Texture Mapping

Introduction and Basic OpenGL functionality

Why Texture Map?

- How can we model this scene with polygons?
  - Lots of detail means lots of polygons to render
  - 100’s, 1000’s, Millions of polygons!
  - Can be difficult to model fine features

Review: Why Texture Map? (2)

- What if we render just one polygon with a picture of a brick wall mapped to it?

- The graphics hardware is faster at texture mapping than processing many polygons

Textures Provide Realism

- Spatially-varying modification of surface appearance at the pixel level
- Characteristics
  - Color
  - Shininess
  - Transparency
  - Bumpiness
  - Etc.
Our Goal

Texture Mapping Overview

Texture mapping: Steps

- **Creation**: Where does the texture image come from, i.e. sources?

- **Parameterization**: Mapping from 3-D shape locations \((x, y, z)\) to 2-D texture image coordinates \((s, t)\)

- **Rasterization (Sampling and Interpolation)**: What do we draw at each pixel?

Texturing: Creation

- **Reproductions**
  - Photographs
  - Handpainted
  - Download from the web

- **Directly-computed functions**
  - e.g., lightmaps

- **Procedurally-built**
  - Synthesize with randomness, pattern-generating rules, etc.
  - Remember our checkerboard procedure?
Enable (Disable) Textures

- Enable texture –
  `glEnable(GL_TEXTURE_2D)`
- Disable texture –
  `glDisable(GL_TEXTURE_2D)`

Remember to disable texture mapping when you draw non-textured polygons

**Texturing: Parameterization (Simple)**

- Assign texture coordinates to the polygon vertices (in **object space**)
- We did this in Lab1.
- Simple Case
  - Apply a 2D texture onto a quadrilateral

![Texture Mapping Diagram](texture_mapping.png)

**Texturing: Parameterization (Object Mesh)**

- How do we assign texture coordinates to objects?
  - Problem: Map from 3D to 2D
  - Idea: Map \((x, y, z)\) to an intermediate space \((u, v)\)
- **Projector** function to obtain object surface coordinates \((u, v)\)
- **Corresponder** function to find texel coordinates \((s, t)\)
- Filter texel at \((s, t)\)
- Modify pixel \((i, j)\)

**Projector Functions**

- How do we map the texture onto a arbitrary (complex) object?
  - Construct a mapping between the 3-D point to an intermediate surface
  - Why?
    - The intermediate surface is simple \(\Rightarrow\) we know its characteristics
    - Still a 3D surface, but easier to map to texture space (2D)
    - Easy to parameterize the intermediate surface in 2D, i.e. \((u, v)\) space
- Idea: Project each object point to the intermediate surface with a parallel or perspective projection
  - The focal point is usually placed inside the object
    - Plane
    - Cylinder
    - Sphere
    - Cube
    - Mesh: piece-wise planar

![Planar Projector Diagram](planar_projector.png)
Planar Projector

Orthographic projection onto $XY$ plane:
\[ u = x, \quad v = y \]

...onto $YZ$ plane ...onto $XZ$ plane

courtesy of R. Wolfe

Cylindrical Projector

- Convert rectangular coordinates $(x, y, z)$ to cylindrical $(r, \mu, h)$, use only $(h, \mu)$ to index texture image

courtesy of R. Wolfe

Spherical Projector

- Convert rectangular coordinates $(x, y, z)$ to spherical $(\theta, \phi)$

courtesy of R. Wolfe

Surface Patches

- A polygon or mesh of polygons defining a surface
  - Map four corners of a quad to $(u, v)$ values
Parametric Surfaces

A parameterized surface patch

- $x = f(u, v)$, $y = g(u, v)$, $z = h(u, v)$
- You will get to these kinds of surfaces in CSE 784.

Examples: Courtesy of Jason Bryan

Notice Distortions Due To Object Shape

Specify texture coordinates

- Give texture coordinates before defining each vertex

```cpp
glBegin(GL_QUADS);
    glTexCoord2D(0, 0);
    glVertex3f(-0.5, 0, 0.5); ...
    glEnd();
```
Transform texture coordinates

- All the texture coordinates are multiplied by GL_TEXTURE matrix before in use
- To transform texture coordinates, you do:
  - glMatrixMode(GL_TEXTURE);
  - Apply regular transformation functions
  - Then you can draw the textured objects

Map textures to surfaces

- Establish mapping from texture to surfaces (polygons):
  - Application program needs to specify texture coordinates for each corner of the polygon

Texture Representation

- Bitmap (pixel map) textures (supported by OpenGL)
- Procedural textures (used in advanced rendering programs)

Bitmap texture:
- A 2D image - represented by 2D array texture[height][width]
- Each pixel (or called texel) by a unique pair texture coordinate (s, t)
- The s and t are usually normalized to a [0,1] range
- For any given (s,t) in the normalized range, there is a unique image value (i.e., a unique [red, green, blue] set)

Map textures to surfaces

- Texture mapping is performed in rasterization (backward mapping)
- For each pixel that is to be painted, its texture coordinates (s, t) are determined (interpolated) based on the corners' texture coordinates (why not just interpolate the color?)
- The interpolated texture coordinates are then used to perform texture lookup
Texture Mapping

1. projection
2. texture lookup
3. patch texel

3D geometry
2D projection of 3D geometry
2D image

Texture Value Lookup

- For the given texture coordinates (s,t), we can find a unique image value from the texture map

Inverse Mapping

- Screen space → … → object space → texture space

OpenGL texture mapping

- Steps in your program
  1) Specify texture
     - read or generate image
     - Assign to texture
  2) Specify texture mapping parameters
     - Wrapping, filtering, etc.
  3) Enable GL texture mapping (GL_TEXTURE_2D)
  4) Assign texture coordinates to vertices
  5) Draw your objects
  6) Disable GL texture mapping (if you don’t need to perform texture mapping any more)
Specify textures

- Load the texture map from main memory to texture memory
  - `glTexImage2D(GLenum target, GLint level, GLint format, int width, int height, int border, GLenum format, GLenum type, Glvoid* img)`
- Example:
  - `glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0, GL_RGB, GL_UNSIGNED_BYTE, myImage);`
    (myImage is a 2D array: GLuint myImage[64][64][3];)
- The dimensions of texture images must be powers of 2

Fix texture size

- If the dimensions of the texture map are not power of 2, you can
  1) Pad zeros
  2) use `gluScaleImage()`
  - Ask OpenGL to filter the data for you to the right size — you can specify the output resolution that you want

Texture Objects

- Like display lists for texture images
  - one image per texture object
  - may be shared by several graphics contexts
- Generate texture names
  - `glGenTextures(n, *texIds);`

Texture Objects (cont.)

- Create texture objects with texture data and state
  - `glBindTexture(target, id);`
- Bind textures before using
  - `glBindTexture(target, id);`

Remember to adjust the texture coordinates for your polygon corners — you don’t want to include black texels in your final picture
Define a texture image from an array of texels in CPU memory
- `glTexImage2D(target, level, components, w, h, border, format, type, *texels);`
- dimensions of image must be powers of 2
- Texel colors are processed by pixel pipeline
  - pixel scales, biases and lookups can be done

If dimensions of image are not power of 2
- `gluScaleImage(format, w_in, h_in, type_in, *data_in, w_out, h_out, type_out, *data_out);`
  - *in is for source image
  - *out is for destination image
- Image interpolated and filtered during scaling

Wrapping Modes
- repeat: Start entire texture over
- mirror: Flip copy of texture in each direction
  - Get continuity of pattern
  - Rather new OpenGL feature.
- clamp to edge: Extend texture edge pixels
- clamp to border: Surround with border color

Texture mapping parameters
- Example: `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)`
Texture mapping parameters (2)

- Since a polygon can get transformed to arbitrary screen size, texels in the texture map can get magnified or minified.
- Filtering: interpolate a texel value from its neighbors or combine multiple texel values into a single one.

Texture mapping parameters (3)

- OpenGL texture filtering:
  1) Nearest Neighbor (lower image quality)
  2) Linear interpolate the neighbors (better quality, slower)

Texture color blending

- Determine how to combine the texel color and the object color
  - GL_MODULATE – multiply texture and object color
  - GL_BLEND – linear combination of texture and object color
  - GL_REPLACE – use texture color to replace object color

Example: `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);`

Texture Application Modes

- **decal**: Overwrite object’s color or material with texel

  ![Texture Application Modes: decal](image)

- **modulate**: Combine object pixel with texel via multiplication

  ![Texture Application Modes: modulate](image)
Put it all together

```
...  
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);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
...

glEnable(GL_TEXTURE_2D);
gTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0, GL_RGB,
              GL_UNSIGNED_BYTE, mytexture);

Draw_picture1(); // define texture coordinates and vertices in the function
```

Advanced: Multitexture + Modulation

- New cards can modulate multiple textures
- CSE 781 will examine many uses of texture mapping.

**Light maps**

- Also can avoid the expensive lighting calculations.

```
Light maps
```

```
Simple Applications: Billboards
```

[Images courtesy of www.gamasutra.com]
Texture Rasterization

- Texture coordinates are interpolated from polygon vertices just like ... remember ...
  - Color: Gouraud shading
  - Depth: Z-buffer

  - First along polygon edges between vertices
  - Then along scanlines between left and right sides

Why?

- Equal spacing in screen (pixel) space is not the same as in texture space in perspective projection

  - Perspective foreshortening

Linear Texture Coordinate Interpolation

- This doesn’t work in perspective projection!
- The textures look warped along the diagonal
- Noticeable during an animation

Perspective-Correct Texture Coordinate Interpolation

- Interpolate \((\text{tex\_coord}/w)\) over the polygon, then do perspective divide after interpolation

  - Compute at each vertex after perspective transformation
    - "Numerators" \(s/w, t/w\)
    - "Denominator" \(1/w\)

  - Linearly interpolate \(1/w, s/w, \text{ and } t/w\) across the polygon

  - At each pixel
    - Perform perspective division of interpolated texture coordinates \((s/w, t/w)\) by interpolated \(1/w\) (i.e., numerator over denominator) to get \((s, t)\)
Perspective-Correct Interpolation

- That fixed it!

Perspective-Correct Interpolation: Notes

- But we didn’t do this for Gouraud shading…
  - Actually, we should have, but the error is not as obvious
- Alternative: Use regular linear interpolation with small enough polygons that effect is not noticeable
- Linear interpolation for Z-buffering is correct

Perspective Correction Hint

- Texture coordinate and color interpolation:
  - Linearly in screen space (wrong) OR
  - Perspective correct interpolation (slower)

- `glHint (GL_PERSPECTIVE_CORRECTION_HINT, hint)`, where `hint` is one of:
  - GL_NICEST: Perspective
  - GL_FASTEST: Linear
  - GL_DONT_CARE: Linear

Texture Objects

- Like display lists for texture images
  - one image per texture object
  - may be shared by several graphics contexts
- Generate texture names
  - `glGenTextures(n, *texIds);`
Texture Objects (cont.)

- Create texture objects with texture data and state
  ```c
  glBindTexture( target, id );
  ```
- Bind textures before using
  ```c
  glBindTexture( target, id );
  ```

Texture Animation

- Just change the texture coordinates

Texture Animation

- Sprite Animations

Sprites and Billboards

- Sprites – usually refer to 2D animated characters that move across the screen.
  - Like Pacman
  - Three types (or styles) of billboards
    - Screen-aligned (parallel to top of screen)
    - World aligned (allows for head-tilt)
    - Axial-aligned (not parallel to the screen)
Creating Billboards in OpenGL

- Annotated polygons do not exist with OpenGL 1.3 directly.
- If you specify the billboards for one viewing direction, they will not work when rotated.

Example

Example 2

- The alpha test is required to remove the background.
- More on this example when we look at depth textures.

Re-orienting

- Billboards need to be re-oriented as the camera moves.
- This requires immediate mode (or a vertex shader program).
- Can either:
  - Recalculate all of the geometry.
  - Change the transformation matrices.
Re-calculating the Geometry

- Need a projected point (say the lower-left), the projected up-direction, and the projected scale of the billboard.
- Difficulties arise if we are looking directly at the ground plane.

Undo the Camera Rotations

- Extract the projection and model view matrices.
- Determine the pure rotation component of the combined matrix.
- Take the inverse.
- Multiply it by the current model-view matrix to undo the rotations.

Screen-aligned Billboards

- Alternatively, we can think of this as two rotations.
- First rotate around the up-vector to get the normal of the billboard to point towards the eye.
- Then rotate about a vector perpendicular to the new normal orientation and the new up-vector to align the top of the sprite with the edge of the screen.
- This gives a more spherical orientation.
  - Useful for placing text on the screen.

World Aligned Billboards

- Allow for a final rotation about the eye-space z-axis to orient the billboard towards some world direction.
- Allows for a head tilt.
Axial-Aligned Billboards

- The *up*-vector is constrained in world-space.
- Rotation about the up vector to point normal towards the eye as much as possible.
- Assuming a ground plane, and always perpendicular to that.
- Typically used for trees.

Fin Billboard

- Use two polygons at right angles.
- Typically right angles.
OpenGL Architecture

- Display List
- Rasterization
- Texture Memory
- Pixel Operations
- Frame Buffer
- Per Vertex Operations & Primitive Assembly
- Polynomial Evaluator

Per-Fragment Operations

- Display List
- Rasterization
- Texture Memory
- Pixel Operations
- Frame Buffer
- Per Vertex Operations & Primitive Assembly
- Polynomial Evaluator

Getting to the Framebuffer

- Scissor Test
- Alpha Test
- Stencil Test
- Framebuffer
- Depth Test
- Blending
- Dithering
- Logical Operations

Alpha Test

- Reject pixels based on their alpha value
  - `glAlphaFunc(func, value)`
  - `glEnable(GL_ALPHA_TEST)`
- Use alpha as a mask in textures
Billboard Clouds

- Add several planes for different parts of the model (images from Univ. of Utah).

Demo - SpeedTree

Shaders allow better animation

- Teaser for CSE 781
- Pretty water: