CS 480/680
INTERACTIVE COMPUTER GRAPHICS

Texture Mapping and NURBS

Week 7

David Breen
Department of Computer Science
Drexel University

Based on material from Ed Angel, University of New Mexico

Objectives

- Introduce Mapping Methods
  - Texture Mapping
  - Environmental 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 100 million polygons per second, that number is insufficient for many phenomena
  - Clouds
  - Grass
  - Terrain
  - Skin
  - Bark
  - Scales
  - Marble
  - Fabric

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

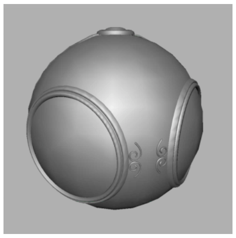
Modeling an Orange (2)

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

Three Types of Mapping

- Texture Mapping
  - Uses images to fill inside of polygons
- Environmental (reflection mapping)
  - Uses a picture of the environment for texture maps
  - Allows simulation of highly specular surfaces
- Bump mapping
  - Emulates altering normal vectors during the rendering process

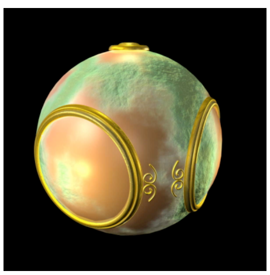
Texture Mapping



geometric model texture mapped

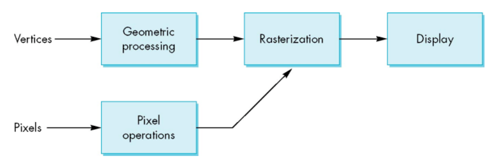
Environment Mapping

Bump Mapping



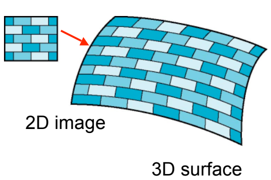
Where does mapping take place?

- Mapping techniques are implemented at the end of the rendering pipeline
  - Very efficient because few polygons pass down the geometric pipeline



Is it simple?

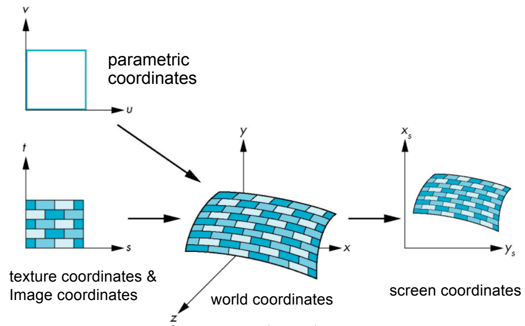
- Although the idea is simple---map an image to a surface---there are 3 or 4 coordinate systems involved



Coordinate Systems

- Parametric coordinates
  - May be used to model curved surfaces
- Texture coordinates
  - Used to identify points in the image to be mapped
- World Coordinates
  - Conceptually, where the mapping takes place
- Screen Coordinates
  - Where the final image is really produced

Texture Mapping

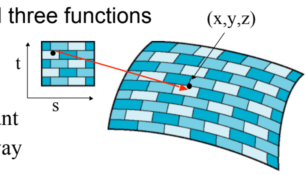


Mapping Functions

- 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



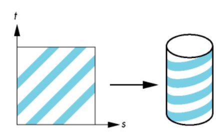
Backward Mapping

- 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

$x = r \cos(2\pi u)$
$y = r \sin(2\pi u)$
$z = v \cdot h$

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

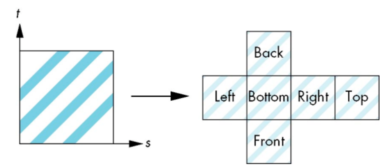
$x = r \cos(2\pi u)$
$y = r \sin(2\pi u) \cos(2\pi v)$
$z = r \sin(2\pi u) \sin(2\pi v)$

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

Spheres are used in environmental maps

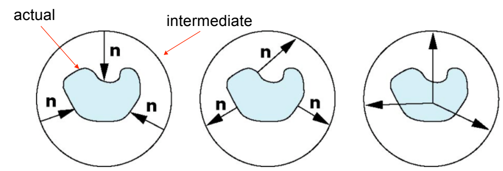
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environmental maps



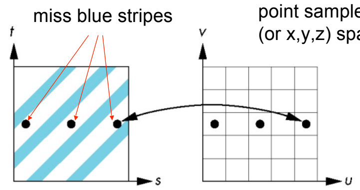
Second Mapping

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



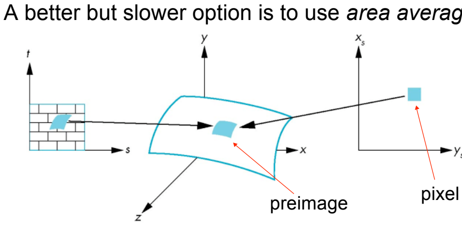
Aliasing

- Point sampling of the texture can lead to aliasing errors



Area Averaging

A better but slower option is to use *area averaging*



Note that *preimage* of pixel is curved

OpenGL Texture Mapping

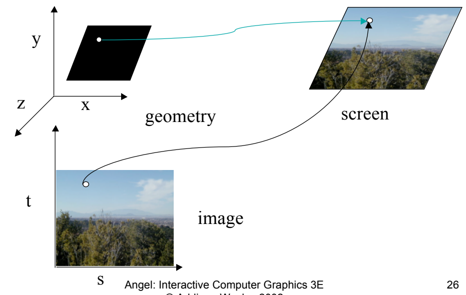
Objectives

- Introduce the OpenGL texture functions and options

Basic Stragegy

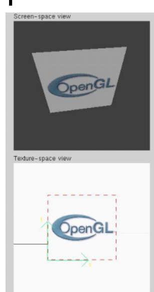
- Three steps to applying a texture
  1. specify the texture
     - read or generate image
     - assign to texture
     - enable texturing
  2. assign texture coordinates to vertices
     - Proper mapping function is left to application
  3. specify texture parameters
     - wrapping, filtering

Texture Mapping



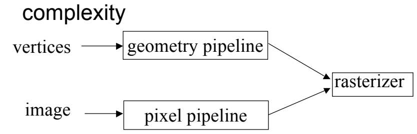
Texture Example

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



Texture Mapping and the OpenGL Pipeline

- Images and geometry flow through separate pipelines that join at the rasterizer
  - “complex” textures do not affect geometric complexity



Specify Texture Image

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

  `Glubyte my_texels[512][512][3];`
- 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, iformat,
      w, h, border, format, type, texels );
```

`target:` type of texture, e.g. `GL_TEXTURE_2D`
`level:` used for mipmapping (discussed later)
`iformat :` internal format of texels
`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

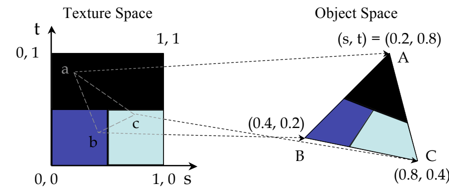
```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 512, 512, 0,
   GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

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 );`
  - `format` → `GL_RGB,GL_RGBA, GL_LUMINANCE,` etc.
  - `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

Transforming Textures

- Texture coordinates can be transformed
- `glMatrixMode( GL_TEXTURE );`
- Texture matrix can rotate, translate and scale textures

Interpolation

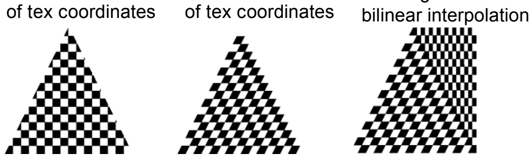
OpenGL uses bilinear interpolation to find proper texels from specified texture coordinates

Can be distortions

good selection of tex coordinates

poor selection of tex coordinates

texture stretched over trapezoid showing effects of bilinear interpolation



Texture Parameters

- OpenGL has a variety of parameters that determine how texture is applied
  - Wrapping parameters determine what happens if s and t are outside the (0,1) range
  - Filter modes allow us to use area averaging instead of point samples
  - Mipmapping allows us to use textures at multiple resolutions
  - Environment parameters determine how texture mapping interacts with shading

Wrapping Mode

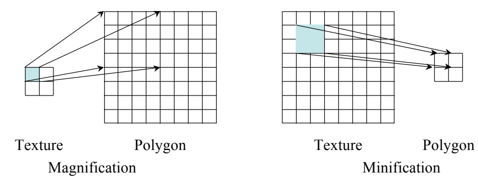
Clamping: if s,t > 1 use 1, if s,t <0 use 0
Repeating: use s,t modulo 1

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

Magnification and Minification

Many pixels cover one texel (*magnification*) or
one pixel covers many texels (*minification*)

Can use point sampling (nearest texel) or linear filtering
(2 x 2 filter) to obtain texture values



Filter Modes

Modes determined by
- `glTexParameteri( target, type, mode )`

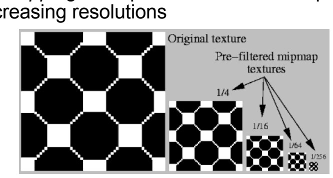
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MAG_FILTER,
          GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
          GL_LINEAR);
```

Note that linear filtering requires a border of an
extra texel for filtering at edges (border = 1)

Mipmapped Textures

- *Mipmapping* uses prefiltered texture maps of decreasing resolutions



- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition
  `glTexImage2D( GL_TEXTURE_2D, level, … )`
- GLU mipmap builder routine will build all the textures from a given image - `gluBuild2DMipmaps( … )`

Mipmapped Textures

```
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, 64, 64,
  GL_RGB, GL_UNSIGNED_BYTE, image0)
glTexImage2D( GL_TEXTURE_2D, 1, GL_RGB, 32, 32,
  GL_RGB, GL_UNSIGNED_BYTE, image1)
glTexImage2D( GL_TEXTURE_2D, 2, GL_RGB, 16, 6,
  GL_RGB, GL_UNSIGNED_BYTE, image1)
…

gluBuild2DMipmaps(GLenum target, GLint
  internalFormat, GLsizei width, GLsizei height,
  GLenum format, GLenum type, const void *data )

gluBuild2DMipmaps( GL_TEXTURE_2D, GL_RGB, width,
  height, GL_RGB, GL_UNSIGNED_BYTE, data );
```


Mipmap Filter Modes

Modes determined by
- `glTexParameteri( target, type, mode )`

How to filter in image (GL_*).
How to filter between images (MIPMAP_*).

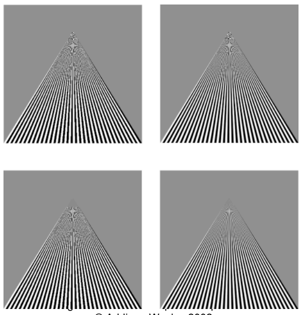
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MAG_FILTER,
          GL_NEAREST_MIPMAP_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
          GL_LINEAR_MIPMAP_NEAREST);

GL_NEAREST_MIPMAP_LINEAR

GL_LINEAR_MIPMAP_LINEAR
```

Example



Texture Functions

- Controls how texture is applied
  - `glTexEnv{fi}[v]( GL_TEXTURE_ENV, prop, param )`
- `GL_TEXTURE_ENV_MODE` modes
  - `GL_MODULATE`: modulates with computed shade (multiply colors)
  - `GL_BLEND`: blends with an environmental color
  - `GL_REPLACE`: use only texture color
- `glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);`
- Set blend color with `GL_TEXTURE_ENV_COLOR`
- `glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, color);`

Perspective Correction Hint

- Texture coordinate and color interpolation
  - either linearly in screen space
  - or using depth/perspective values (slower)
- Noticeable for polygons "on edge"
  - `glHint( GL_PERSPECTIVE_CORRECTION_HINT, hint )`

  where `hint` is one of
    - `GL_DONT_CARE`
    - `GL_NICEST`
    - `GL_FASTEST`

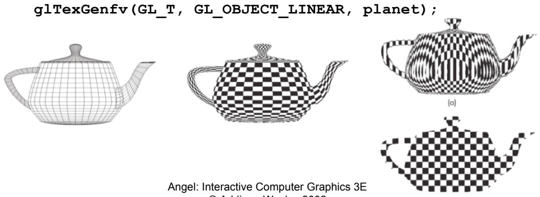
Generating Texture Coordinates

- OpenGL can generate texture coordinates automatically

  `glTexGen{ifd}[v]()`
- specify a plane
  - generate texture coordinates based upon distance from the plane
  - $s = a_s x + b_s y + c_s z + d_s w$
  - $t = a_t x + b_t y + c_t z + d_t w$
- generation modes
  - `GL_OBJECT_LINEAR`
  - `GL_EYE_LINEAR`
  - `GL_SPHERE_MAP` (used for environmental maps)

Generating Texture Coordinates

```
/* s = x/2 + 1/2 */
Glfloat planes[] = {0.5, 0.0, 0.0, 0.5};
/* t = y/2 + 1/2 */
Glfloat planet[] = {0.0, 0.5, 0.0, 0.5};
glTexGeni(GL_S, GL_TEXTURE_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_LINEAR, planes);
glTexGenfv(GL_T, GL_OBJECT_LINEAR, planet);
```



Texture Objects

- Texture is part of the OpenGL state
  - If we have different textures for different objects, OpenGL will be moving large amounts data from processor memory to texture memory
- Recent versions of OpenGL have *texture objects*
  - one image per texture object
  - Texture memory can hold multiple texture objects

```
glGenTextures()
glBindTexture()
glDeleteTextures()
```

Other Texture Features

- Environmental Maps
  - Start with image of environment through a wide angle lens
    - Can be either a real scanned image or an image created in OpenGL
  - Use this texture to generate a spherical map
  - Use automatic texture coordinate generation
- Multitexturing
  - Apply a sequence of textures through cascaded texture units

Applying Textures Summary

1. specify textures in texture objects
2. set texture filter
3. set texture function
4. set texture wrap mode
5. set optional perspective correction hint
6. bind texture object
7. enable texturing
8. supply texture coordinates for vertex
   - coordinates can also be generated

NURBS in OpenGL

Go to CS430 B-Spline Lecture

Data Structures

```
// c object
GLUnurbsObj * nurb ;
// control points for the nurbs
float cpoints [ 4 ][ 4 ][ 3 ];
// ((0,0), (0,1), (1,0), (1,1))
float tcoords [ 2 ][ 2 ][ 2 ];
// knots.  #cp's = #knots - order for each param
float knots[ 8 ] = { 0.0, 0.0, 0.0, 0.0,
                     1.0,1.0, 1.0, 1.0 };
float tknots[ 4 ] = { 0.0, 0.0, 1.0, 1.0 };
```


Create the Object

```
// set up the object to draw the NURB
// create the NURB
nurb = gluNewNurbsRenderer();
// set a tolerance (drawing nurbs is not exact)
gluNurbsProperty( nurb ,
   GLU_SAMPLING_TOLERANCE, 25.0 );
// how to draw the nurb (fill/patch
   outline/points/lines)
gluNurbsProperty( nurb,
   GLU_DISPLAY_MODE, GLU_FILL );
```

Enabling NURBS

```
// for depth
glEnable( GL _DEPTH_TEST );
// to automatically generate NURB normals
glEnable( GL _AUTO_NORMAL );
// normalize the normals
glEnable( GL _NORMALIZE );
// enable lighting to see
glEnable( GL _LIGHTING );
```


Drawing the NURB

```
// begin the surface
gluBeginSurface( nurb );
// draw the nurb
gluNurbsSurface( nurb, 4, tknots, 4, tknots, 2*2,
   2, &tcoords[0][0][0], 2, 2,
   GL_MAP2_TEXTURE_COORD_2 );
gluNurbsSurface( nurb , 8, knots, 8, knots, 4 *
   3, 3, &cpoints[0][0][0], 4, 4,
   GL_MAP2_VERTEX_3 );
// end the surface
gluEndSurface( nurb );
```


gluNurbsSurface

- void gluNurbsSurface(GLUnurbs *nurb, GLint sKnotCount, GLfloat* sKnots, GLint tKnotCount, GLfloat* tKnots, GLint sStride, GLint tStride, GLfloat* control, GLint sOrder, GLint tOrder, GLenum type)
- nurb - the NURBS object created with gluNewNurbsRenderer()
- sKnots - array of sKnotCount nondecreasing knot values in the parametric s direction
- tKnots - array of tKnotCount nondecreasing knot values in the parametric t direction
- sStride, tStride - offset (# of floating point values) between successive control points in the s & t direction in control
- control - array containing control points for the NURBS surface
- sOrder, tOrder - order of NURBS polynomial in s & t direction
- type - GL_MAP2_VERTEX_3, GL_MAP2_NORMAL, GL_MAP2_TEXTURE_COORD_2, etc.

```
gluNurbsSurface( nurb , 8, knots, 8, knots, 4 * 3, 3,
         &cpoints[0][0][0], 4, 4, GL_MAP2_VERTEX_3 );
```