Texture Mapping

OpenGL and Implementation Details

CS351-50

11.05.03

Surface mapping

- Texture mapping
- Bump Mapping
- Displacement mapping
  - Actually moving geometry
  - ie Create screw from cylinder, Terrain, etc

An Overview

- Steps in Texture Mapping
  1) Specify the texture. (R, G, B, A, mipmap)
  2) Indicate how the texture is to be applied to each pixel. (decal, modulate, blend)
  3) Enable texture mapping via glEnable( )
     GL_TEXTURE_1D or GL_TEXTURE_2D
  4) Draw the scene, supplying both texture and geometric coordinates.

From Jim X. Chen, jchen@cs.gmu.edu

At each rendered pixel, selected texels are used either to substitute for or to scale one or more of the surface’s material properties, such as its diffuse color components.

One pixel is often covered by a number of texels: nearest pixel, bilinear interpolation, bi-cubic interpolation, etc.
What does a pixel see?

Pixel and Texel Relations

- Corresponding vertices of texture and polygons
- Minification and magnification filters: nearest or bilinear interpolation
- Texel color components replace or modulate pixel color

Controlling Filtering

Texture Sampling

- Sinc(x) is not feasible in real time
- Box filter (nearest-neighbor) is poor quality
Tent Filter (Linear Interpolation)

- Looks better
- Easy in 1D:
  \[(1-t)\text{Color}_0 + t\text{Color}_1\]

Interpolation

- Texture coordinate \((p_u, p_v)\) in \([0,1]\)
- Texture images size \(n \times m\) texels
- Nearest neighbor would access:
  \((\text{floor}(n*u), \text{floor}(m*v))\)
- Interpolate 1D in \(x\) and \(y\)

Bilinear Interpolation

- Let \(t(u,v)\) access texture map
- \(b(u,v) = \text{filtered texel}\)

\[(u', v') = (p_u - \text{floor}(p_u), p_v - \text{floor}(p_v))\]

\[b(p_{u'}, p_{v'}) = (1 - u') (1-v') t(x_u, y_v) +
  u' (1-v') t(x_v, y_v) +
  (1-u') v' t(x_u, y_t) +
  u' v' t(x_t, y_v)\]

Parameter Values

- \(\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST));}\)
- \(\text{glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST});}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_TEXTURE_MAG_FILTER</td>
<td>GL_NEAREST or GL_LINEAR</td>
</tr>
<tr>
<td>GL_TEXTURE_MIN_FILTER</td>
<td>GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, or GL_LINEAR_MIPMAP_LINEAR</td>
</tr>
</tbody>
</table>
Repeat, Mirror, Clamp, Border

- You can assign texture coordinates outside the range [0,1] and have them either clamp or repeat.
- With repeating textures, if you have a large plane with texture coordinates running from 0.0 to 10.0 in both directions, for example, you'll get 100 copies of the texture tiled together on the screen.
- For most applications where the texture is to be repeated, the texels at the top of the texture should match those at the bottom, and similarly for the left and right edges.
- To clamp the texture coordinates: Any values greater than 1.0 are set to 1.0, and any values less than 0.0 are set to 0.0. Clamping is useful for applications where you want a single copy of the texture to appear on a large surface.
- If the surface-texture coordinates range from 0.0 to 10.0 in both directions, one copy of the texture appears in the lower corner of the surface. The rest of the surface is painted with the texture’s border colors as needed.

```
 glBegin(GL_POLYGON);
  glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
  glTexCoord2f(0.0, 3.0); glVertex3f(-2.0, 1.0, 0.0);
  glTexCoord2f(3.0, 3.0); glVertex3f(0.0, 1.0, 0.0);
  glTexCoord2f(3.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
 glEnd();

 glBegin(GL_POLYGON);
  glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
  glTexCoord2f(0.0, 3.0); glVertex3f(1.0, 1.0, 0.0);
  glTexCoord2f(3.0, 3.0); glVertex3f(2.41421, 1.0, -1.41421);
  glTexCoord2f(3.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
 glEnd();
```

Repeating and Clamping Textures

You can also clamp in one direction and repeat in the other.
Assigning Texture Coordinates

void glTexCoord{1234}{sifd}{v}(TYPE coords);

- Texture coordinates can comprise one, two, three, or four coordinates. They’re usually referred to as the s, t, r, and q coordinates.
- For one-dimensional textures, you use the s coordinate; for two-dimensional textures, you use s and t.
- The q, like w, is typically given the value 1 and can be used to create homogeneous coordinates
- The command to specify texture coordinates, glTexCoord{1}{sifd}{v}, is similar to glVertex{1}{sifd}{v}, glColor{1}{sifd}{v}, and glNormal{1}{sifd}{v}. Usually, texture-coordinate values range between 0 and 1; values can be assigned outside this range, however, with the results described in “Repeating and Clamping Textures.”

Computing Approximate Texture Coordinates

- Polygon aspect ratio 3/2 (w/h); texture aspect ratio 1. To avoid distorting the texture, use texture coordinates of (0.0, 1.0, 1.25), (0.25).
- A can with label 4 units tall and 12 units around (aspect ratio 3/1).
- Textures must have aspect ratio of 2 to 1. We can copy and paste it twice to make an aspect ratio of 1. Let’s approximate the can by 38 (4*12/30) polygons. We can use the following texture coordinates:
  - (0,0), (1/30,0), (1/30,1/3), (0,1/3)
  - (1/30,0), (2/30,0), (2/30,1/3), (1/30,1/3)
  - (2/30,0), (3/30,0), (0,1/3), (2/30,1/3)
- Only a few curved surfaces (cone and cylinders) can be mapped to a flat surface without geodesic distortion. For example, a sphere (cos(θ)cos(φ))sin(θ)). The rectangle can be mapped directly to a rectangular texture map, but the closer you get to the poles, the more distorted the texture.
### Specifying the Texture

```c
void glTexImage2D(GLenum target, GLint level, GLint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels);
```

1. The `target` parameter: constant `GL_TEXTURE_2D`.
2. The `level` parameter for multiple resolutions.
3. `Components` is 1 of 38 symbolic constants. A value of 1 selects the R component, 2 selects the R and A components, 3 selects R, G, and B, and 4 selects R, B, G, and A. It indicates which values are selected for texels.
4. The `width` and `height` parameters give the dimensions of the texture image.
5. The `border` indicates the width of the border, which is usually zero.
6. The `format` and `type` parameters describe the format and data type of the texture image data.

From Jim X. Chen, jchen@cs.gmu.edu

#### Some Minor Things

1. The number of texels for both the width and height of a texture image, not including the optional border, must be a power of 2.
2. `gluScaleImage()` correct/alter the sizes of your textures.
3. `glCopyTexImage2D()` creates a 2D texture using framebuffer data.

**1D textures and 3D textures**

- 3D textures are used for rendering in medical and geoscience applications (CT or MRI images)

```c
void glTexImage1D()
void glTexImage3D()
```

### Mipmapping

- Image pyramid
- Half height and width
- Compute $d$
  - Gives 2 images
- Bilinear Interpolate in each image

From Tomas Akenine-Moller

### MipMapping Memory Requirements

- 3D textures are used for rendering in medical and geoscience applications (CT or MRI images)

```c
void glTexImage1D()
void glTexImage3D()
```
Multiple Levels of Detail

• To use mipmapping, you provide all sizes of your texture in powers of 2 between the largest size and a 1x1 map. OpenGL chooses matching size for texture mapping

A Mipmapping Example: mipmap.c

void myinit(void) {
    ...
    loadImages();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, 32, 32, 0,
        GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage32[0][0][0]);
    glTexImage2D(GL_TEXTURE_2D, 1, 3, 16, 16, 0,
        GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage16[0][0][0]);
    ...
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
}

From Jim X. Chen, jchen@cs.gmu.edu

Mipmapping

• Interpolate between those bilinear values
  – Trilinear interpolation

Mipmapping

• Compute $d$

  OpenGL chooses matching size for texture mapping

  A Mipmapping Example: mipmap.c

  void myinit(void) {
      ...
      loadImages();
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
      glTexImage2D(GL_TEXTURE_2D, 0, 3, 32, 32, 0,
          GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage32[0][0][0]);
      glTexImage2D(GL_TEXTURE_2D, 1, 3, 16, 16, 0,
          GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage16[0][0][0]);
      ...
      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
  }

  From Jim X. Chen, jchen@cs.gmu.edu

Mipmapping

• Compute $d$

  OpenGL chooses matching size for texture mapping

  A Mipmapping Example: mipmap.c

  void myinit(void) {
      ...
      loadImages();
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
      glTexImage2D(GL_TEXTURE_2D, 0, 3, 32, 32, 0,
          GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage32[0][0][0]);
      glTexImage2D(GL_TEXTURE_2D, 1, 3, 16, 16, 0,
          GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage16[0][0][0]);
      ...
      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
  }

  From Jim X. Chen, jchen@cs.gmu.edu

Mipmapping

• Compute $d$

  OpenGL chooses matching size for texture mapping

  A Mipmapping Example: mipmap.c

  void myinit(void) {
      ...
      loadImages();
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
      glTexImage2D(GL_TEXTURE_2D, 0, 3, 32, 32, 0,
          GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage32[0][0][0]);
      glTexImage2D(GL_TEXTURE_2D, 1, 3, 16, 16, 0,
          GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage16[0][0][0]);
      ...
      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
  }

  From Jim X. Chen, jchen@cs.gmu.edu

Anisotropic Texture mapping

• Approximate quad with several smaller mip maps

Anisotropic Texture mapping

• Approximate quad with several smaller mip maps
Anisotropic Texture mapping

• 16 samples

Demo

• Nate Robins Texture Mapping Tutor

• Texture mapping works only in RGB mode in OpenGL
• Example: A Texture-Mapped Checkerboard: checker.c

```c
void makeCheckImage(void)
{
    int i, j, c;
    for (i = 0; i < checkImageHeight; i++) {
        for (j = 0; j < checkImageWidth; j++) {
            c = ((((i&0x8)==0)^((j&0x8)==0))*255);
            checkImage[i][j][0] = (GLubyte) c;
            checkImage[i][j][1] = (GLubyte) c;
            checkImage[i][j][2] = (GLubyte) c;
            checkImage[i][j][3] = (GLubyte) 255;
        }
    }
}
```

```c
void init(void)
{
    makeCheckImage();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    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);
    glTexImage2D(GL_TEXTURE_2D, 0, 4, checkImageWidth, checkImageHeight,
                 0, GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
}
```
void display(void)
{
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
    glEnd();
    glFlush();
    glDisable(GL_TEXTURE_2D);
}

From Jim X. Chen,
jchen@cs.gmu.edu

void initTexture(void)
{
    read_stars_image();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    //   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);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_LUMINANCE, stars_pixels, stars_pixels,
0, GL_LUMINANCE, GL_UNSIGNED_BYTE, stars_image);
}

From Jim X. Chen,
jchen@cs.gmu.edu

void drawTexture(float x, float y, float z)
{
    //   glRasterPos3f(x, y, z);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
    glEnable(GL_TEXTURE_2D);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(x, y, z);
    glTexCoord2f(0.0, 1.0); glVertex3f(-x, y, z);
    glTexCoord2f(1.0, 1.0); glVertex3f(-x, -y, z);
    glTexCoord2f(1.0, 0.0); glVertex3f(x, -y, z);
    glEnd();
    glDisable(GL_TEXTURE_2D);
}

From Jim X. Chen,
jchen@cs.gmu.edu

void display(void)
{
    drawTexture(-2.4*Width, -2.4*Height, -1.9*Width);
    drawRobot(A, B, C, alpha, beta, gama);
    glutSwapBuffers();
}

From Jim X. Chen,
jchen@cs.gmu.edu
**Texture Objects**

- Generate texture objects or names: `glGenTextures()`
- Initially bind texture objects to texture data, including the image arrays and texture properties: `glBindTexture()`
- More often used objects stay in memory (prioritize the objects).
- Bind and rebind texture objects, making their data available for rendering texture models.
- When we call `glBindTexture()` with a texture name, all subsequent `glTex*()` commands that specify the texture and its associated parameters are saved in the memory.

**Automatic Texture-Coordinate Generation**

- To make contours on a 3D model
- To simulate reflections from an arbitrary environment on a shiny model
  ```
  glGen* (Glenum coord, Glenum pname, TYPE param);
  glGen*+v (Glenum coord, Glenum pname, TYPE *param);
  ```
  - `GL_S` or `GL_T`
  - Which coord to be auto. generated

**Environment Mapping**

- Assume environment infinitely far away
- Sphere mapping
- Cube mapping (now norm)
  - No singularities
  - Much less distortion
  - Better result
  - Not dependent on view position

**Cube Mapping**

- Simple math:
  - Compute reflection vector \( r \)
  - Largest abs-value of component determines which cube face
    - Example: \( r = (5, -1, 2) \) give POS_X face
    - Divide \( r \) by 5 gives \((u,v) \approx -1/5, 2/5)\)
  - Hardware often does all the work
Bump Mapping

\[ \text{Geometry} + \text{Bump map} = \text{Bump mapped geometry} \]

Bump Mapping Example

Displacement Mapping

- In displacement mapping, the surface is actually modified, in contrast to bump mapping where only the surface normal is modified. This means that displacement mapped surfaces will show the effect even at the silhouette.
- A Pixar Renderman image with displacement mapping. Notice that the surface not only appears non-smooth but the silhouette is also non-smooth.