Lecture 12 of 41
Surface Detail 3 of 5: Mappings
OpenGL Textures

William H. Hsu
Department of Computing and Information Sciences, KSU

Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:
Today: Sections 20.5 – 20.13, Eberly
Next class: Section 3.1, Eberly

Surface Detail 3 of 5: Mappings
OpenGL Textures
Lecture 12 of 41

Reading for Last Class: §2.6.3, 20.3 – 20.4, Eberly
Reading for Today: §20.5 – 20.13, Eberly
Reading for Next Class: §3.1, Eberly

Last Time: Texture Mapping Explained
Definitions and design principles
Enclosing volumes: cylinder, sphere, box
Mapping methods
reflected ray – bounce ray off object
object normal – ray from normal of object (polygon mesh)
object center – ray from center of object
Intermediate surface normal – ray from inside of enclosing
Today: Mappings, OpenGL Texturing
Shadow, reflection/environment, transparency, bump, displacement
Other mappings: gloss, volumetric fog, skins, rainbows, water
OpenGL texture mapping how-to

Overview of Mappings: Eberly 2e Chapter 20 Sections
Fine Surface Detail: Bump (§20.5 Eberly 2e)
Material Effects: Gloss (§20.6)
Enclosing Volumes
Sphere (§20.7)
Cube (§20.8)
Light
Refraction for Transparency (§20.9)
Reflection, aka Environment (§20.10)
Shadow
Shadow Maps (§20.11, 20.13)
Projective Textures (§20.12)
More Special Effects (SFX)
Fog (§20.14)
Skinning (§20.15)
Iridescence (§20.16), Water (§20.17)

Acknowledgements: Many Mappings
Stefan Jeschke
Research Assistant
http://www.cis.ksu.edu/~bhsu

Frank Pfenning
Professor of Computer Science
School of Computer Science
Carnegie Mellon University
http://www.cs.cmu.edu/~fp/

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University

Review:
OpenGL Shading (Overview)
Set Up Point Light Sources
Set Up Materials, Turn Lights On
Start Drawing (glBegin … glEnd)

Overview of Mappings: Eberly 2e Chapter 20 Sections
Fine Surface Detail: Bump (§20.5 Eberly 2e)
Material Effects: Gloss (§20.6)
Enclosing Volumes
Sphere (§20.7)
Cube (§20.8)
Light
Refraction for Transparency (§20.9)
Reflection, aka Environment (§20.10)
Shadow
Shadow Maps (§20.11, 20.13)
Projective Textures (§20.12)
More Special Effects (SFX)
Fog (§20.14)
Skinning (§20.15)
Iridescence (§20.16), Water (§20.17)
Computing & Information Sciences
Kansas State University

7
CIS 536/636
Introduction to Computer Graphics
Lecture 12 of 41

Shadow Mapping [1]: Basic Concept
- Process for Adding Shadows in 3-D CG
- Compatible with Local Illumination
- Global method: shadow rays
- Not needed here as in raytracing
- Instead, use decaling
- Decals
  - “Paste” surface detail onto model
  - Semi-transparent: alpha blending
  - Can simulate many attributes

Without shadow map
With shadow map

Shadow Mapping © 2007 XVR Wiki

- Ways to Handle Shadows
  - Projected planar shadows: works well on flat surfaces only
  - Shadow stencil buffer: powerful, excellent results possible; hard!
- OpenGL Shadow Mapping Tutorials

Adapted from "Shadow Mapping" © 2001 C. Everitt, nVidia

- Reflection/Environment Mapping [1]: Basic Concept
  - For a given viewing direction
  - For each normal direction
  - For each incoming direction (hemispherical integral)
  - Evaluate reflection equation
  - Idea: Take Picture of Scene Faced by Object, Apply as Map to Object
  - Requirements: Need to Take Account of Projective Distortions

Reflection/Environment Mapping © 2009 K. Turkowski

- Reflection/Environment Mapping [2]: Techniques
  - Gazing Ball (Mirrorball)
  - Reflection Functions
    - Diffuse: irradiance map
    - Glossy: radiance map
    - Anisotropic: for each tangent direction
  - Mirror: reflection map (related to environment map)
  - Illumination Functions: Environment Map or Procedural Light Sources

Reflection/Environment Mapping Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University

- Reflection/Environment Mapping [3]: Advanced Methods & Research
  - Shadow Mattes (Hanrahan)
    - Can Be Layered (See Maya 2011 Tutorial by Maciek Gryka)

Reflection/Environment Mapping Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University
Transparency Mapping [1]: Basic Concept

- Transparency: One Term for Many Techniques
  - Goal: “See Through” Objects (Could Be Real Decals)
  - Ideas: Render Background Object, Then Foreground Object or Material
    - Blend in color of (semi-)transparent/translucent foreground object
    - Simulate little holes in foreground material (screen door)

Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University

Transparency Mapping [2]: Techniques

- Alpha Compositing aka Alpha Blending
  - Combine colors of transparent foreground, opaque background
  - Uses alpha channel $A$ of $(R, G, B, A)$ – think “% transparency”

Screen Door Transparency

- Simulate little holes in foreground material (screen door)
- Result: visual effect of being able to see through foreground

Transparency Mapping [3]: Advanced Methods & Research

- Screen Door Transparency
  - Use glPolygonStipple(), glEnable(GL_POLYGON_STIPPLE)
- Glass-Like Transparency using Alpha Blending
  - Use glEnable(GL_BLEND), glBlendFunc(...)
  - See http://bit.ly/hudsR27a

Bump Mapping [1]: Basic Concept

- Goal: Create Illusion of Textured Surface
- Idea
  - Start with regular smooth object
  - Make height map (by hand and/or using program, i.e., procedurally)
  - Use map to perturb surface normals
  - Plug new normals into illumination equation
- Will This Look Realistic? Why/Why Not?

Bump Mapping [2]: Techniques

- From Blinn 1976

Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University

Bump Mapping [3]: Advanced Methods & Research

- Right Ball (Displacement Mapped) Casts Rough Shadow
- Left Ball (Bump Mapped) Casts Smooth Shadow – Why?
- Bump Mapping Only Perturbs Normals (Surface Only)
Displacement Mapping [1]: Basic Concept
- Remember What We Did to Perform Bump Mapping?

\[
\mathbf{F}(u, v) = \mathbf{F}(u, v) + \Delta \mathbf{N}(u, v)
\]

- Displacement
- Perturbed normal

Q: Can We Make This Permanent? How?
A: Sure! Let Perturbed Normals Define New Surface; Save Out Vertices

Displacement Mapping [2]: Techniques
- Displacement Map: Similar to Bump Map – Contains Delta Values

- Displacement Mapping: Uses Open GL Shading Language (GLSL)

Displacement Mapping [3]: Advanced Methods & Research
- When To Consider Using Displacement Mapping
  - Very "deep" texture effect: veins, ridges, etc.
  - Shadows expected

Like Many Mappings and Other Effects, Wanted In Hardware!

Acknowledgements: Texture Mapping Slides
Andy van Dam
T. J. Watson University Professor of Technology and Education & Professor of Computer Science
Brown University
http://www.cs.brown.edu/~avd

Texture Mapping Technique [1]
- Texture mapping is the process of mapping a geometric point in space to a value (color, normal, other...) in a texture
- Our goal is to map any arbitrary geometry to a texture of any dimension
- This is done in two steps:
  1. Map a point on the geometry to a point on the unit square
  2. Map the unit square point to a point on the texture
- Second mapping is much easier, we'll present it first.

Texture Mapping Technique [2]
- Mapping a point in the unit u, v square to a texture of arbitrary dimensions:
  - In general, any point (u, v) on the unit square, the corresponding point on the texture of length \( L \) pixels and height \( H \) pixels is \((x, y, z) = (uL, vH, k)\).
  - Above: \((0, 0, 0) \rightarrow (0, 0), (1, 0, 0) \rightarrow (1000, 1000), (7, 425) \rightarrow (1400, 425)\)
  - Once we have coordinates for the texture, we need to look up the color of the texture at these coordinates
  - Coordinates not always a discrete point on texture as they come from continuous space. May need to average neighboring texture pixels (i.e. filter)
**Texture Mapping Technique [3]**

- Texture mapping polygons
  - (x, y) texture coordinates are pre-calculated and specified per vertex
  - Vertices may have different texture coordinates for different faces

- Texture coordinates are linearly interpolated across polygon
  
  \[(x_0, y_0), (x_1, y_1), (x_2, y_2)\]

- Barycentric coordinates

- Interpolation Trick: Barycentric Coordinates

  - Consider interpolating between two values along a line
    - Given two colors \(C_0\) and \(C_1\), you can compute any value along the "line" between the two colors by evaluating:

  \[C(t) = (1 - t)C_0 + tC_1\]

  - This equation can be written as:

  \[C(x, u) = xC_0 + uC_1\]

  - \(x\) and \(u\) are the Barycentric Coordinates of the line segment between \(C_0\) and \(C_1\)

  - The Eq of the line is a convex linear combination of its endpoints. We've seen this before (splines, color theory)

  - Barycentric coordinates can be generalized to triangles

- Applying Barycentric Coordinates

  - When you intersect a ray with a polyhedral object (not needed for our intersection projects)
    - return the barycentric coordinates \((x_0, y_0, z_0)\) of the intersection point
    - These coordinates can be used to interpolate between vertex colors, normals, texture coordinates, or other data

- What weights do we use for our vertex set such that the triangle would be perfectly balanced on a point P?

- Alternatively, think of a smiley face from P with a nose \(A_0\), \(A_1\), and \(A_2\)

- Compute CoC of intersection of line through \(A_0\) and \(P\) and \(A_1\) and \(P\)

- \(x' = x_0 - x_1\)

- \(t' = \frac{x'}{x_2 - x_0}\)

- \(x' = \frac{x_0 - x_1}{x_2 - x_0}\)

- \(w' = 1 - t'\)

- \(x = (1 - t')x_0 + t'x_1\)

- Another way of thinking about this is triangle area. The weight \(w'\) should be proportional to the area of the triangle \(A_1, A_0, A_2\)

- Texture Mapping Technique [4]: Map Point to Object on \((u, v)\) Square

- Texture mapping in "Ray" mapping solids

- Using ray tracing, we obtain an intersection point \((x, y, z)\) in object space

- We need to map this point to a point on the \((u, v)\) unit square, so we can map that to a texture value

- Three easy cases: planes, cylinders, and spheres

- Easiest to compute the mapping from \((x, y, z)\) coordinates in object space to \((u, v)\)

- Can cause unwanted texture scaling

- Texture filtering is an option in most graphics libraries

- OpenGL allows you to choose filtering method (GL_NEAREST, GL_LINEAR, etc...)

- Texture Mapping Technique [5]

- How to texture map cylinders and cones:

  - Given a point \(P\) on the surface:
    - If \(P\) is on one of the caps, map as though the cap is a plane
    - If \(P\) is on the curved surface:
      - Use the height of the point to determine \(v\)

- Mapping \(v\) is trivial, \([0, 1, \frac{1}{2}]\) gets mapped to \([u, a, b]\) just by adding \(s\)

- Texture Mapping Technique [6]

  - Given a point \(P\) on the surface:
    - If \(P\) is on one of the caps, map as though the cap is a plane
    - If \(P\) is on the curved surface:
      - Use the height of the point to determine \(v\)

- Mapping \(v\) is trivial, \([0, 1, \frac{1}{2}]\) gets mapped to \([u, a, b]\) just by adding \(s\)
Texture Mapping Technique [7]

- Computing the $s$ coordinate for cones and cylinders:
  - We need to map all the points on the perimeter of the object to $[0, 1]$
  - The easiest way is to say $s = x$, but computing $d$ can be tricky

- Standard $\sin$ function computes a result for $d$ but it’s always between $0$ and $\pi$ and it maps two positions on the perimeter to the same $d$ value.
- Examples: $\sin d = 0$: $d = 0, 180^\circ$
- $\sin d = 1$: $d = 90^\circ, 270^\circ$
- $\sin d = -1$: $d = 270^\circ, 90^\circ$
- $\sin d = 0$: $d = 360^\circ, 0^\circ$ (The same cycle)

Texture Mapping Technique [8]

- Texture mapping spheres:
  - Find $(u, v)$ coordinates for $P$
  - We compute the same we do for cylinders and cones

- If $u = 0$ or $u = 1$, there is a singularity. Set $u$ to some predefined value ($0.5$ is good)
- $d$ is a function of the latitude of $P$

$$
\sin \frac{\phi}{2} = \frac{r}{2} \Rightarrow \phi = \pi - \frac{2 \cdot \sin^{-1} \frac{r}{2}}{r}
$$

- Texture Mapping Style [1]: Tiling

- We want to create a brick wall with a brick pattern texture.
  - A brick pattern is very repetitive, we can use a small texture and “tile” it across the wall.

- Tiling allows you to scale repetitive textures to make texture elements just the right size.

Texture Mapping Style [2]: Stretching

- With non-repetitive textures, we have less flexibility.
  - Have to fill an arbitrarily large object with a texture of finite size.
  - Can’t tile, have to stretch.

- Example, creating a sky backdrop.

Texture Mapping Complex Geometry [1]

- Sometimes, reducing objects to primitives for texture mapping doesn’t achieve the right result.

- Consider a simple house shape as an example.
  - If we texture map it by our old method, we get discontinuities at some edges.

- Solution: Pretend object is a sphere and texture map using the sphere $(u, v)$ map.

Texture Mapping Complex Geometry [2]

- Intuitive approach: Place a bounding sphere around the complex object.
  - Find $z$, object space intersection with bounding sphere.

- Convert to $(u, v)$ coordinates.

- We actually don’t need a bounding sphere.
  - Once we have the intersection point with the object, we just treat it as though it were on the sphere. Same results, but be careful with radii.
Texture Mapping
Complex Geometry [3]

- What radius to use?
  - Compute the radius as the distance from the center of the complex object to the intersection point. Use that as the radius for the (u, v) mapping.

Complex Geometry [4]

- Results of spherical (u, v) mapping:
  - You can use cylindrical or planar mappings for complex objects as well.
  - Each has drawbacks:
    - Spherical warping at the "poles" of the object
    - Cylindrical discontinuities at the axes
    - Planes can detail must be preserved
  - Sphere mapped with spherical projection
  - Cylindrical mapped with planar projection

For best overall results, mapping techniques can be swapped

OpenGL Texturing [1]: Steps

- Create and specify a texture object
  - Create a texture object
  - Specify the texture image
  - Specify how texture has to be applied for each pixel
  - Enable texture mapping
  - Draw the textured polygons
    - Identify the active texture
    - Specify texture coordinates with vertices

OpenGL Texturing [2]: Specify 2-D Texture Object

- glTexImage2D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *data);
  - Specifies format and type used to specify the way the textures are stored
  - Specifies how OpenGL should store the data internally
  - width and height have to be powers of 2: you can use glClampNorm() to scale

OpenGL Texturing [3]: Specify How Texture Is Applied

- glTexParameteri(GLenum target, GLenum pname, TYPE param);
  - param can be: GL_TEXTURE_1D, GL_TEXTURE_2D, ...
  - GL_TEXTURE_WRAP_S, GL_CLAMP, GL_REPEAT
  - GL_TEXTURE_WRAP_T, GL_CLAMP, GL_REPEAT
  - GL_TEXTURE_MAG_FILTER, GL_NEAREST, GL_LINEAR
  - GL_TEXTURE_MIN_FILTER, GL_NEAREST, GL_LINEAR

OpenGL Texturing [4]: Enable Texture and Draw

- glEnable(GL_TEXTURE_2D);
  - Enable 2D texturing
- glTexCoord2f(GL_FLOAT u, GL_FLOAT v);
  - Specify texture coordinates per vertex (just as normals, color, etc.)
OpenGL Texturing [S]:
Create Texture Object

- glGenTextures(GLsizei n, GLuint* textureIDs);
  - Returns n currently unused texture IDs in textureIDs
  - Each texture ID is an integer greater than 0
- glBindTexture(GLenum target, GLuint textureID);
  - target is GL_TEXTURE_1D, GL_TEXTURE_2D, or GL_TEXTURE_3D
  - If textureID is being used for the first time a new texture object is created and assigned
  - ID = textureID becomes active

OpenGL Texturing [E]:
Putting It All Together

In initialization:
- glGenTextures(...);
- glBindTexture(...);
- glTexImage2D(...);
- glEnable(GL_TEXTURE_2D);

In display:
- Activate the texture defined in initialization
- glTexImage2D(...);
- glVertex3f(...);
- glTexCoord(...);
- glTexCoord(...);
- glEnd();

Preview:
Texturing with Blocks

Random placement of blocks
Neighboring blocks constrained by overlap
Minimum error boundary cut

Preview:
Mipmapping

Terminology
- Texture Mapping - Adding Detail, Raster Image, Color, etc. to CG Model
- Planar projection: apply flat texture to flat surface(s)
- Encoding volumes: cylinder, sphere, box
- Mapping methods
  - reflected ray
  - object normal
  - object center
  - intermediate surface normal
- Today: Mappings, OpenGL Texturing
  - Idea: define "texture" to simulate surface detail
  - Shadow, reflection/environment, transparency, bump, displacement
  - Other mappings: gloss, volumetric fog, skins, rainbows, water
- OpenGL texture mapping how-to

Summary
- Last Time: Texture Mapping Explained
  - Definitions and design principles
  - Enclosing volumes: cylinder, sphere, box
  - Mapping methods
    - reflected ray
    - object normal
    - object center
    - intermediate surface normal
  - Today: Mappings, OpenGL Texturing
    - Idea: define "texture" to simulate surface detail
    - Shadow, reflection/environment, transparency, bump, displacement
    - Other mappings: gloss, volumetric fog, skins, rainbows, water
    - OpenGL texture mapping how-to

Texturing with Blocks

Rand placement of blocks
Neighboring blocks constrained by overlap
Minimum error boundary cut

Mipmapping

The Fellowship of the Ring
© 2001 New Line Cinema