CS559: Computer Graphics

Lecture 22: Texture mapping

Li Zhang

Spring 2010

Many slides from Ravi Ramamoorthi, Columbia Univ, Greg Humphreys, UVA and Rosalee Wolfe, DePaul tutorial teaching texture mapping visually
Texture Mapping

Linear interpolation of texture coordinates
Correct interpolation with perspective divide


Hill Figure 8.42
Why don’t we notice?

Traditional screen-space Gourand shading is wrong. However, you usually will not notice because the transition in colors is very smooth (And we don't know what the right color should be anyway, all we care about is a pretty picture). There are some cases where the errors in Gourand shading become obvious.

• When do we notice?
  – When switching between different levels-of-detail representations
  – At "T" joints.
Dealing with Incorrect Interpolation

You can reduce the perceived artifacts of non-perspective correct interpolation by subdividing the texture-mapped triangles into smaller triangles (why does this work?). But, fundamentally the screen-space interpolation of projected parameters is inherently flawed.

http://groups.csail.mit.edu/graphics/classes/6.837/F98/Lecture21/Slide15.html
Mipmap

Let \( d = |\text{PIX}| \) be a measure of pixel size

Option 1: use the longer edge of the quadrilateral formed by the pixel's cell to approximate the pixel's coverage

Option 2: use the max of \(|du/dx|, |du/dy|, |dv/x|, |dv/dy|\)

Then take logarithm
Storing MIP Maps

• One convenient method of storing a MIP map is shown below (It also nicely illustrates the 1/3 overhead of maintaining the MIP map).
Ripmap

Sample \((u,v,du,dv)\) Using Trilinear \(\Rightarrow\) Quadrilinear

What’s the memory overhead?
Figure 6.15: The pixel cell is back-projected onto the texture, bound by a rectangle, and the four corners of the rectangle are used to access the summed-area table.

\[
c = \frac{S[x_{ur}, y_{ur}] - S[x_{ur}, y_{ll}] - S[x_{ll}, y_{ur}] + S[x_{ll}, y_{ll}]}{(x_{ur} - x_{ll})(y_{ur} - y_{ll})}
\]

What's the memory overhead?
Simple OpenGL Example

public void Draw() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslated(centerx, centery, depth);
    glMultMatrixf(Rotation);
    // Draw Front of the Cube
    glEnable(GL_TEXTURE_2D);
    glBegin(GL_QUADS);
        glTexCoord2d(0, 1);
        glVertex3d(1.0, 1.0, 1.0);
        glTexCoord2d(1, 1);
        glVertex3d(-1.0, 1.0, 1.0);
        glTexCoord2d(1, 0);
        glVertex3d(-1.0, -1.0, 1.0);
        glTexCoord2d(0, 0);
        glVertex3d(1.0, -1.0, 1.0);
    glEnd();
    glDisable(GL_TEXTURE_2D);
    glFlush();
}

glTexCoord works like glColor

- Specify a texture coordinate at each vertex \((s, t)\)
- Canonical coordinates where \(s\) and \(t\) are between 0 and 1

Slides from Jingyi Yu
static GLubyte image[64][64][4];
static GLuint texname;

void init(void) {
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
    glEnable(GL_DEPTH_TEST);
    //load in or generate image;
    ...
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

glGenTextures(1, &texName);
glBindTexture(GL_TEXTURE_2D, texName);

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_RGBA, checkImageWidth, checkImageHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
}
Other Issues with Textures

- Tedious to specify texture coordinates for every triangle
- Textures are attached to the geometry
- Can't use just any image as a texture

The "texture" can't have projective distortions
Reminder: linear interpolation in image space is not equivalent to linear interpolation in 3-space (This is why we need "perspective-correct" texturing). The converse is also true.

- Makes it hard to use pictures as textures
Projective Textures

- Treat the texture as a light source (like a slide projector)
- No need to specify texture coordinates explicitly
- A good model for shading variations due to illumination (cool spotlights)
- A fair model for view-dependent reflectance (can use pictures)
The Mapping Process

This is the same process, albeit with an additional transform, as perspective correct texture mapping. Thus, we can do it for free! Almost.
What transformation do we need?

- OpenGL is able to insert this extra projection transformation for textures by including another matrix stack, called GL_TEXTURE.
- Also we can use 3d vertex coordinate as texture coordinates
- The transform we want is:

$$T_{\text{eye}} = \begin{bmatrix}
\frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & \frac{1}{2} & 0 & \frac{1}{2} \\
0 & 0 & \frac{1}{2} & \frac{1}{2} \\
0 & 0 & 0 & 1
\end{bmatrix} P_{\text{proj}} V_{\text{proj}}$$

This extra matrix maps from normalized device coordinates ranging from \([-1,1]\) to valid texture coordinates ranging from \([0,1]\).

This matrix positions the projector in the world, much like the viewing matrix positions the eye within the world. (HINT, you can use gluLookAt() to set this up if you want.

This matrix specifies the frustum of the projector. It is a non-affine, projection matrix. You can use any of the projection transformations to establish it, such as glFrustum(), glOrtho() or gluPerspective().
OpenGL Example

Here is where the extra “Texture” transformation on the vertices is inserted.

```java
private void projectTexture() {
    glMatrixMode(GL_TEXTURE);
    glLoadIdentity();
    glTranslated(0.5, 0.5, 0.5); // Scale and bias the [-1,1] NDC values
    glScaled(0.5, 0.5, 0.5);    // to the [0,1] range of the texture map
    gluPerspective(15, 1, 5, 7); // projector “projection”
    gluLookAt(lightPosition[0], lightPosition[1], lightPosition[2],
              0, 0, 0, 0, 1, 0);
    glMatrixMode(GL_MODELVIEW);
}
```

How to know where the light is to project an captured image?
CS766 Computer Vision
Here is a code fragment implementing projective textures in OpenGL

```c
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, (int) GL_OBJECT_LINEAR);
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, (int) GL_OBJECT_LINEAR);
    glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, (int) GL_OBJECT_LINEAR);
    glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, (int) GL_OBJECT_LINEAR);

    // These calls initialize the TEXTURE_MAPPING function to identity. We will be using
    // the Texture matrix stack to establish this mapping indirectly.

    float [] eyePlaneS = { 1.0f, 0.0f, 0.0f, 0.0f };
    float [] eyePlaneT = { 0.0f, 1.0f, 0.0f, 0.0f };
    float [] eyePlaneR = { 0.0f, 0.0f, 1.0f, 0.0f };
    float [] eyePlaneQ = { 0.0f, 0.0f, 0.0f, 1.0f };

    glTexGenfv(GL_S, GL_EYE_PLANE, eyePlaneS);
    glTexGenfv(GL_T, GL_EYE_PLANE, eyePlaneT);
    glTexGenfv(GL_R, GL_EYE_PLANE, eyePlaneR);
    glTexGenfv(GL_Q, GL_EYE_PLANE, eyePlaneQ);
```
OpenGL Example (Shortcut)

Here is a code fragment implementing projective textures in OpenGL

// The following information is associated with the current active texture
// Basically, the first group of setting says that we will not be supplying texture coordinates.
// Instead, they will be automatically established based on the vertex coordinates in “EYE-SPACE”
// (after application of the MODEL_VIEW matrix).

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);

// These calls initialize the TEXTURE_MAPPING function to identity. We will be using the Texture matrix stack to establish this mapping indirectly.

float [] eyePlaneS = { 1.0f, 0.0f, 0.0f, 0.0f };
float [] eyePlaneT = { 0.0f, 1.0f, 0.0f, 0.0f };
float [] eyePlaneR = { 0.0f, 0.0f, 1.0f, 0.0f };
float [] eyePlaneQ = { 0.0f, 0.0f, 0.0f, 1.0f };

glTexGenfv(GL_S, GL_EYE_PLANE, eyePlaneS);
glTexGenfv(GL_T, GL_EYE_PLANE, eyePlaneT);
glTexGenfv(GL_R, GL_EYE_PLANE, eyePlaneR);
glTexGenfv(GL_Q, GL_EYE_PLANE, eyePlaneQ);
GL_OBJECT_LINEAR vs GL_EYE_LINEAR

When the mode is GL_OBJECT_LINEAR, the texture coordinate is calculated using the plane-equation coefficients that are passed via `glTexGenfv(GL_S, GL_OBJECT_PLANE, planeCoefficients)`.

The coordinate (S in this case) is computed as:

\[ S = Ax + By + Cz + D \]

using the \([x \ y \ z]\) coordinates of the vertex (the values passed to `glVertex`).

If \([A \ B \ C]\) is a unit vector, this means that the texture coordinate \(S\) is equal to the distance of the vertex from the texgen plane.

GL_EYE_LINEAR works similarly, with the coefficients passed via `glTexGenfv(GL_S, GL_EYE_PLANE, planeCoefficients)`.

The difference is that GL_OBJECT_LINEAR operates in "object coordinates", while GL_EYE_LINEAR works in "eye coordinates".

This means that the texture coordinates computed with GL_OBJECT_LINEAR depend solely on the values passed to `glVertex`, and do not change as the object (or camera) moves via transformations.

The texture coordinates computed with GL_EYE_LINEAR use the positions of vertices after all modeling & viewing transformations - i.e. they use the positions of the vertices on the screen.

http://resumbrae.com/ub/dms424_s05/10/print.html
OpenGL Example (cont)
The following code fragment is inserted into Draw( ) or Display( )

```c
if (projTexture) {
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_GEN_R);
    glEnable(GL_TEXTURE_GEN_Q);
    projectTexture();
}

// … draw everything that the texture is projected onto
if (projTexture) {
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);
    glDisable(GL_TEXTURE_GEN_R);
    glDisable(GL_TEXTURE_GEN_Q);
    glDisable(GL_TEXTURE_GEN_Q);
}
```
Outline

• *Types of mappings*
• Interpolating texture coordinates
• Texture Resampling
• Texture mapping in OpenGL
• *Broader use of textures*
Modulation textures

Map texture values to scale factor

Wood texture
Illumination Maps

- Quake introduced *illumination maps* or *light maps* to capture lighting effects in video games.

Texture map: + light map:
Bump Mapping

• Texture = change in surface normal!

Sphere w/ diffuse texture  Swirly bump map  Sphere w/ diffuse texture and swirly bump map
Bump map representation

$(bu, bv)$ as texel values

What happens if the light is head on?

High values as texel values need to be converted to normal

Figure 6.22 from RTR book
More Bump Map Examples

Since the actual shape of the object does not change, the silhouette edge of the object will not change. Bump Mapping also assumes that the Illumination model is applied at every pixel (as in Phong Shading or ray tracing).
One More Bump Map Example
Dot product normal map

(r,g,b) encodes (nx,ny,nz)

Figure 6.24 from RTR book
Bump map in shading

\[ I = k_e + k_a L_a + k_d L_d \cdot \max(0, L \cdot N) + k_s L_s \cdot \max(0, V \cdot R)^{n_s} \]
Displacement Mapping

Texture maps can be used to actually move surface points. This is called \textit{displacement mapping}. How is this fundamentally different than bump mapping?

Bump map does not have occlusion effects.
Rendering Displacement map is tricky

How to find the intersection?
See RTR book

Figure 6.29 from RTR book
Bump map vs Displacement map

Figure 6.24 and 30 from RTR book
Displacement map in a scene
Solid textures

- Use model-space coordinates to index into a 3D texture
- Like “carving” the object from the material

• One difficulty of solid texturing is coming up with the textures.
Solid textures (cont'd)

• Here's an example for a vase cut from a solid marble texture:

  ![Vase cut from solid marble texture](http://legakis.net/justin/MarbleApplet/)

  • *Solid marble texture by Ken Perlin*

  [http://legakis.net/justin/MarbleApplet/](http://legakis.net/justin/MarbleApplet/)
Environment Maps

Images from *Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments*
Gene Miller and C. Robert Hoffman
SIGGRAPH 1984 “Advanced Computer Graphics Animation” Course Notes
Environment Maps

Instead of using transformed vertices to index the projected texture', we can use transformed *surface normal* s to compute indices into the texture map. These sorts of mapping can be used to simulate reflections, and other shading effects. This approach is not completely accurate. It assumes that all reflected rays begin from the same point, and that all objects in the scene are the same distance from that point.
Applications of Environment Map

"Interface", courtesy of Lance Williams, 1985
williams_robot4.mov

Terminator 2, 1991

More history info on Reflectance map
http://www.debevec.org/ReflectionMapping/
Sphere Mapping Basics

- OpenGL provides special support for a particular form of Normal mapping called sphere mapping. It maps the normals of the object to the corresponding normal of a sphere. It uses a texture map of a sphere viewed from infinity to establish the color for the normal.
Sphere Mapping

- Mapping the normal to a point on the sphere
Sphere Mapping

- Mapping the normal to a point on the sphere

Recall:

\[ r = 2(\overline{n} \cdot \overline{v})\overline{n} - \overline{v} \]
\[ r + \overline{v} = \alpha \overline{n} \]

\[ \alpha \overline{n} = \overline{r} + \overline{v} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \]
\[ \frac{\alpha \overline{n}}{||\alpha \overline{n}||} = \begin{bmatrix} \frac{r_x}{p} \\ \frac{r_y}{p} \\ r_z + \frac{1}{p} \end{bmatrix} \]
\[ p = \sqrt{r_x^2 + r_y^2 + (r_z + \frac{1}{p})^2} \]

\[ s = \frac{r_x}{p} \quad t = \frac{r_y}{p} \]

\[ s' = \frac{s}{2} + \frac{1}{2} \quad t' = \frac{t}{2} + \frac{1}{2} \]

\[ s' = \frac{r_x}{2p} + \frac{1}{2} \quad t' = \frac{r_y}{2p} + \frac{1}{2} \]
OpenGL code Example

// this gets inserted where the texture is created

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, (int) GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, (int) GL_SPHERE_MAP);

glEnable(GL_TEXTURE_2D);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);

Without a mirror ball, where to get environment map images?
http://www.debevec.org/probes/

When doesn’t EM work well?
• Flat/planar surface
  • In particular under orthographic projection
What’s the Best Map?

A sphere map is not the only representation choice for environment maps. There are alternatives, with more uniform sampling properties, but they require different normal-to-texture mapping functions.
Multi-Texturing
Why Multi-texturing

• Interpolating Textures
  – From day to night

• Implementing Bump maps
  – Create texture for diffuse coefficient, light vector, and normal maps: \( kd \times \max(L\cdot n, 0) \)
Multi-Texture: OpenGL

Gluint texture0, texture1;
// in Init() Load in images, Initialize textures as before
// in Display() do the following

// bind and enable texture unit 0
glActiveTexture (GL_TEXTURE0);
glBindTexture (GL_TEXTURE_2D, texture0);
 glEnable (GL_TEXTURE_2D);

// bind and enable texture unit 1
glActiveTexture (GL_TEXTURE1);
 glBindTexture (GL_TEXTURE_2D, texture1);
 glEnable (GL_TEXTURE_2D);

// specify two sets of texture coordinates for each vertex
 glBegin (GL_TRIANGLES);
  glMultiTexCoord2f (GL_TEXTURE0, 0.0, 1.0);
  glMultiTexCoord2f (GL_TEXTURE1, 1.0, 0.0);
  glVertex3f (...)
  ...
 glEnd ();
Combining Textures

```c
glActiveTexture(GL_TEXTURE0);
 glEnable(GL_TEXTURE_2D);
 glBindTexture(GL_TEXTURE_2D, texName0);
 glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

glActiveTexture(GL_TEXTURE1);
 glEnable(GL_TEXTURE_2D);
 glBindTexture(GL_TEXTURE_2D, texName1);
 int choice = 0;
 if (choice == 0) //modulate the two textures
 {
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
 }
 else if (choice == 1) //interpolate the two textures using the alpha of current primary color
 {
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
   glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB, GL_INTERPOLATE);
   glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_RGB, GL_TEXTURE1);
   glTexEnvi(GL_TEXTURE_ENV, GL_SRC1_RGB, GL_PREVIOUS);
   glTexEnvi(GL_TEXTURE_ENV, GL_SRC2_RGB, GL_PRIMARY_COLOR);
   glTexEnvi(GL_TEXTURE_ENV, GL_OPERAND0_RGB, GL_SRC_COLOR);
   glTexEnvi(GL_TEXTURE_ENV, GL_OPERAND1_RGB, GL_SRC_COLOR);
   glTexEnvi(GL_TEXTURE_ENV, GL_OPERAND2_RGB, GL_SRC_ALPHA);
 }
 else //disable the second texture
 {
   glActiveTexture(GL_TEXTURE1);
   disable(GL_TEXTURE_2D);
 }
```

```c
// continue on the right
```
Different Combinations for different apps

• Interpolate for fade-in-fade-out between textures
  – Night and day

• Dot product followed by modulation for bump map
  – [http://www.paulsprojects.net/tutorials/simplebump/simplebump.html](http://www.paulsprojects.net/tutorials/simplebump/simplebump.html)
Multi-Texture: OpenGL

• Each texture unit (GL_TEXTURE*) has its own state:
  – Texture image
  – Environment mode
  – Filtering parameters
  – Texture matrix stack
  – Automatic texture coordinate generation
  – ...

• Create visual effect using your imagination.
GL Extensions

- Multi-texturing is not supported in visual studio.
- Need to download GL extension managers:
Texture animation

- Moving water texture to simulate flow
- zoom, rotation, and shearing image on a surface
- Fading in and fading out (Blending marble to skin) with multi-texturing
Shadow map

- Cast shadow on curved/non planar surfaces

Lance Williams, “Casting Curved Shadows on Curved Surfaces,” SIGGRAPH 78
Shadow map

Let's consider this scene:

Point light source

Q: Is hack shadow enough?
Shadow map

With shadow

Without shadow
Shadow map

Scene from the light’s viewpoint
In rasterization, check light visibility using the depth map with respect to the light.
Shadow map

The A < B shadowed fragment case

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE, GL_COMPARE_R_TO_TEXTURE);
Shadow map

The $A \equiv B$ unshadowed fragment case

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE, GL_COMPARE_R_TO_TEXTURE);
Shadow Map

• Depth testing from the light’s point-of-view

A Two pass algorithm:

First, render depth buffer from the light’s point-of-view

– the result is a “depth map” or “shadow map” essentially a 2D function indicating the depth of the closest pixels to the light

– This depth map is used in the second pass
Shadow Map

• Shadow determination with the depth map

Second, render scene from the eye’s point-of-view

– For each rasterized fragment
  • determine fragment’s XYZ position relative to the light
  • compare the depth value for XYZ and the depth value in the buffer computed in the first pass, relative to the light.

More details, see Red book v2.1. pp459-463
Free Version 1.1 does not have this feature.
Also see a tutorial
http://www.paulsprojects.net/tutorials/smt/smt.html