TEXTURE MAPPING

Texture Mapping

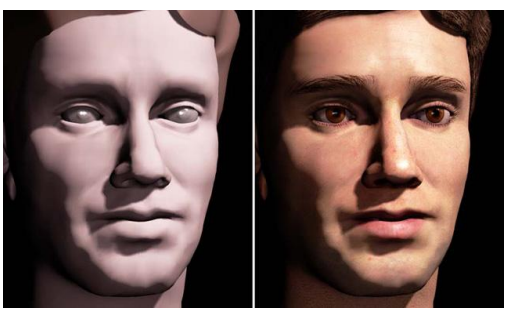
- We have our Phong-shaded objects, or maybe a raytraced image. It still doesn't look real.
- A major problem is surface detail. Now What?
  - Add more polygons or primitives. (slow!)
  - Add detail with a texture!

Texture Mapping: An Example

Images Jeremy Birn

Image: Matt Lewis

Texture Mapping: An Example



Images Jeremy Birn

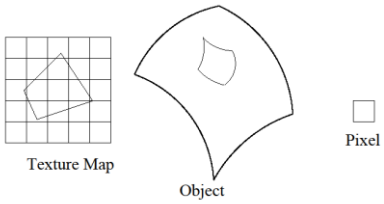
Texture Mapping: An Example

Image: Jeremy Birn

Texture Mapping: An Example

Image: Jeremy Birn

Texture Mapping

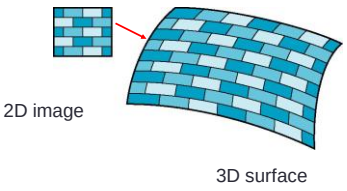


The Basic Algorithm

- Must map a 2D image to a 3D model.
- Use texture coordinates.
  - 2D coordinates (s,t), maps to 2D texture location.
  - Usually (s,t) are over [0,1]
- Each vertex in polygon gets a texture coordinate.
- Or if a parametric model, use the (u,v) of the model.
- Use linear interpolation to associate texels to a pixel

Is it simple?

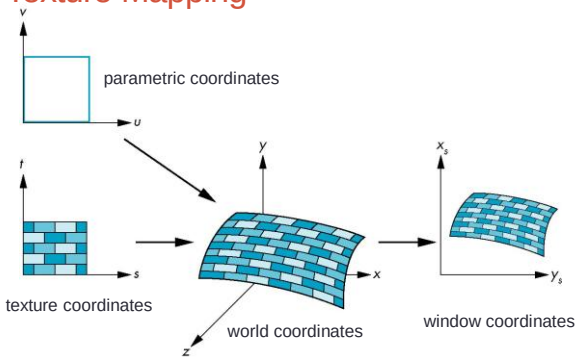
- Although the idea is simple---map an image to a surface---there are 3 or 4 coordinate systems involved



Coordinate Systems

- Parametric coordinates
  - May be used to model 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

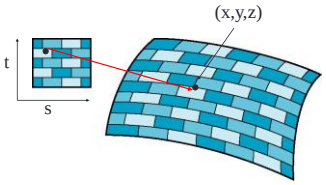
Texture Mapping



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

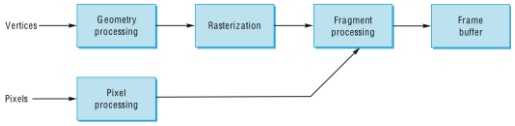
  s = s(x,y,z)
  t = t(x,y,z)
- Such functions are difficult to find in general

How Do Pixels and Texels Relate?

- Magnification
  - One pixel covers only a small part of a texel
- Minification
  - One pixel covers all or part of many texels

When Do We Interpolate

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



- Perspective projection is a non-linear transformation.
- It can make a big difference if we interpolate before or after.
- For proper foreshortening we should interpolate before perspective projection.

Interpolation

- Linear Interpolation
  - Take the closest texel and use its RGB value for shading.
  - Has problems. Especially at a distance, (Moire patterns)
- Take the average of the closest pixels.
- Bilinear uses the four closest points
- Trilinear – two bilinear filters
- Anistropic – Elliptical mask for filter, expensive

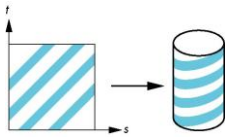
How Do We Create the Mapping?

- Will a square picture map onto a sphere without distortion?
- How do you map a texture onto a wall?
- How about a cow?
- How about a car?

Two-Part Mapping

- Map to intermediate 3D object first.
- Then map 3D object to the final model.
- Typical intermediate objects: Cylinder, Cube, Sphere.

- Example: map to cylinder

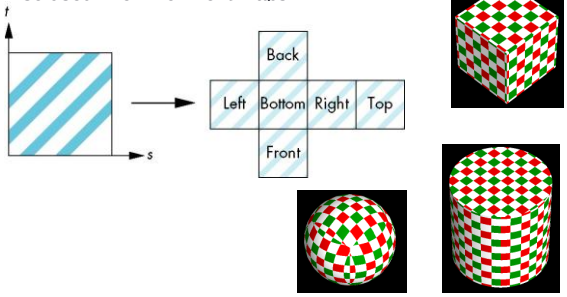


Two-Part Mapping: Planar

- Pick a plane, say xy plane of size (w,h)
  - f(p) = (px/w, py/h)

Two-Part Mapping: Cube

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



Two-Part Mapping: Cylindrical

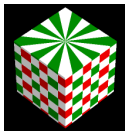
parametric cylinder

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

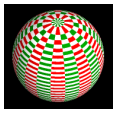
maps rectangle in u,v space to cylinder of radius r and height h in world coordinates

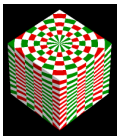
$$s = u$$
$$t = v$$

maps from texture space



Two-Part Mapping: Rectangular Cylindrical



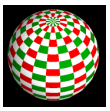


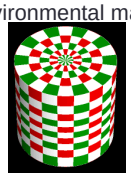
Two-Part Mapping: Spherical

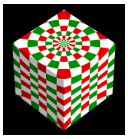
in a similar manner to the cylinder but have to decide where to put the distortion

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

Spheres are used in environmental maps

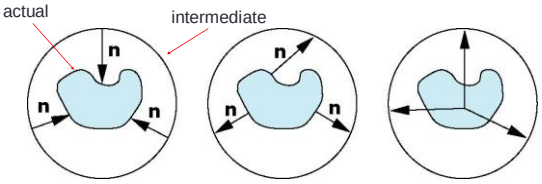






Mapping to Intermediate Object?

- We are just determining the texture coordinate.
- Use the normal from intermediate object.
- Use the normal from the object.
- Use the centroid of the object.

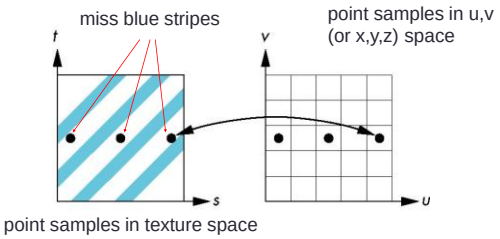


How do We Apply the Texture?

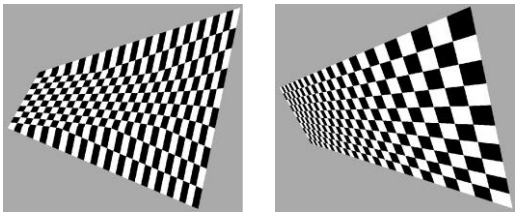
- Decal (replace).
- Modulate (multiply).
- Blend (interpolate).
- Wrap (repeat).
- No wrap (no repeat).

Aliasing

- Point sampling of the texture can lead to aliasing errors

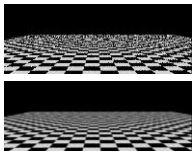


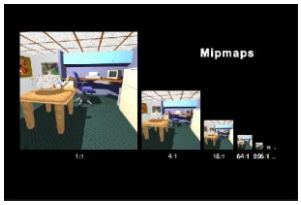
Problems: Linear vs Trilinear



Antialiasing: Mipmapping

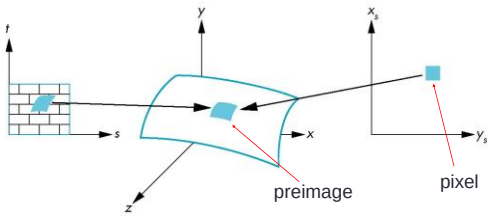
- Pre-calculate the texture at various resolutions (powers of 2). Use the appropriate map based on the size of the polygon on the screen.
- Fast and somewhat accurate.
- How much extra storage does it take?





Area Averaging

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



Note that *preimage* of pixel is curved

Multiple Maps

- Can fix stretching at pol by using two maps
- Blend between them

Image: Jerry Sandefur

Texture Mapping: Not Just Color

- Can apply to any shading attribute
- Color
- Specular Highlights
- Reflections
- Diffuse Reflections
- Normals
- Refraction
- Transparency

Texture Mapping Issues

- Tiling textures can introduce seems
  - A tileable texture will fix this
- If use an image, will have built in lighting that doesn't match screen
- The mapping causes distortions
- Suffers from level of detail (minification, magnification)

OPENGL TEXTURE MAPPING

Objectives

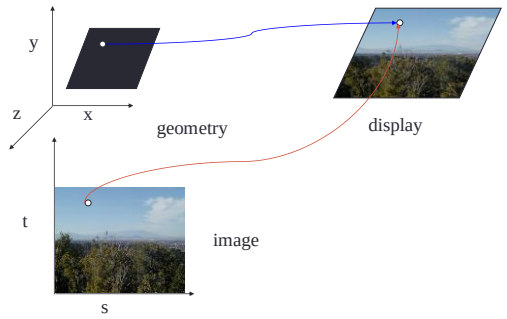
- Introduce the OpenGL texture functions and options

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

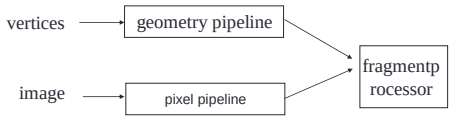


Texture Example

- 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 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

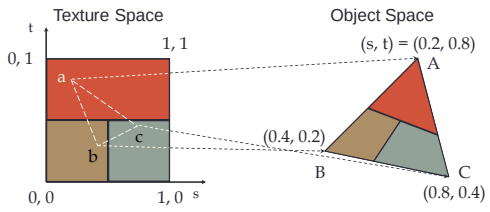
```
glTexImage2D( target, level, components,
    w, h, border, format, type, texels );
```

`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

- Based on parametric texture coordinates
- `glTexCoord*()` specified at each vertex



Typical Code

```
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

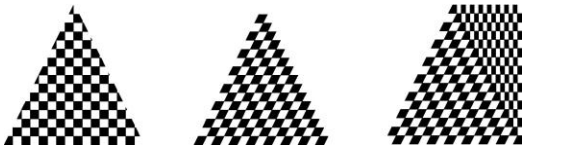
OpenGL uses interpolation to find proper texels from specified texture coordinates

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

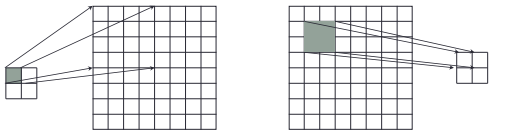
GL_REPEAT wrapping

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



Texture Polygon

Magnification

Texture Polygon

Minification

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);
```

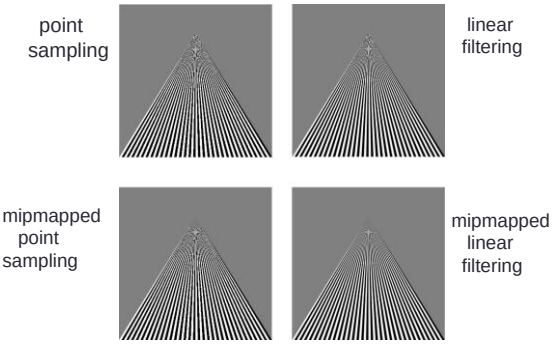
Note that linear filtering requires a border of an extra texel for filtering at edges (border = 1)

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, … )
```


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
  - `GL_BLEND:` blends with an environmental color
  - `GL_REPLACE:` use only texture color
  - `GL(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);`
- Set blend color with `GL_TEXTURE_ENV_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
- generation modes
  - `GL_OBJECT_LINEAR`
  - `GL_EYE_LINEAR`
  - `GL_SPHERE_MAP` (used for environmental maps)

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

Applying Textures II

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

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
  - Use automatic texture coordinate generation
- Multitexturing
  - Apply a sequence of textures through cascaded texture units