Texture Mapping

CS 432 Interactive Computer Graphics
Prof. David E. Breen
Department of Computer Science

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

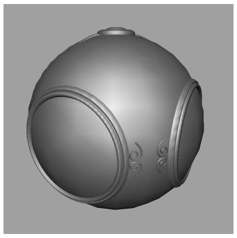
Modeling an Orange (2)

- 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

Three Types of Mapping

- Texture Mapping
  - Uses images to fill inside of polygons
- Environment (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

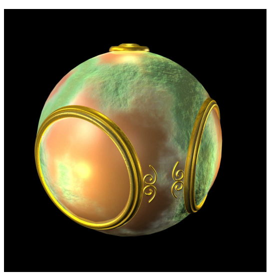


geometric model

texture mapped

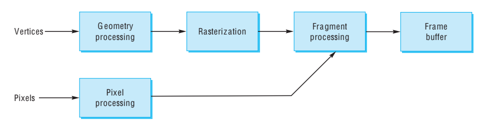
Environment Mapping

Bump Mapping



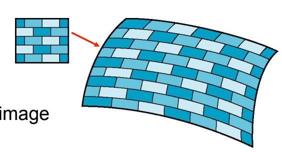
Where does mapping take place?

- Mapping techniques are implemented at the end of the rendering pipeline
  - Very efficient because few polygons make it past the clipper



Is it simple?

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



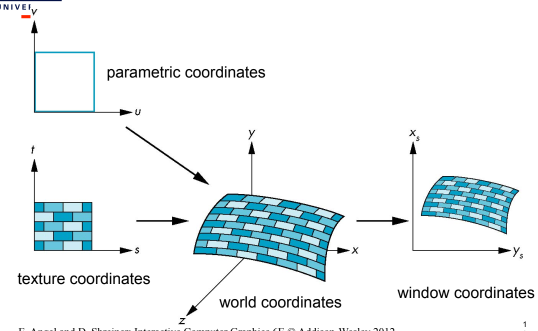
2D image

3D surface

Coordinate Systems

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

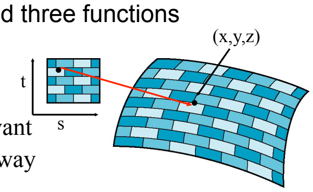
Texture Mapping



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



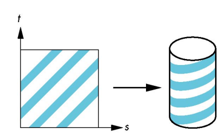
Backward Mapping

- 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

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

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

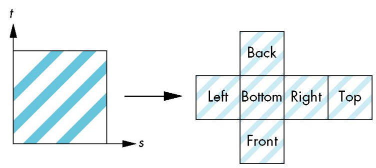
$x = r \cos 2\pi u$
$y = r \sin 2\pi u \cos 2\pi v$
$z = r \sin 2\pi u \sin 2\pi v$

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

Spheres are used in environmental maps

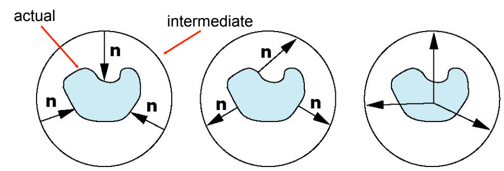
Box Mapping

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



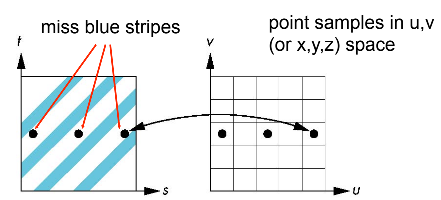
Second Mapping

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



Aliasing

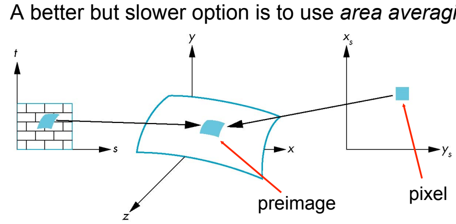
- Point sampling of the texture can lead to aliasing errors



point samples in texture space

Area Averaging

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



Note that *preimage* of pixel is curved

Go to ART Texture Mapping slides

OpenGL Texture Mapping

CS 432 Interactive Computer Graphics
Prof. David E. Breen
Department of Computer Science

Objectives

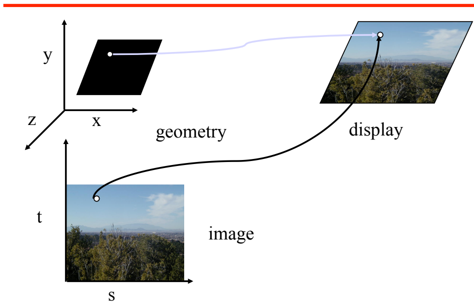
- Introduce the OpenGL texture functions and options

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

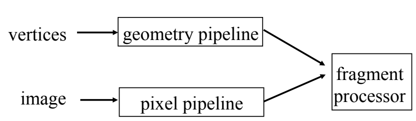


Texture Example

- 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 during fragment processing
  - "complex" textures do not affect geometric complexity



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

Define Image as a Texture

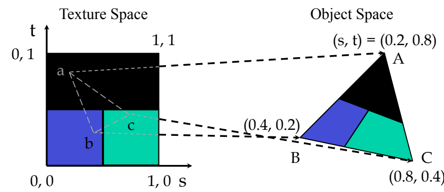
```
glTexImage2D( target, level, components,
    w, h, border, format, type, texels );
```

`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);
```


Mapping a Texture

- Application or shaders define texture coordinates
- Texture coordinates specified at each vertex



Typical Code

- Application sending texture coordinates

```
offset = 0;
GLuint vPosition = glGetAttribLocation( program,
  "vPosition" );
glEnableVertexAttribArray( vPosition );
glVertexAttribPointer( vPosition, 4, GL_FLOAT,
  GL_FALSE, 0,BUFFER_OFFSET(offset) );

offset += sizeof(points);
GLuint vTexCoord = glGetAttribLocation( program,
  "vTexCoord" );
glEnableVertexAttribArray( vTexCoord );
glVertexAttribPointer( vTexCoord, 2,GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(offset) );
```


Interpolation

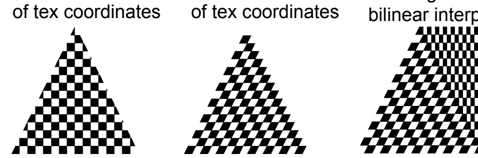
OpenGL uses interpolation to find proper texels from specified texture coordinates

Can be distortions

good selection of tex coordinates

poor selection of tex coordinates

texture stretched over trapezoid showing effects of bilinear interpolation



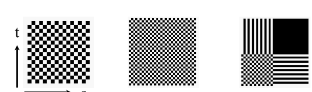
Texture Parameters

- 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
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 )
```

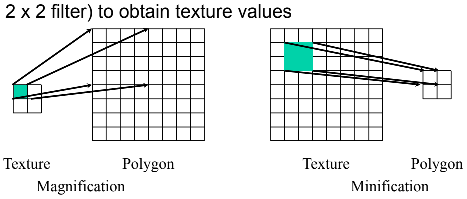


texture GL_REPEAT wrapping GL_CLAMP wrapping

Magnification and Minification

More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

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

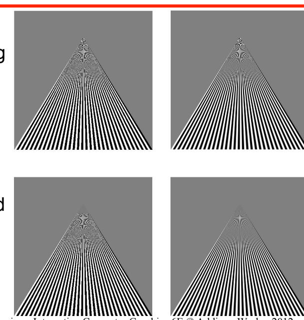
- *Mipmapping* allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition
  `glTexImage2D( GL_TEXTURE_*D, level, … )`
- Have OpenGL make your mipmap
  `glGenerateMipmap(target)`
- Mipmaps invoked by setting sampling

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


Example

point sampling

linear filtering



mipmapped point sampling

mipmapped linear filtering

Using Texture Objects

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

Other Texture Features

- Environment 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
  - Alternative is to use a cube map
- Multitexturing
  - Apply a sequence of textures through cascaded texture units

Checkerboard Texture

```
GLubyte image[64][64][3];

// Create a  64 x 64 checkerboard pattern
   for ( int i = 0; i < 64; i++ ) {
      for ( int j = 0; j < 64; j++ ) {
         GLubyte c = (((i & 0x8) == 0) ^ ((j & 0x8)  == 0)) * 255;
         image[i][j][0]  = c;
         image[i][j][1]  = c;
         image[i][j][2]  = c;
```


Adding Texture Coordinates

```
void quad( int a, int b, int c, int d )
{
   quad_colors[Index] = colors[a];
   points[Index] = vertices[a];
   tex_coords[Index] = vec2( 0.0, 0.0 );
   index++;
  quad_colors[Index] = colors[a];
   points[Index] = vertices[b];
   tex_coords[Index] = vec2( 0.0, 1.0 );
   Index++;

// other vertices
}
```


Texture Object

```
GLuint textures[1];
glGenTextures( 1, textures );

glBindTexture( GL_TEXTURE_2D, textures[0] );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, TextureSize,
  TextureSize, 0, GL_RGB, GL_UNSIGNED_BYTE, image );
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 );
glActiveTexture( GL_TEXTURE0 );
```


Linking with Shaders

```
GLuint vTexCoord = glGetAttribLocation( program, "vTexCoord" );
glEnableVertexAttribArray( vTexCoord );
glVertexAttribPointer( vTexCoord, 2, GL_FLOAT, GL_FALSE, 0,
               BUFFER_OFFSET(offset) );

// Set the value of the fragment shader texture sampler variable
//   ("texture") to the the appropriate texture unit. In this case,
//   zero, for GL_TEXTURE0 which was previously set by calling
//   glActiveTexture().
glUniform1i( glGetUniformLocation(program, "texture"), 0 );
```


Vertex Shader

- Usually vertex shader will output texture coordinates to be rasterized
- Must do all other standard tasks too
  - Compute vertex position
  - Compute vertex color if needed

```
in vec4 vPosition; //vertex position in object coordinates
in vec4 vColor;  //vertex color from application
in vec2 vTexCoord; //texture coordinate from application

out vec4 color; //output color to be interpolated
out vec2 texCoord; //output tex coordinate to be interpolated
```


Applying Textures

- Textures are applied during fragments shading by a **sampler**
- Samplers return a texture color from a texture object

```
in vec4 color;  //color from rasterizer
in vec2 texCoord; //texure coordinate from rasterizer
uniform sampler2D texture; //texture object id from application

void main()  {
   gl_FragColor = color * texture2D( texture, texCoord );
}
```

Using Textures

- Texture value may be used in ANY of the components of the shading formula
- For example
  - Diffuse color
  - Specular color
  - Ambient color
  - Shininess
  - Normals
  - Alpha
- Or as a decal, or mask or blended in