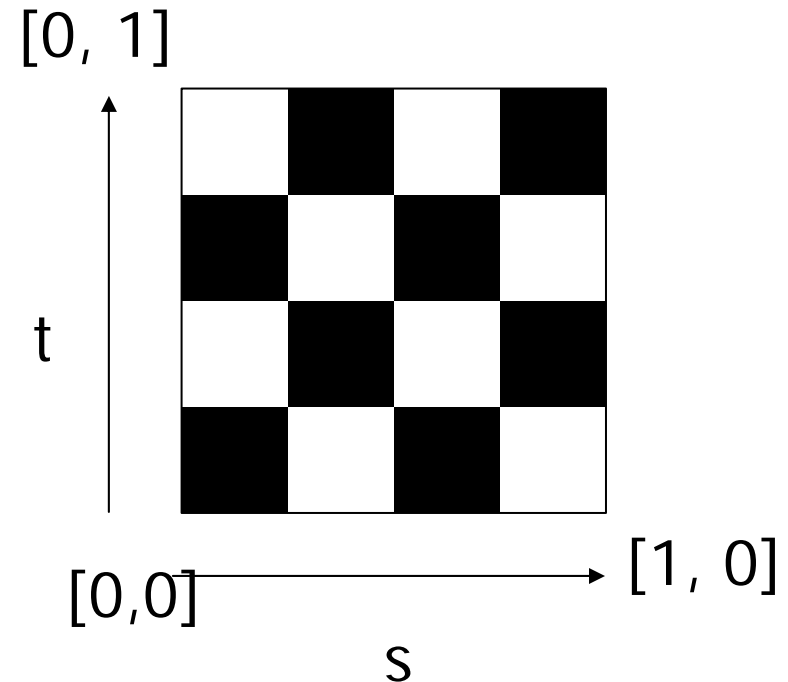
# Texture Coordinates

- A texture is usually addressed by values between zero and one
- The “addresses” of points on the texture consist of two numbers ( $s, t$ )
- A vertex can be associated with a point on the texture by giving it one of these *texture coordinates*

```
glTexCoord*(s, t);
glVertex*(x, y, z);
```

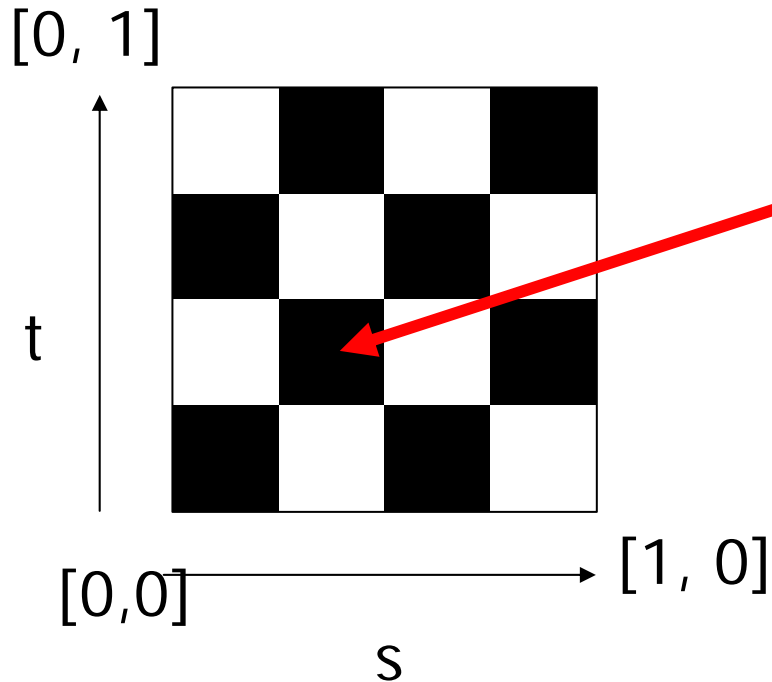
Texture coordinates part of state like colors and normals.

...although it is really just an array of pixels



$[s,t]$ -space  $\rightarrow$   $[u,v]$ -space of surface  $\rightarrow$   $[x,y,z]$ -space of surface

# Pixels & Texture Coordinates



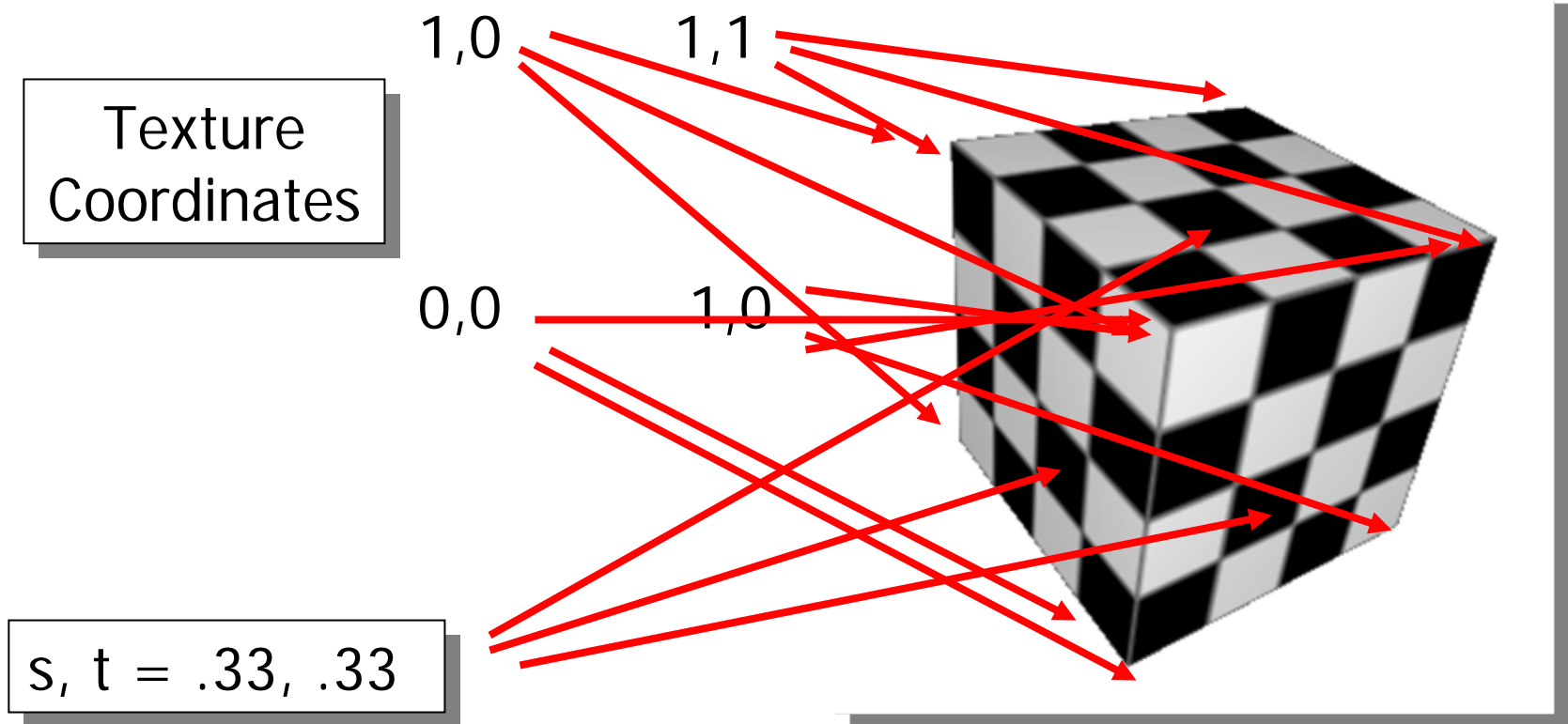
$s, t = .33, .33$

For example: a 32 x 32 pixel image

```
Glubyte mychecker[32][32][3];  
glEnable(GL_TEXTURE_2D);
```

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 32, 32,  
             0, GL_RGB, GL_UNSIGNED_BYTE, mychecker);
```

# Texture Coordinates to Polygons



- Each vertex of each polygon is assigned a texture coordinate.
- OGL finds the appropriate texture coordinate for points "between" vertices during rasterization. – same idea as smooth shading!

# Mapping Functions

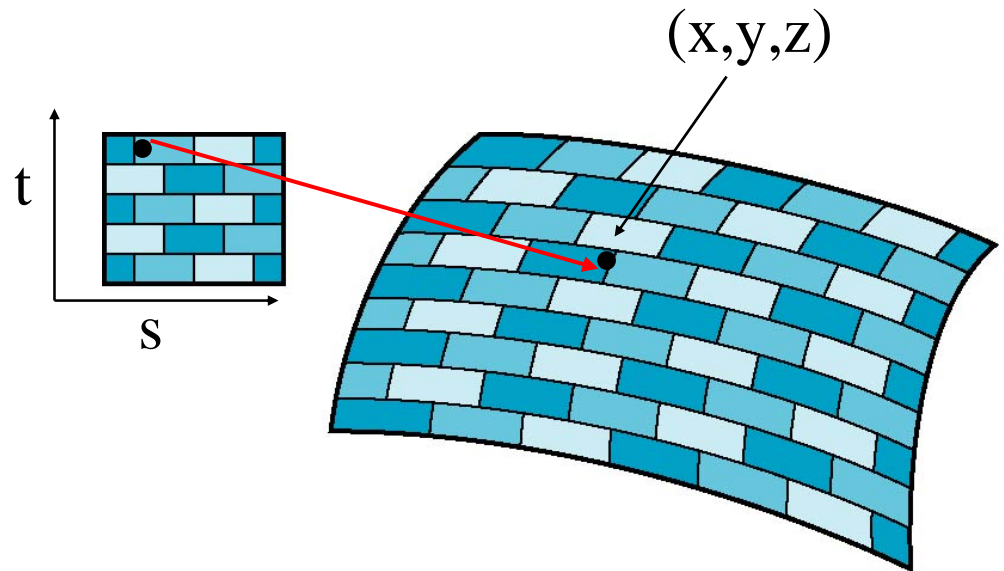
- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions

$$x = x(s,t)$$

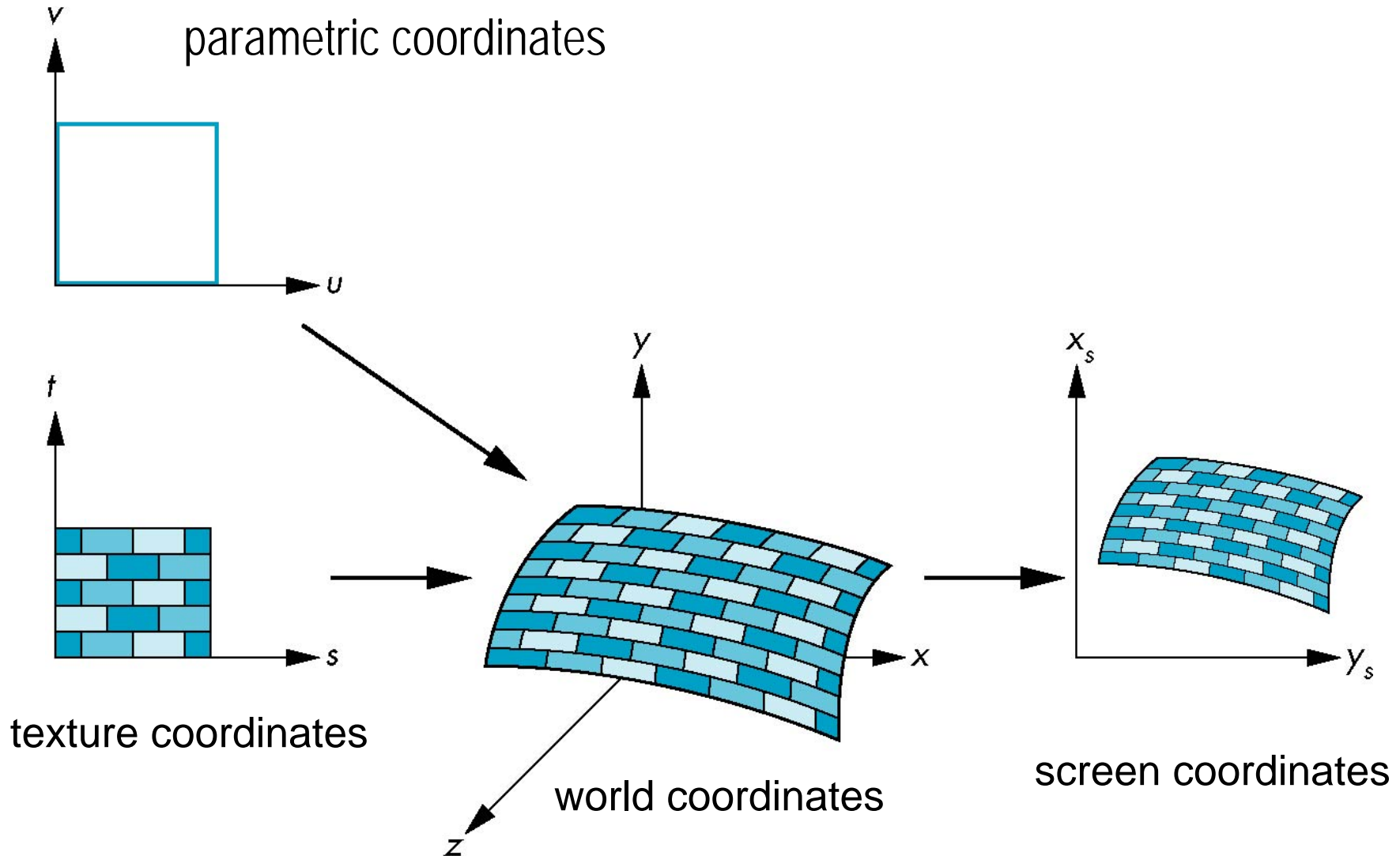
$$y = y(s,t)$$

$$z = z(s,t)$$

- But we really want to go the other way

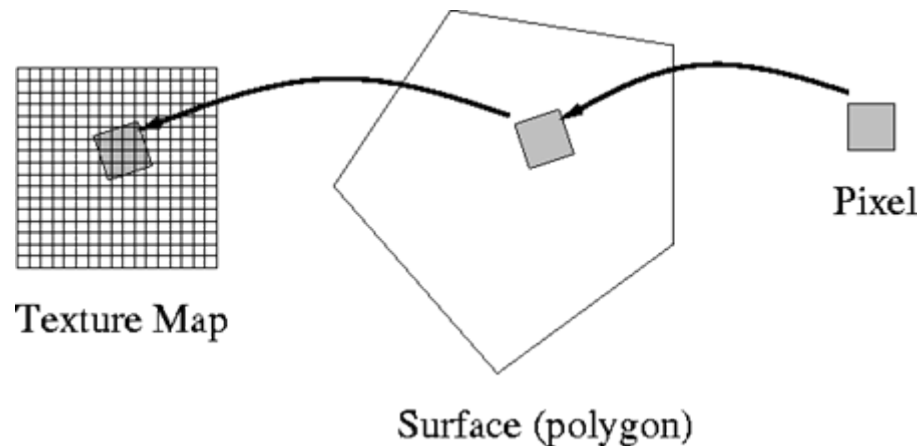


# Texture Mapping: Coordinates Systems



# Basic Idea of Mapping

- The texture lives in a 2D space
  - Parameterize points in the texture with 2 coordinates:  $(s, t)$
- Define the mapping from  $(x, y, z)$  in world space to  $(s, t)$  in texture space
  - To find the color in the texture, take an  $(x, y, z)$  point on the surface, map it into texture space, and use it to look up the color of the texture
- With polygons:
  - Specify  $(s, t)$  coordinates at vertices
  - Interpolate  $(s, t)$  for other points based on given vertices

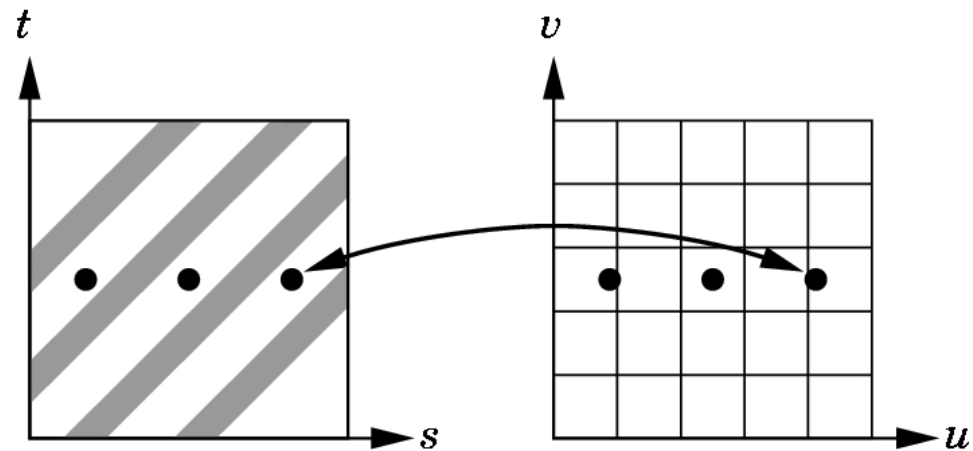
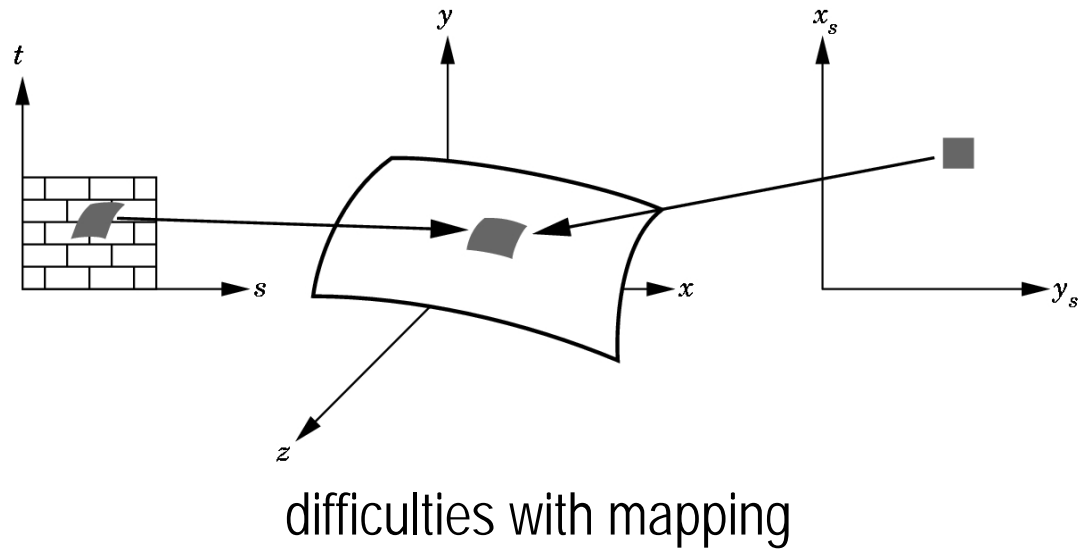


# Backward Mapping

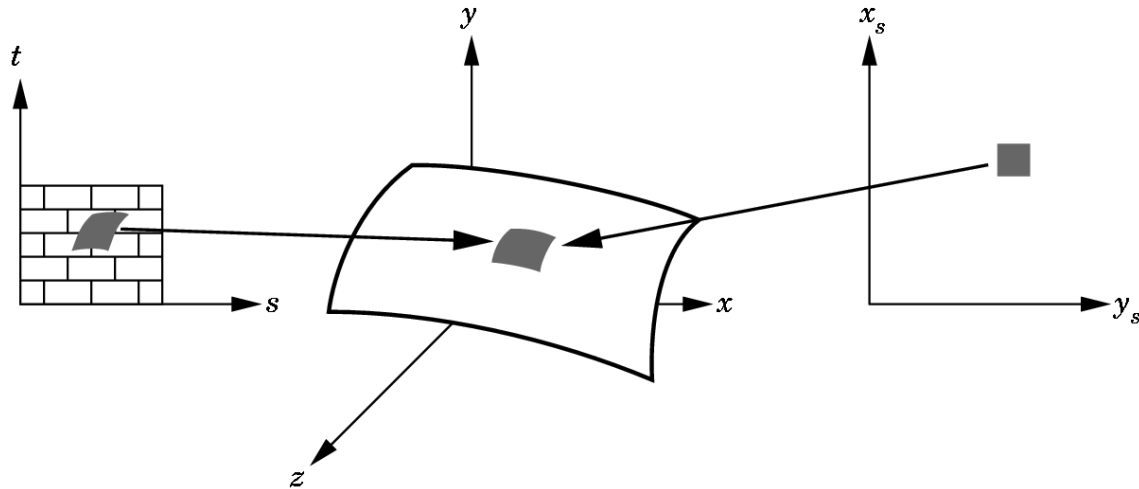
- We really want to go backwards
  - Given a pixel, we want to know to which point on an object it corresponds
  - Given a point on an object, we want to know to which point in the texture it corresponds
  - *Need a map of the form*
    - $s = s(x,y,z)$
    - $t = t(x,y,z)$
- Such functions are difficult to find in general

# Texture Mapping

- pixel  $(x_s, y_s)$  corresponds to  $(x, y, z)$  on "curved" object
- problems with finding inverse mapping



# Texture mapping transformation



Consider surface visible at current pixel.

Find the patch on the surface that corresponds to it.

- Map screen coord of pixel corners back to object
- Find texels that map to the surface patch
- If multiple texels lie on patch combine them:  
weighted average; supersampling with postfiltering

# Linear Mapping

Most curved surfaces represented parametrically as

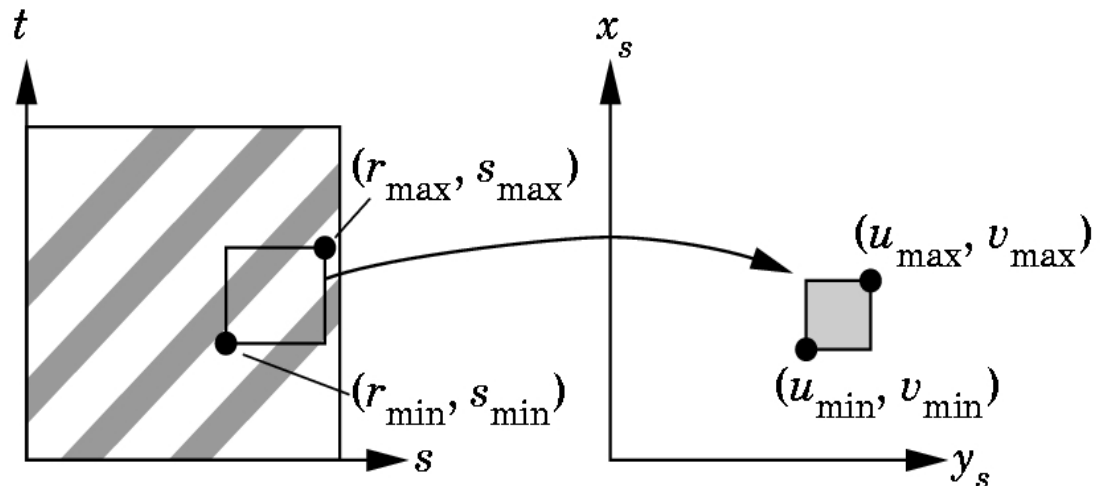
$$p(u,v) = [x(u,v), y(u,v), z(u,v)]$$

a point in the texture map  $T(s,t)$  is to be mapped to a point on the surface  $p(u,v)$  by a linear map

$$u = as + bt + c \qquad v = ds + et + f$$

(if  $ae \neq bd$  mapping is invertible)

- mapping is easy to use
- it does not respect the curvature of the object

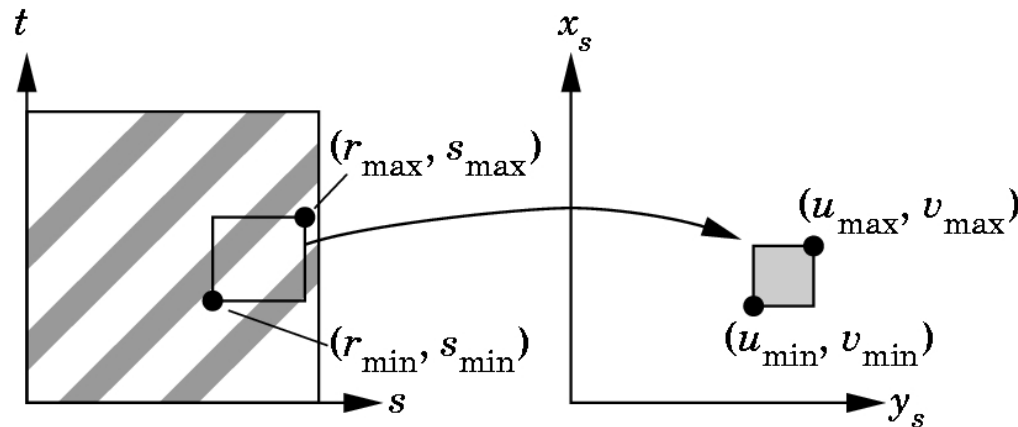


# Linear Mapping

- Linear mapping is defined as:

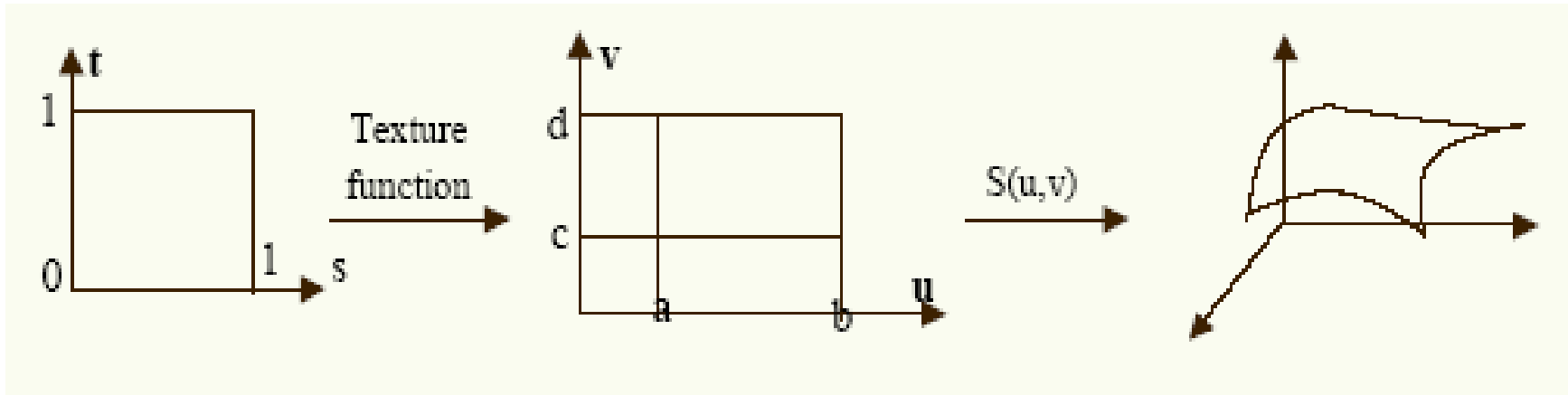
$$u = u_{\min} + \frac{s - s_{\min}}{s_{\max} - s_{\min}} (u_{\max} - u_{\min})$$

$$v = v_{\min} + \frac{t - t_{\min}}{t_{\max} - t_{\min}} (v_{\max} - v_{\min})$$



# Curved Surface Texture Coordinates

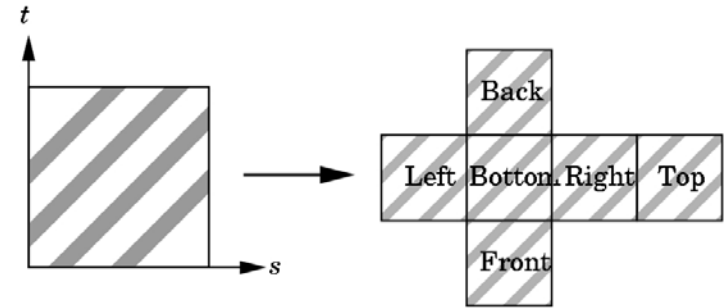
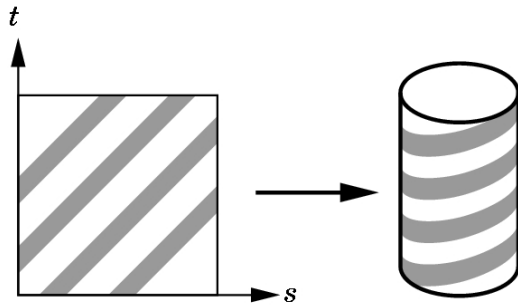
- Assume a curved parametric surface  
$$S = \{x(u, v), y(u, v), z(u, v)\}$$
approximated by a polygonal mesh
- Need to find appropriate texture coordinates for mesh vertices



# Mapping texture to a surface using an intermediate surface

- **Two-step mapping**
  - Map the texture to a simple intermediate surface (sphere, cylinder, cube)
  - Map the intermediate surface (with the texture) onto the surface being rendered

# Two-step mapping example



- parametric form cylinder:  $x = r \cos(2 \pi u)$   
 $y = r \sin(2 \pi u)$   
 $z = v h$

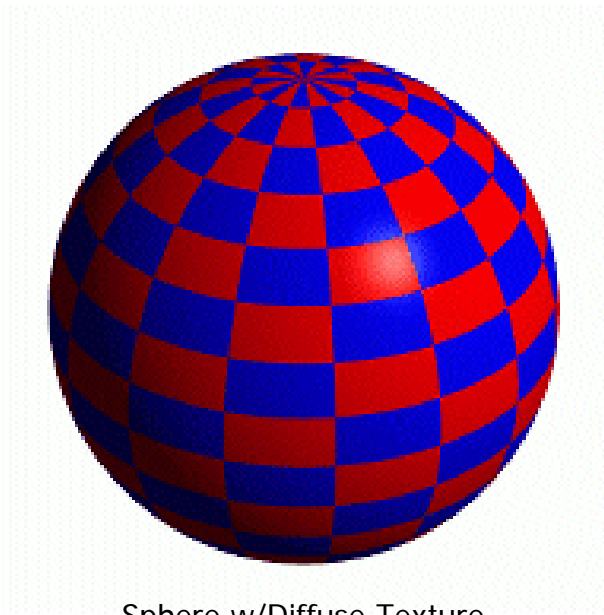
$$0 \leq u, v \leq 1$$

first step:  $u = s, v = t$

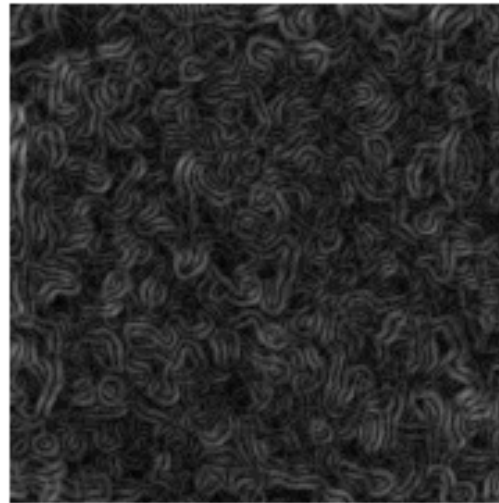
- sphere
- cube

# Bump Mapping

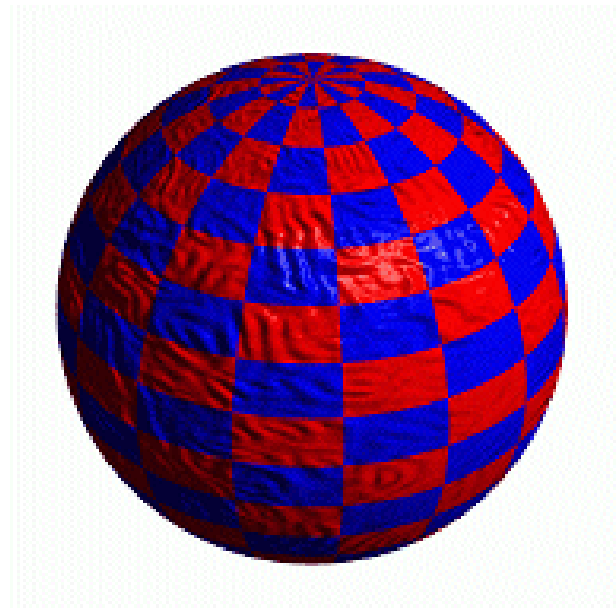
- Use textures to alter the surface normal
  - Does not change the actual shape of the surface
  - Just shaded as if it were a different shape



Sphere w/Diffuse Texture



Swirly Bump Map



Sphere w/Diffuse Texture & Bump Map

# Bump mapping

- Bump map is in texture array:  $b(u,v) \ll 1$
- $\mathbf{p}$  point on the surface corresponding to texture coordinates  $u,v$ .
- $\mathbf{N}$  the normal at  $\mathbf{p}$
- $\mathbf{p}'$  the bump point for  $\mathbf{p}$   
$$\mathbf{p}' = \mathbf{p} + b(u,v)\mathbf{N}$$

We actually do not “bump” the surface, just the normal at  $\mathbf{p}$ .

- $\mathbf{N}'$  the normal at  $\mathbf{p}'$ . This normal used by the illumination model at  $\mathbf{p}$ .

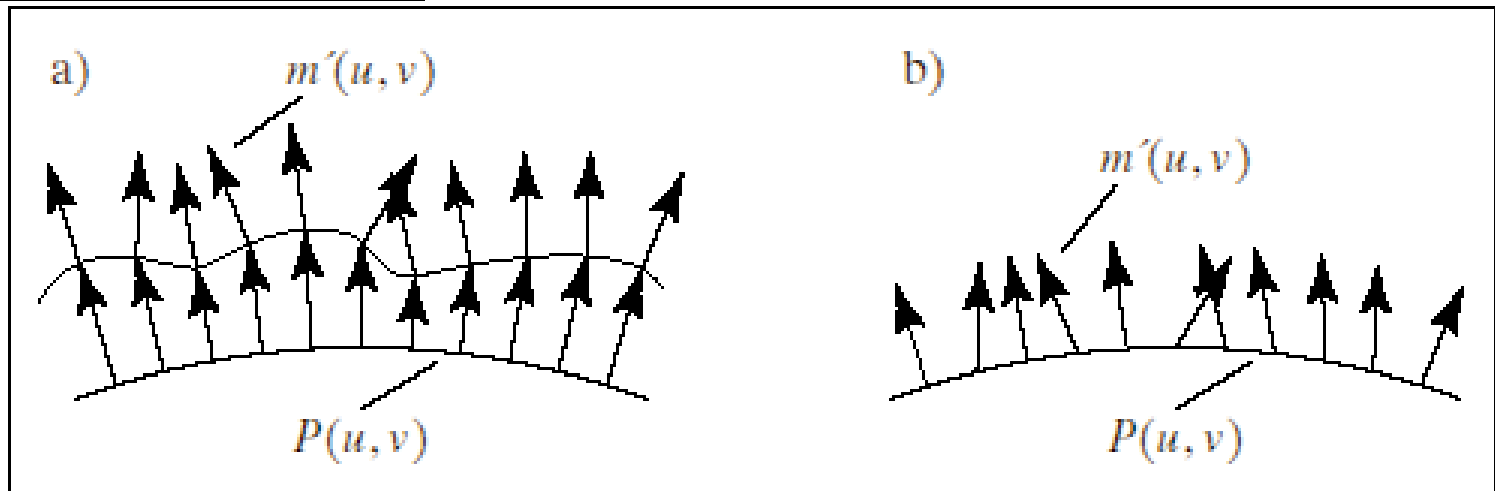
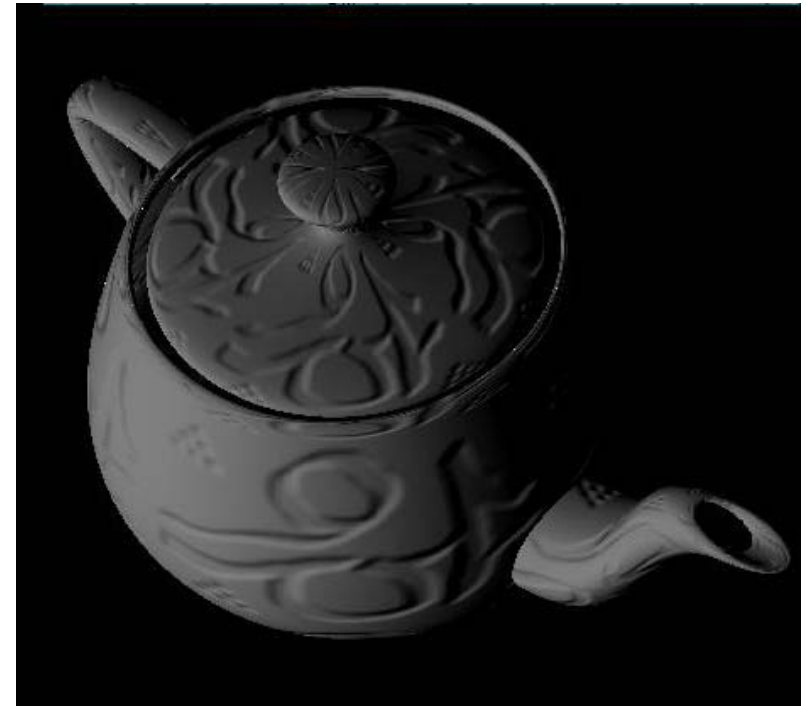
# Bump Mapping: How?

- Idea: Perturb pixel normals  $N(u, v)$  derived from object geometry to get additional detail for shading
- Compute lighting per pixel (like Phong)

Height map

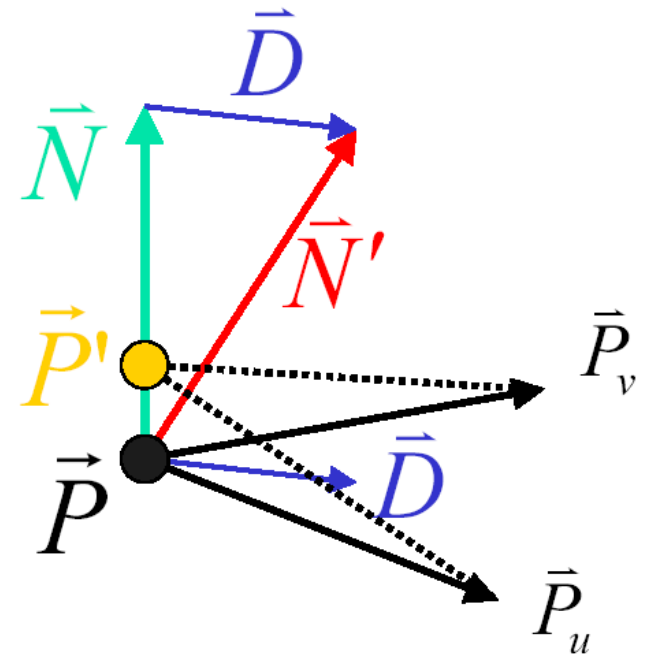
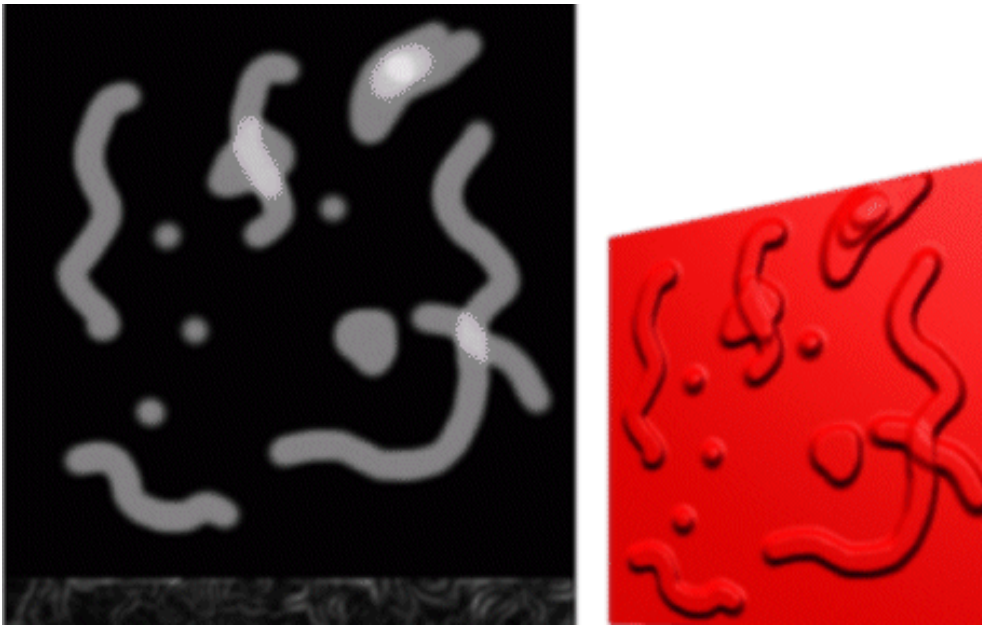


Bump texture applied to teapot



# Bump Mapping

- Treat the texture as a single-valued height function
- Compute the normal from the partial derivatives in the texture

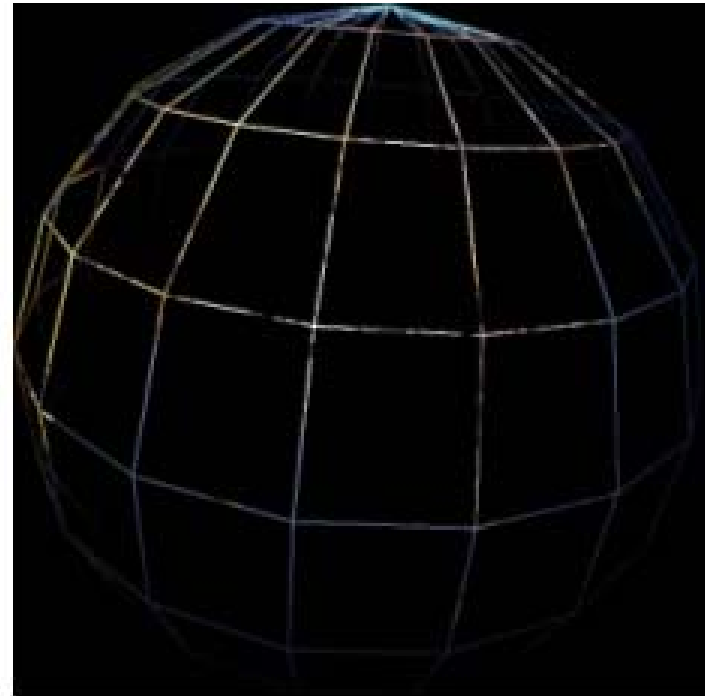
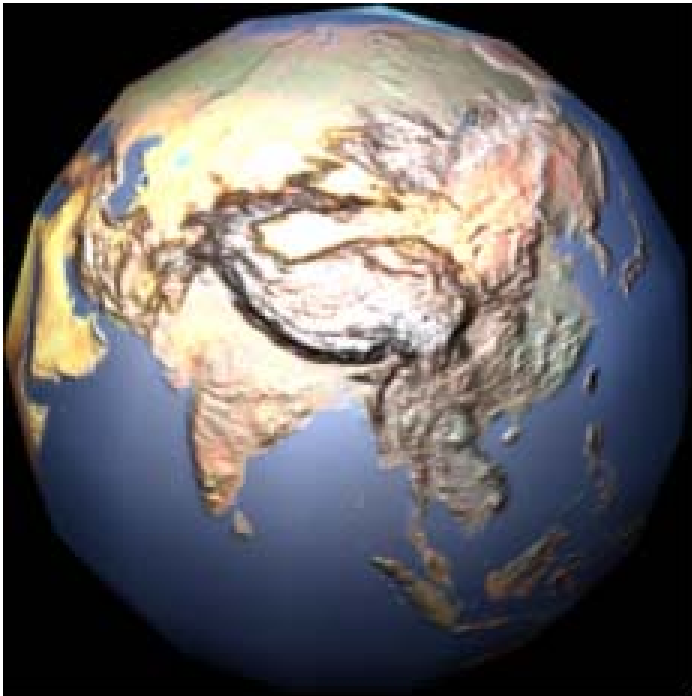


# Bump mapping

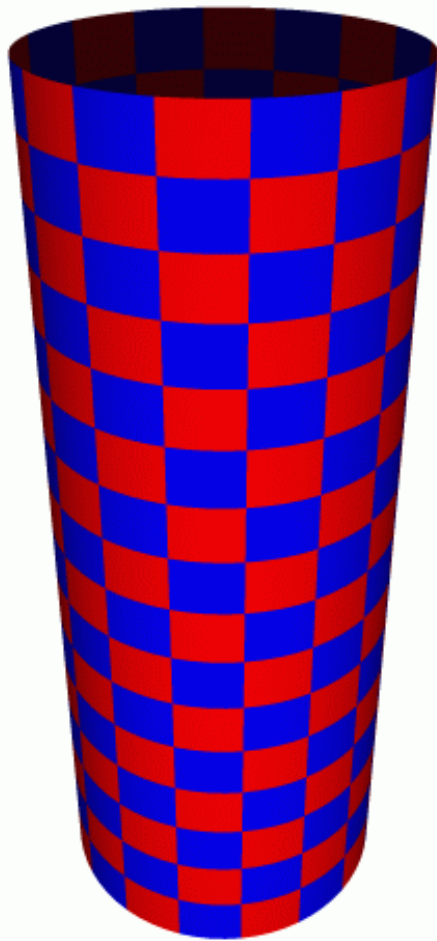
- 2D Texture map creates odd looking rough surfaces
- Bump mapping: texture map that alters surface normals.
  - Use texture array to set a function which perturbs surface normals
  - Altered normals match a bumpy surface
  - Applying illumination model to the new normals shades the bumps correctly

# Bump mapping: Why?

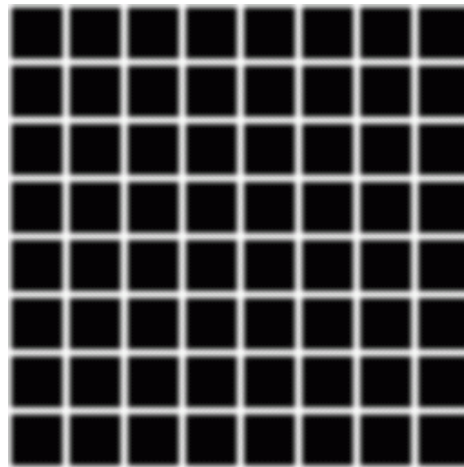
- Can get a lot more surface detail without expense of more object vertices to light, transform



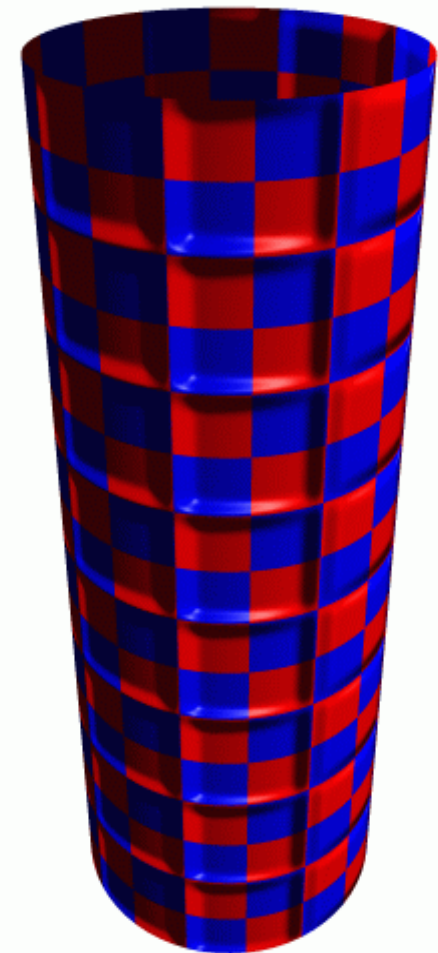
# Another Bump Map Example



Cylinder w/Diffuse Texture Map



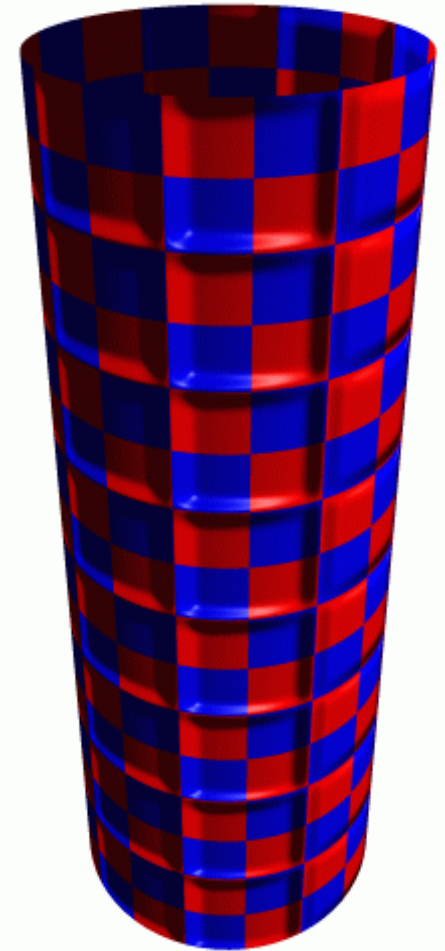
Bump Map



Cylinder w/Texture Map & Bump Map

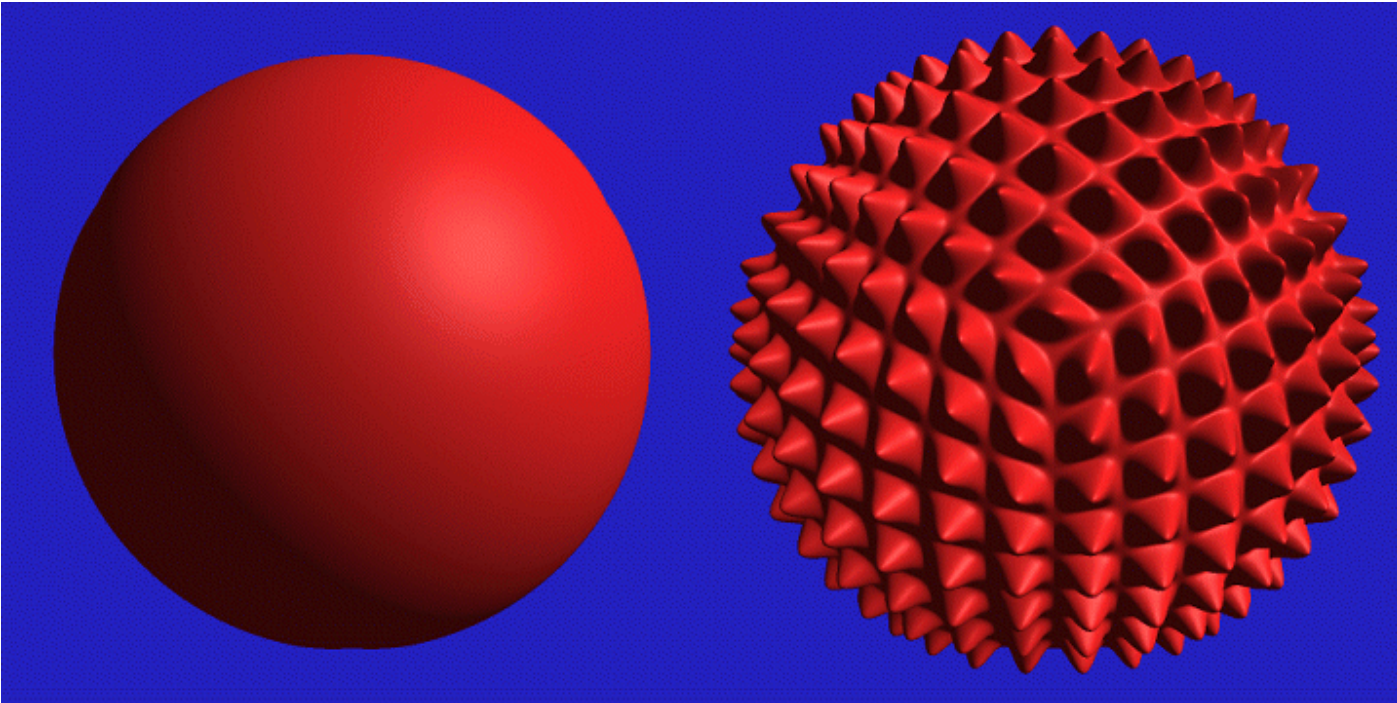
# What's Missing?

- There are no bumps on the silhouette of a bump-mapped object
- Bump maps don't allow self-occlusion or self-shadowing



# Displacement Mapping

- Use the texture map to actually move the surface point
- The geometry must be displaced before visibility is determined



# Displacement Mapping



Image from:

*Geometry Caching for  
Ray-Tracing Displacement Maps*

by Matt Pharr and Pat Hanrahan.

*note the detailed shadows  
cast by the stones*

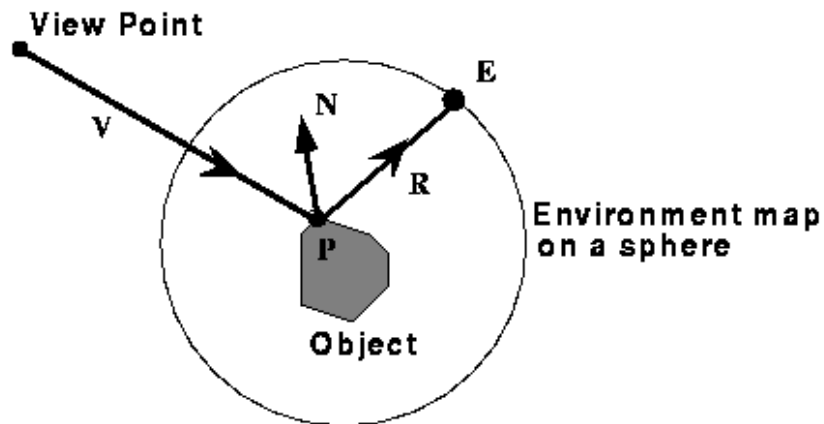
# Displacement Mapping



Ken Musgrave

# Environment Maps

- We can simulate reflections by using the direction of the reflected ray to index a spherical texture map at "infinity".
- Assumes that all reflected rays begin from the same point.



# What's the Best Chart?

*Box Map*



*Latitude Map*



*GL Map*



# Programming with Texture Maps

1. Create texture and load with `glTexImage()` by either
  - a. read a jpeg, bmp, .... file
  - b. define texture within application
  - c. copy image from color buffer

2. Define parameters as to how texture is applied  
`glTexParameter*()`

*Next slides describe the options here....*  
See Table 9.6 in OGL Red Book

3. Enable texture maps  
`glEnable(GL_TEXTURE_2D)`
4. Define texture coordinates for vertices  
`glTexCoord*(s,t);`  
`glVertex*(x,y,z);`

# Steps in Texture Mapping

## 1. Specify the Texture

GLvoid **glTexImage2D** (GLenum target, GLint level, GLint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid \*pixels);

## 2. Indicate How the Texture is to be Applied to Each Pixel

-- Indicate the function for computing the final RGBA value of the pixel from the object color and the texture-image data.

GLvoid **glTexParameterf** (GLenum target, GLenum pname, GLfloat param);

-- Define how the texture is attached to the object.

GLvoid **glTexEnvf** (GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GLfloat param);

-- Enable Texture Mapping

GLvoid **glEnable** (GL\_TEXTURE\_2D)

-- Draw the scene, supplying both texture and geometric coordinates

GLvoid **glTexCoord2f** (GLfloat xtex, GLfloat ytex);

# 2D Texturing

- Enable 2D Texturing:
  - `glEnable(GL_TEXTURE_2D)`
- Define a 2D Texture Map
  - `glTexImage2D(GL_TEXTURE_2D, level, components, width, height, border, format, type, texels)`
    - **level**: number of the texture resolutions. For now = 0
    - **components**: number of color components. (3 for RGB)
    - **width, height**: size of the texture map
    - **border**: with (1) or without (0) border
    - **format**: format of the texture data, e.g. GL\_RGB
    - **type**: data type of the texture data, e.g. GL\_BYTE
    - **textels**: pointer to the actual texture image data

# 2D Texturing

- *Texture objects* store texture data and keep it readily available for usage. Many texture objects can be generated.
- Generate identifiers for texture objects
  - **glGenTextures(num, textureNames)**
    - **num**: the number of texture objects identifiers to generate
    - **textureNames**: int array holding identifiers
- Bind a texture object and Following texture commands refer to the bound texture, e.g. **glTexImage2D()**
  - **glBindTexture(target, identifier)**
    - **Target**: can be `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, or `GL_TEXTURE_3D`
    - **Identifier**: a texture object identifier

# Texture Coordinates

- Every point on a surface should have a texture coordinate (s, t) in texture mapping
- We often specify texture coordinates to polygon vertices and interpolate texture coordinates with the polygon.
  - Use `glTexCoord2f(s,t)` to set texture coordinates:
    - E.g. `glTexCoord2f(0.5f,0.5f);`
    - `glVertex3f(x,y,z);`
  - Texture coordinates (s, t) are within a (0, 1) domain
  - Texture coordinates are state variables (similar to normal)

# Boundaries

- textures are assumed to be in a (0,0) to (1,1) domain in texture space. You can control what happens if a point maps to a texture coordinate outside of the domain
  - Repeat: Assume the texture is tiled
    - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)`
  - Clamp: Clamp to Edge: the texture coordinates are truncated to valid values, and then used
    - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)`
  - Can specify a special border color:
    - `glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, R,G,B,A)`

# Specify Texture Image

- Define a texture image from an array of *texels* (texture elements) in CPU memory

```
Glubyte my_texels[512][512];
```

- Define as any other pixel map
  - Scan
  - Via application code
- Enable texture mapping

```
glEnable(GL_TEXTURE_2D)
```

- OpenGL supports 1-4 dimensional texture maps

## Define Image as a Texture

```
glTexImage2D( target, level, components,  
             w, h, border, format, type, texels );
```

**target:** type of texture, e.g. **GL\_TEXTURE\_2D**

**level:** used for mipmapping (discussed later)

**components:** elements per texel

**w, h:** width and height of **texels** in pixels

**border:** used for smoothing (discussed later)

**format and type:** describe texels

**texels:** pointer to texel array

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512,  
            0, GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

# Converting A Texture Image

- OpenGL requires texture dimensions to be powers of 2
- If dimensions of image are not powers of 2

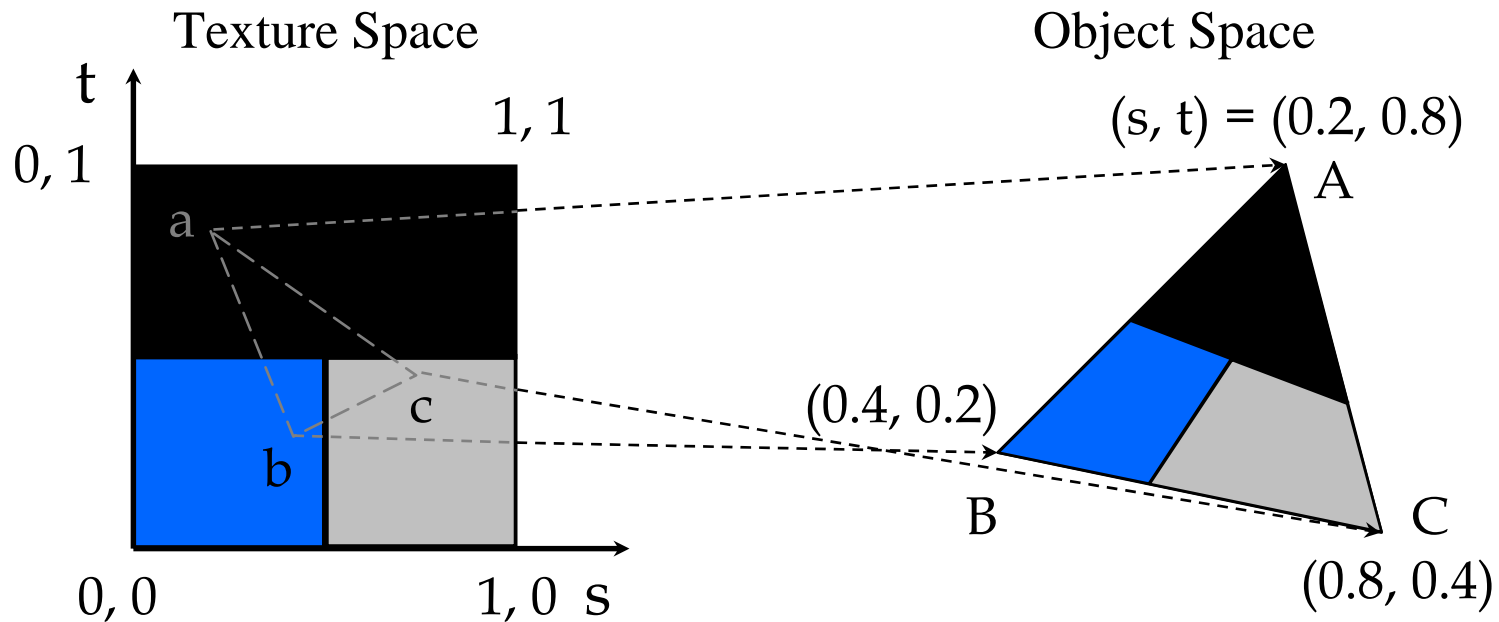
```
gluScaleImage( format, w_in, h_in,  
              type_in, *data_in, w_out, h_out,  
              type_out, *data_out );
```

- **data\_in** is source image
- **data\_out** is for destination image
- Image interpolated and filtered during scaling

# Mapping a Texture

- Based on parametric texture coordinates

**glTexCoord\* ( )** specified at each vertex



# Typical Code

```
glBegin(GL_POLYGON);  
    glColor3f(r0, g0, b0);  
    glNormal3f(u0, v0, w0);  
    glTexCoord2f(s0, t0);  
    glVertex3f(x0, y0, z0);  
    glColor3f(r1, g1, b1);  
    glNormal3f(u1, v1, w1);  
    glTexCoord2f(s1, t1);  
    glVertex3f(x1, y1, z1);  
    .  
    .  
glEnd();
```

Note that we can use vertex arrays to increase efficiency

# Generating Texture Coordinates

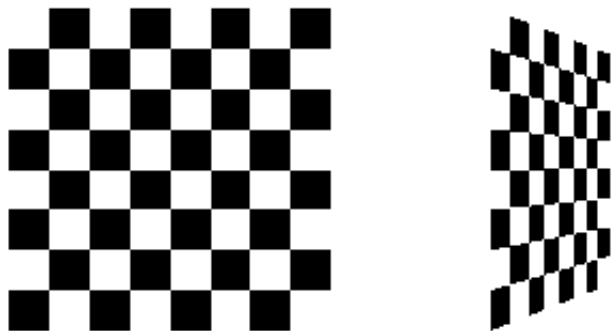
- OpenGL can generate texture coordinates automatically
  - **`glTexGen{ifd}[v]()`**
- specify a plane
  - generate texture coordinates based upon distance from the plane
- generation modes
  - **`GL_OBJECT_LINEAR`**
  - **`GL_EYE_LINEAR`**
  - **`GL_SPHERE_MAP`** (used for environmental maps)

# Make Checker Image

```

#define checkImageWidth 64
#define checkImageHeight 64
GLubyte checkImage[checkImageWidth][checkImageHeight][3];
void makeCheckImage(void) {
    int i, j, r, c;
    for (i = 0; i < checkImageWidth; i++)
        for (j = 0; j < checkImageHeight; j++) {
            c = (((i&0x8)==0)^(j&0x8)==0)*255;
            checkImage[i][j][0] = (GLubyte) c;
            checkImage[i][j][1] = (GLubyte) c;
            checkImage[i][j][2] = (GLubyte) c;
        }
}

```



# Specify the texture

```

makeCheckImage();
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexImage2D(
    GL_TEXTURE_2D,
    0,          // Use 1 resolution for mapping
    3,          // a texel has 3 components: RGB
    checkImageWidth,
    checkImageHeight,
    0,          // width of the border
    GL_RGB,
    GL_UNSIGNED_BYTE,
    &checkImage);

```

# Scale the Texture Image

- Texture width and height must be powers of 2
- If not, the texture image has to be **scaled** so as to satisfy that requirement

```
int gluScaleImage(  
    GLenum format, // e.g., GL_RGB  
    int widthin, int heightin, GLenum typein, void *datain,  
    int widthout, int heightout, GLenum typeout, void *dataout  
);
```

// typein, typeout: data type of the texture array