Texture Mapping

Ed Angel

Professor of Computer Science, Electrical and Computer Engineering, and Media Arts

University of New Mexico

Objectives

- Introduce Mapping Methods
  - Texture Mapping
  - Environment Mapping
  - Bump Mapping
- Consider basic strategies
  - Forward vs backward mapping
  - Point sampling vs area averaging

The Limits of Geometric Modeling

- Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena
  - Clouds
  - Grass
  - Terrain
  - Skin

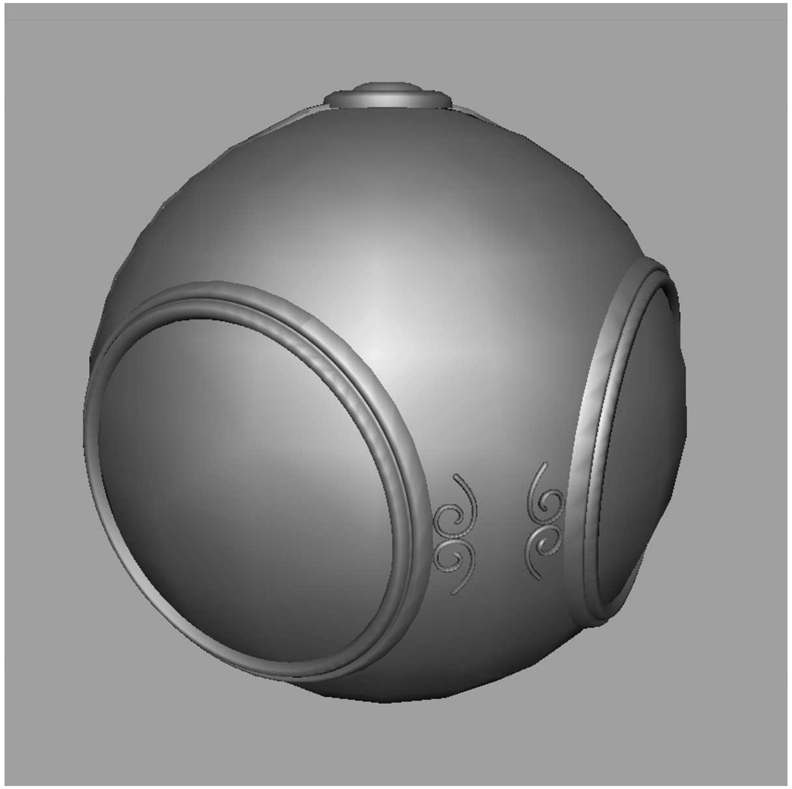
Modeling an Orange

- Consider the problem of modeling an orange (the fruit)
- Start with an orange-colored sphere
  - Too simple
- Replace sphere with a more complex shape
  - Does not capture surface characteristics (small dimples)
  - Takes too many polygons to model all the dimples

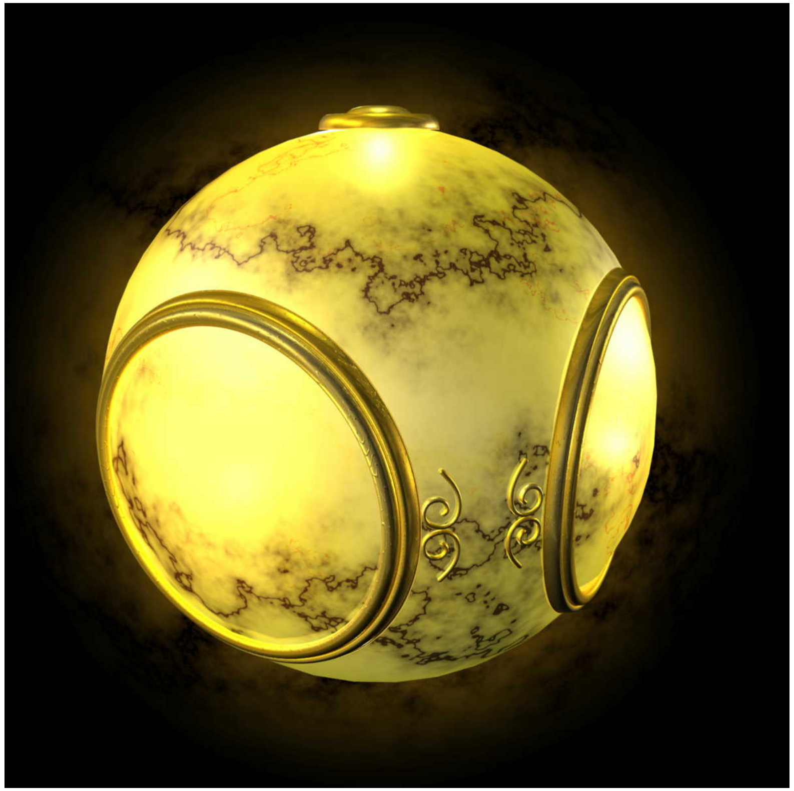
Modeling an Orange (2)

- Take a picture of a real orange, scan it, and “paste” onto simple geometric model
  - This process is known as texture mapping
- Still might not be sufficient because resulting surface will be smooth
  - Need to change local shape
  - Bump mapping

Texture Mapping



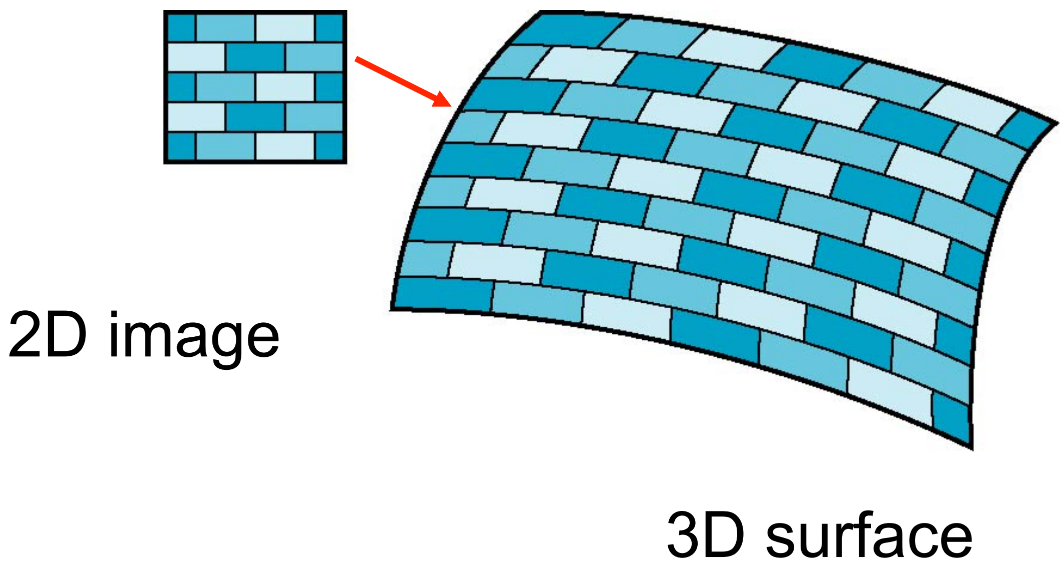
geometric model



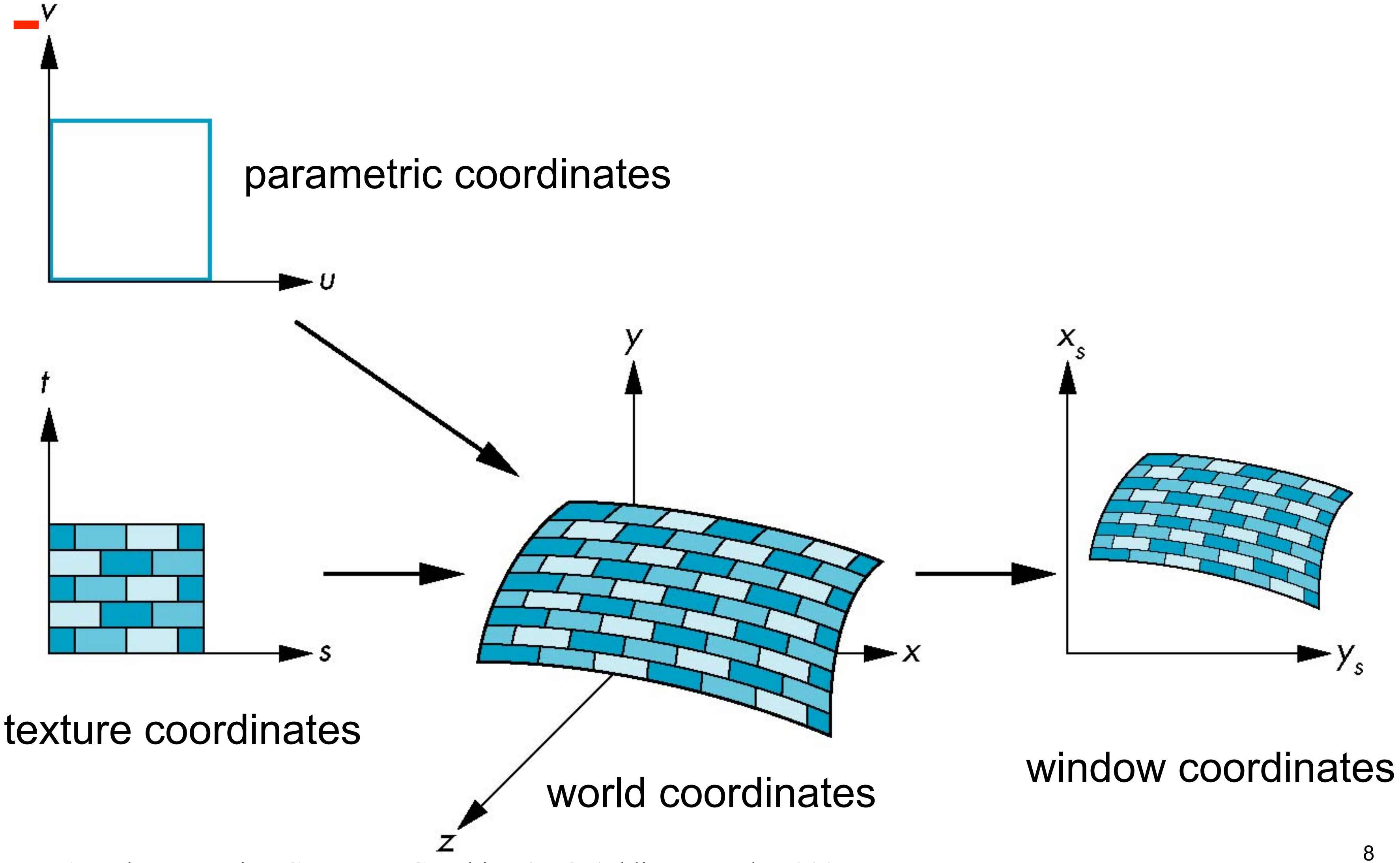
texture mapped

Is it simple?

- Although the idea is simple---map an image to a surface---there are 3 or 4 coordinate systems involved

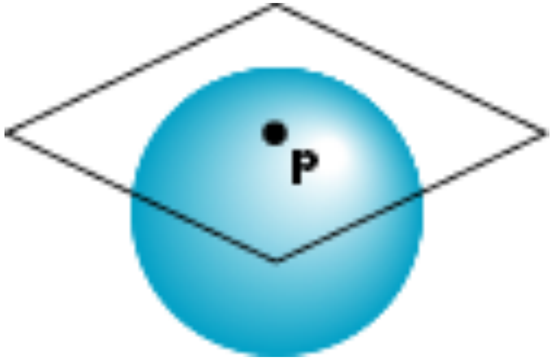


Texture Mapping



Parametric Form

- For sphere



$$x=x(u,v)=\cos u \sin v$$
$$y=y(u,v)=\cos u \cos v$$
$$z= z(u,v)=\sin u$$

- Tangent plane determined by vectors

$$\partial \mathbf{p}/\partial u = [\partial x/\partial u, \partial y/\partial u, \partial z/\partial u]T$$
$$\partial \mathbf{p}/\partial v = [\partial x/\partial v, \partial y/\partial v, \partial z/\partial v]T$$

- Normal given by cross product

$$\mathbf{n} = \partial \mathbf{p}/\partial u \times \partial \mathbf{p}/\partial v$$

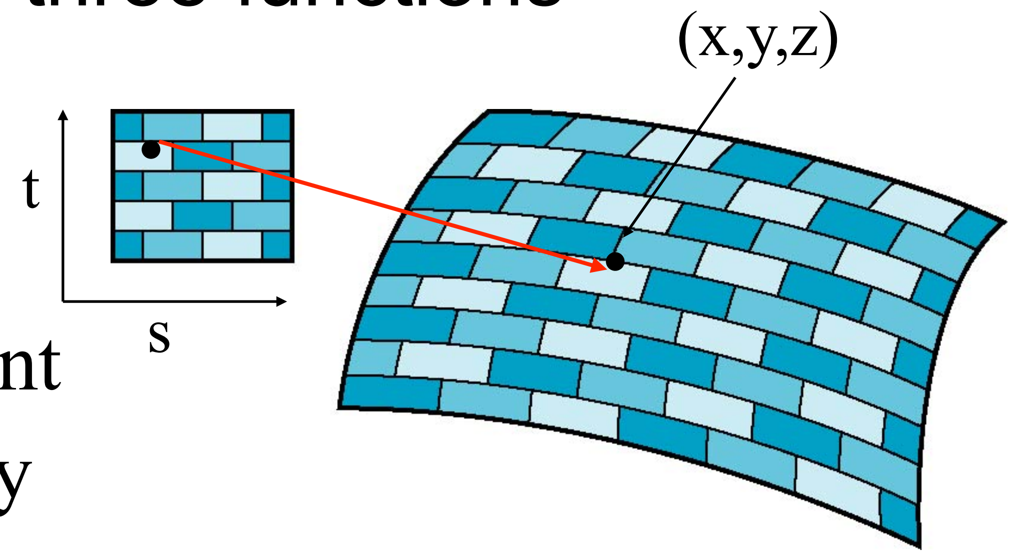
Mapping Functions

- 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

Backward Mapping

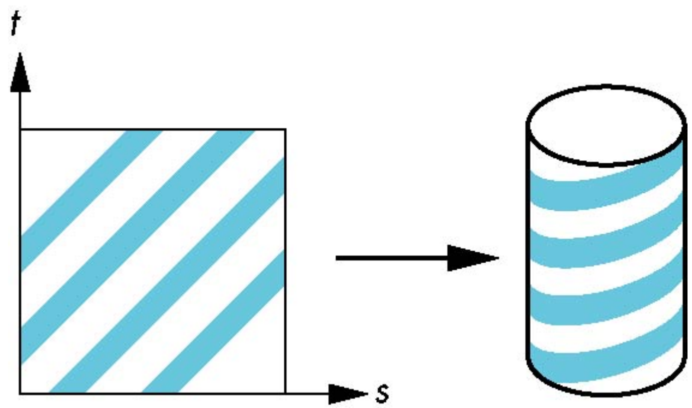
- We really want to go backwards
  - Given a pixel, we want to know to which point on an object it corresponds
  - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form

  $s = s(x,y,z)$

  $t = t(x,y,z)$
- Such functions are difficult to find in general

Two-part mapping

- One solution to the mapping problem is to first map the texture to a simple intermediate surface
- Example: map to cylinder



Cylindrical Mapping

parametric cylinder

$$x = r \cos 2\pi\, u$$
$$y = r \sin 2\pi u$$
$$z = v/h$$

maps rectangle in u,v space to cylinder of radius r and height h in world coordinates

$$s = u$$
$$t = v$$

maps from texture space

Spherical Map

We can use a parametric sphere

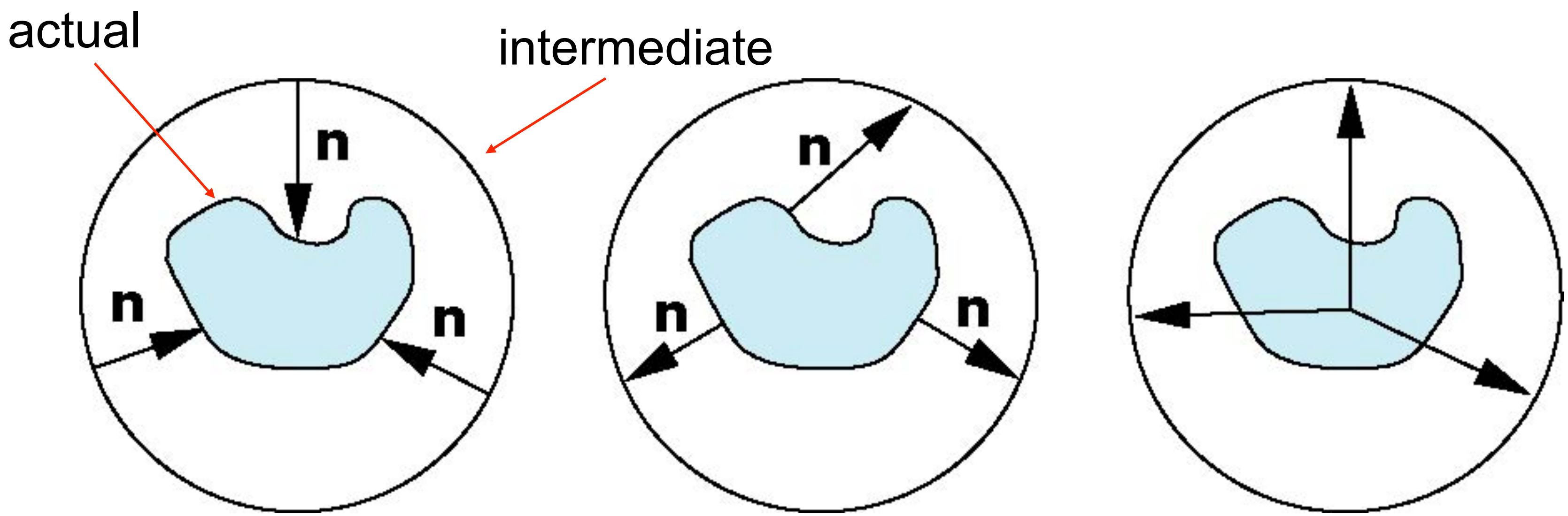
$$x = r \cos 2\pi u$$
$$y = r \sin 2\pi u \cos 2\pi v$$
$$z = r \sin 2\pi u \sin 2\pi v$$

in a similar manner to the cylinder but have to decide where to put the distortion

Spheres are used in environmental maps

Second Mapping

- Map from intermediate object to actual object
  - Normals from intermediate to actual
  - Normals from actual to intermediate
  - Vectors from center of intermediate



OpenGL Texture Mapping

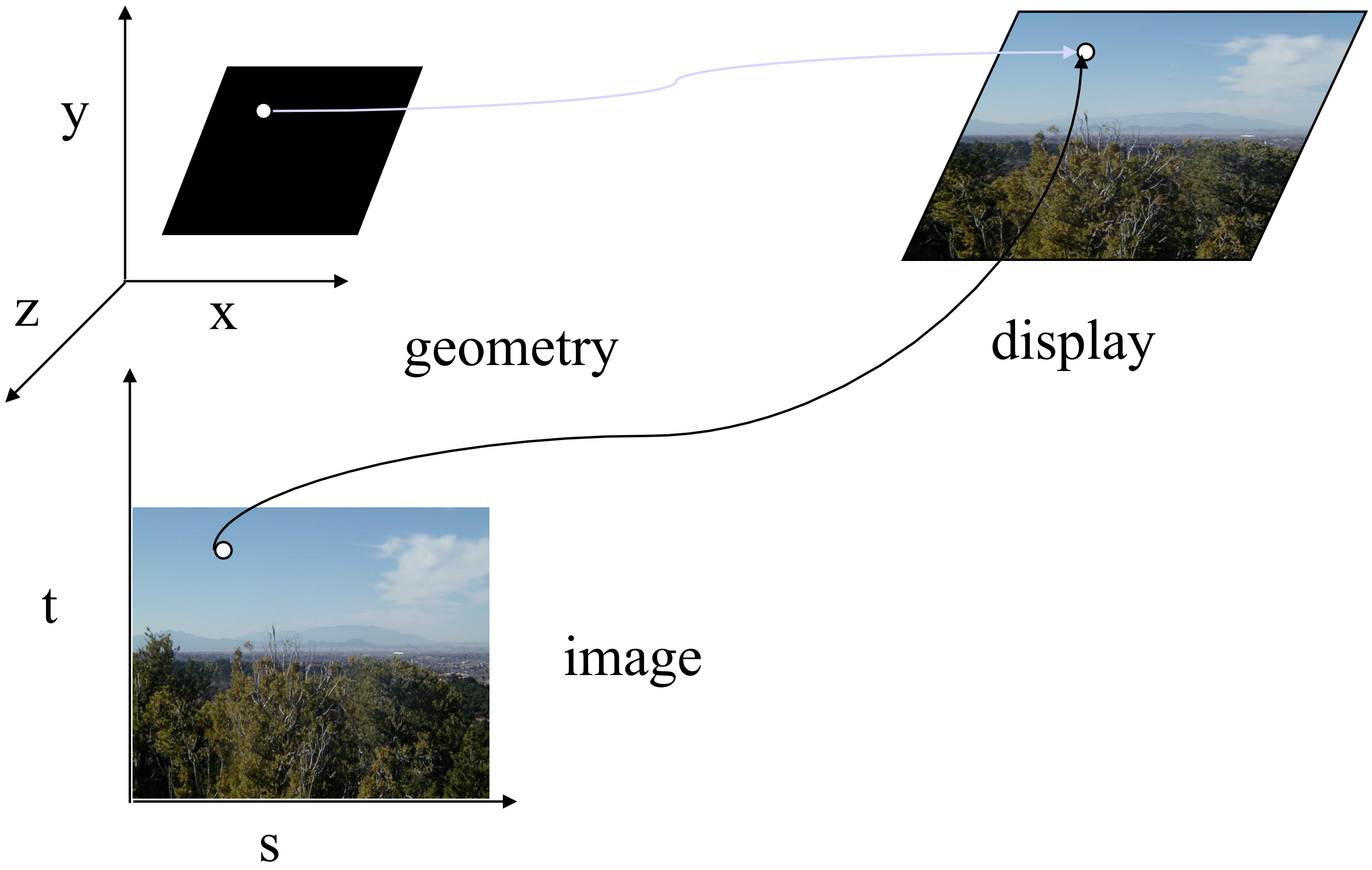
Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico

Basic Strategy

Three steps to applying a texture

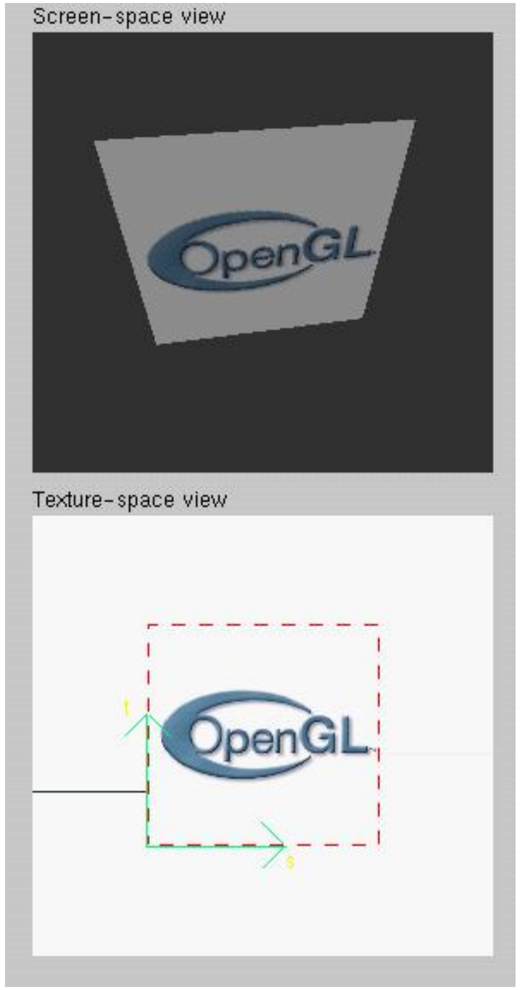
1. specify the texture
   - read or generate image
   - assign to texture
   - enable texturing
2. assign texture coordinates to vertices
   - Proper mapping function is left to application
3. specify texture parameters
   - wrapping, filtering

Texture Mapping



Texture Example

- The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective



Specifying a 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
  - Scanned image
  - Generate by application code
- Enable texture mapping
  - **`glEnable(GL_TEXTURE_2D)`**
  - OpenGL supports 1-4 dimensional texture maps

The University of New Mexico

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

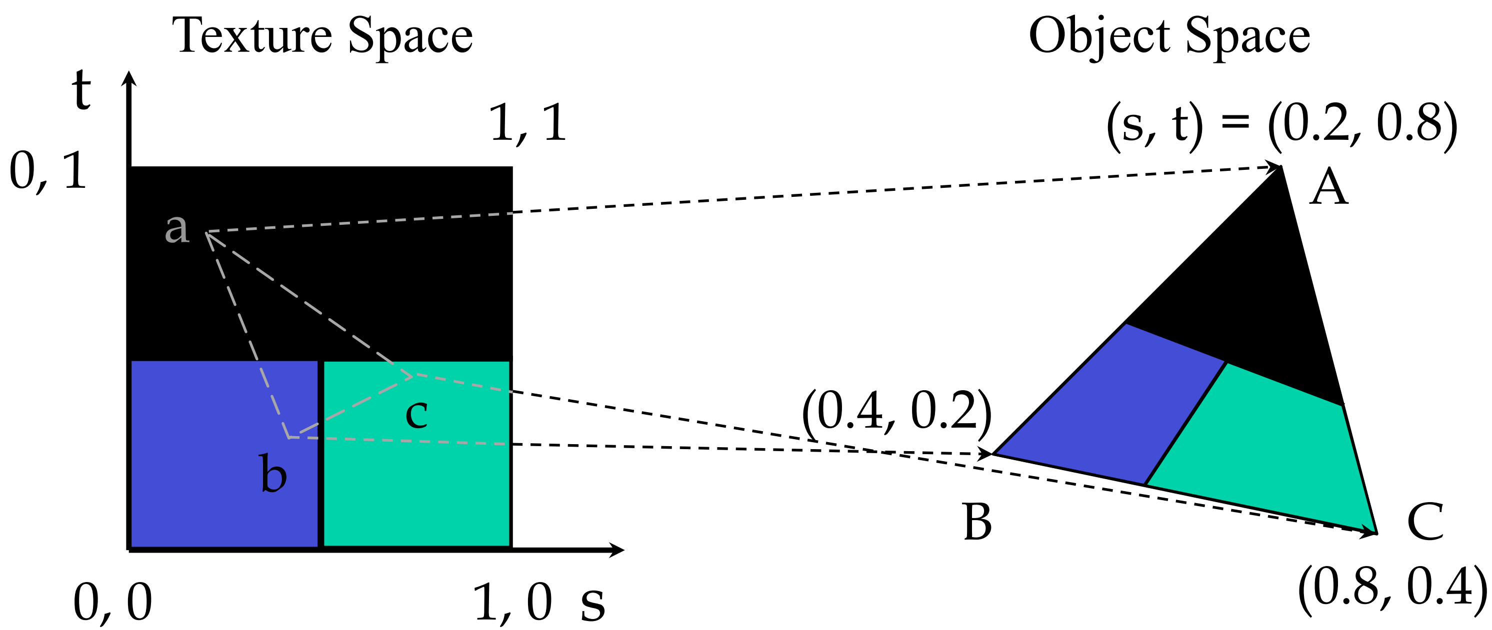
```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0,
 GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

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); //if no shading used
 glNormal3f(u0, v0, w0); // if shading used
 glTexCoord2f(s0, t0);
 glVertex3f(x0, y0, z0);
 glColor3f(r1, g1, b1);
 glNormal3f(u1, v1, w1);
 glTexCoord2f(s1, t1);
 glVertex3f(x1, y1, z1);
      .
      .
glEnd();
```

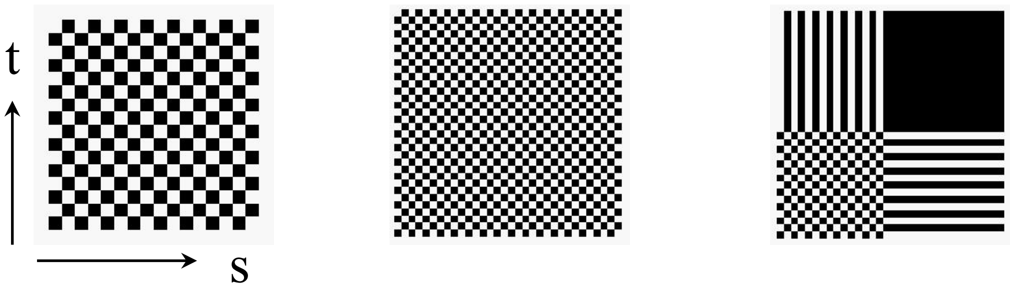
Note that we can use vertex arrays to increase efficiency

Wrapping Mode

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 )
```



texture

GL_REPEAT wrapping

GL_CLAMP wrapping