Lecture 12 of 41

Surface Detail 3 of 5: Mappings
OpenGL Textures

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

KSOL course pages: http://bit.ly/hGvXlH / http://bit.ly/eVizrE
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 2e – see http://bit.ly/ieUq45
Next class: Section 3.1, Eberly 2e
Brown CS123 slides on Polygons/Texture Mapping – http://bit.ly/h2VZn8
Wayback Machine archive of Brown CS123 slides: http://bit.ly/gAhJbh
Gröller & Jeschke slides on Texturing – http://bit.ly/dJFYq9

Lecture Outline

- Reading for Last Class: §2.6.3, 20.3 – 20.4, Eberly 2e
- Reading for Today: §20.5 – 20.13, Eberly 2e (Many Mappings)
- Reading for Next Class: §3.1, Eberly 2e
- Last Time: Texture Mapping Explained
  - Definitions and design principles
  - Enclosing volumes: cylinder, sphere, box
  - Mapping methods
    - reflected ray – bounce ray off object *O*
    - object normal – ray from face normal of object (polygon mesh)
    - object center – ray from center of object
    - intermediate surface normal – ray from inside of enclosing *S*
- Today: Mappings, OpenGL Texturing
  - Shadow, reflection/environment, transparency, bump, displacement
  - Other mappings: gloss, volumetric fog, skins, rainbows, water
  - OpenGL texture mapping how-to

Where We Are

Lecture	Topic	Primary Source(s)
0	Course Overview	Chapter 1, Eberly 2e
1	**CG Basics: Transformation Matrices; Lab 0**	**Sections (§) 2.1, 2.2**
2	Viewing 1: Overview, Projections	§ 2.2.3 – 2.2.4, 2.8
3	Viewing 2: Viewing Transformation	§ 2.3 esp. 2.3.4; FVFH slides
4	**Lab 1a: Flash & OpenGL Basics**	**Ch. 2, 16[1], Angel *Primer***
5	Viewing 3: Graphics Pipeline	§ 2.3 esp. 2.3.7; 2.6, 2.7
6	Scan Conversion 1: Lines, Midpoint Algorithm	§ 2.5.1, 3.1; FVFH slides
7	**Viewing 4: Clipping & Culling; Lab 1b**	**§ 2.3.5, 2.4, 3.1.3**
8	Scan Conversion 2: Polygons, Clipping Intro	§ 2.4, 2.5 esp. 2.5.4, 3.1.6
9	Surface Detail 1: Illumination & Shading	§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2
10	**Lab 2a: Direct3D / DirectX Intro**	**§ 2.7, Direct3D handout**
11	Surface Detail 2: Textures; OpenGL Shading	§ 2.6.3, 20.3 – 20.4, *Primer*
12	Surface Detail 3: Mappings; OpenGL Textures	§ 20.5 – 20.13
13	**Surface Detail 4: Pixel/Vertex Shad.; Lab 2b**	**§ 3.1**
14	Surface Detail 5: Direct3D Shading; OGLSL	§ 3.2 – 3.4, Direct3D handout
15	Demos 1: CGA, Fun; Scene Graphs: State	§ 4.1 – 4.3, **CGA handout**
16	**Lab 3a: Shading & Transparency**	**§ 2.6, 20.1, *Primer***
17	**Animation 1: Basics, Keyframes; HW/Exam**	**§ 5.1 – 5.2**
	Exam 1 review; Hour Exam 1 (evening)	**Chapters 1 – 4, 20**
18	**Scene Graphs: Rendering; Lab 3b: Shader**	**§ 4.4 – 4.7**
19	**Demos 2: SFX; Skinning, Morphing**	**§ 5.3 – 5.5, CGA handout**
20	Demos 3: Surfaces; B-reps/Volume Graphics	§ 10.4, 12.7, **Mesh handout**

Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review; and the green-shaded entry, that of the term project.

Green, **blue** and **red** letters denote **exam review**, **exam**, and **exam solution review** dates.

Review: OpenGL Shading (Overview)

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

See also: *OpenGL: A Primer, 3e* (Angel)
http://bit.ly/hVcVWN

- **Set Up Point Light Sources**
  - Directional light given by "position" vector
```
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```
  - Point source given by "position" point
```
GLfloat light_position[] = {-1.0, 1.0, -1.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```
- **Set Up Materials, Turn Lights On**
```
GLfloat mat_specular[]={0.0, 0.0, 0.0, 1.0};
GLfloat mat_diffuse[]={0.8, 0.6, 0.4, 1.0};
GLfloat mat_ambient[]={0.8, 0.6, 0.4, 1.0};
GLfloat mat_shininess={20.0};
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

glShadeModel(GL_SMOOTH); /*enable smooth shading */
glEnable(GL_LIGHTING); /* enable lighting */
glEnable(GL_LIGHT0);  /* enable light 0 */
```
- **Start Drawing (`glBegin` … `glEnd`)**

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University
http://bit.ly/g1J2nj

Acknowledgements: Many Mappings

Stefan Jeschke
Research Assistant
http://bit.ly/hUUM94

Eduard Gröller
Associate Professor
Director, Visualization Working Group
http://bit.ly/hUUM94

Institute of Computer Graphics and Algorithms
Technical University of Vienna

TECHNISCHE UNIVERSITÄT WIEN
Institut für Computergraphik und Algorithmen
Arbeitsbereich Computergraphik

Texturing material from slides © 2002 E. Gröller & S. Jeschke, Vienna University of Technology
http://bit.ly/dJFYq9

Mapping material from slides © 1995 – 2009 P. Hanrahan, Stanford University
http://bit.ly/hZfsjZ (CS 348B, Computer Graphics: Image Synthesis Techniques)

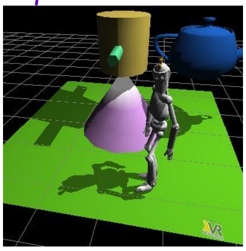
Overview of Mappings: Eberly 2e Chapter 20 Sections

- 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)

Babylon 5

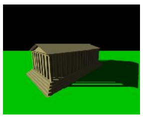
Shadow Mapping [1]: Basic Concept

- 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



Shadow Mapping © 2007 XVR Wiki
http://wiki.vrmedia.it/index.php?title=Shadow_Mapping

Without shadow map

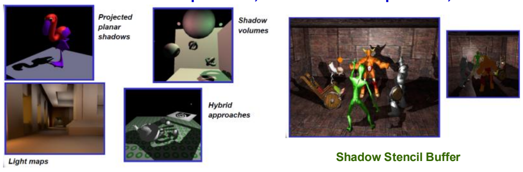


With shadow map

Shadow Mapping © 2005 Wikipedia
http://en.wikipedia.org/wiki/Shadow_mapping

Shadow Mapping [2]: Techniques

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



Shadow Stencil Buffer

- OpenGL Shadow Mapping Tutorials
  - Beginner/Intermediate (Baker, 2003): http://bit.ly/e1LA2N
  - Advanced (Octavian *et al.*, 2000): http://bit.ly/f1iRYB (old NeHe #27)

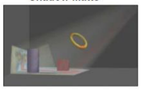
Adapted from "Shadow Mapping" © 2001 C. Everitt, nVidia
http://developer.nvidia.com/object/shadow_mapping.html

Shadow Mapping [3]: Advanced Methods & Research

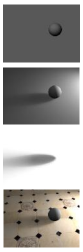
- Shadow Mattes (Hanrahan)

Projected Texture

Shadow Matte



```
UberLight( )
{
  Clip to near/far planes
  Clip to shape boundary
  foreach superelliptical blocker
      atten *= …
  foreach cookie texture
      atten *= …
  foreach slide texture
      color *= …
  foreach noise texture
      atten, color *= …
  foreach shadow map
      atten, color *= …
  Calculate intensity fall-off
  Calculate beam distribution
}
```



© 2010 M. Gryka
http://bit.ly/hJlfps

- Can Be Layered (See Maya 2011 Tutorial by Maciek Gryka)

Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University
http://bit.ly/hZfsjZ (CS 348B)

Reflection/Environment Mapping [1]: Basic Concept

- Reflection Maps (Special Type) ~ Environment Maps (General Case)
  - 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

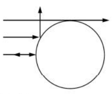
Ray Traced — Environment Map

Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University
http://bit.ly/hZfsjZ (CS 348B)

Reflection/Environment Mapping [2]: Techniques

- Gazing Ball (Mirrorball)

Photo © 2009 K. Turkowski



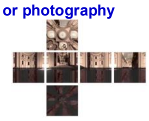
Note:
- Photograph of reflective ball
- Reflection indexed by normal
- Maps entire field of view to circle
- Resolution function of orientation; maximum head-on
- Fish eye camera lens similar

- 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

Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University
http://bit.ly/hZfsjZ (CS 348B)

Reflection/Environment Mapping [3]: Advanced Methods & Research

- How To Create Direction Maps
  - Latitude-Longitude (Map Projections) - paint
  - Gazing Ball - photograph reflective sphere
  - Fisheye Lens - standard (wide-angle) camera lens
  - Cubical Environment Map - rendering program or photography
    - Easy to produce
    - "Uniform" resolution
    - Simple texture coordinates calculation



- Old NeHe OpenGL Mapping Tutorials (2000)
  - #6 (texture map onto cube) – Beginner (Molofee): http://bit.ly/gKj2Nb
  - #23 (sphere) – Intermediate (Schmick & Molofee): http://bit.ly/e3Zb8h
- nVidia Tutorial: OpenGL Cube Map (1999): http://bit.ly/eJEdAM
- Issues: Non-Linear Mapping, Area Distortion, Converting Between Maps

Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University
http://bit.ly/hZfsjZ (CS 348B)

Transparency Mapping [1]: Basic Concept

- Transparency: One Term for Many Techniques

Tom Porter's Bowling Pin

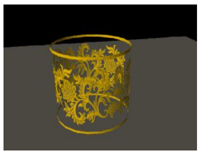
Source: RenderMan Companion, Pls. 12 & 13

- 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)

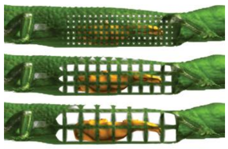
Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University
http://bit.ly/hZfsjZ (CS 348B)

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"
  - Wikipedia: http://bit.ly/ePpwoh (see also RGBA, http://bit.ly/ePpwoh)



Alpha blending: Lim (2010), http://bit.ly/6TsJrb
Goon Creative, Maya Transparency Tutorial



Screen door: Viola *et al.* (2004), http://bit.ly/dVEa7I
Technical University of Vienna, IEEE Vis 2004

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

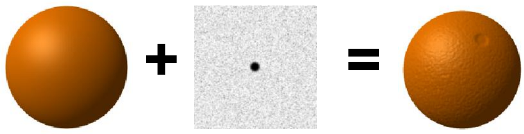
Transparency Mapping [3]: Advanced Methods & Research

- OpenGL Transparency How-To at OpenGL.org: http://bit.ly/hRaQgk
- Screen Door Transparency
  - Use `glPolygonStipple()`, `glEnable(GL_POLYGON_STIPPLE)`
  - See http://bit.ly/g1hQpJ
- Glass-Like Transparency using Alpha Blending
  - Use `glEnable(GL_BLEND)`, `glBlendFunc(…)`
  - See http://bit.ly/hs82Za

Viola *et al.* (2004), http://bit.ly/dVEa7I
Technical University of Vienna, IEEE Vis 2004

Bump Mapping [1]: Basic Concept

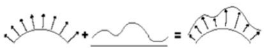
- Goal: Create Illusion of Textured Surface



Bump Mapping © 2010 Wikipedia
http://en.wikipedia.org/wiki/Bump_mapping

- 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



$\mathbf{P}(u,v)$

$\mathbf{S}(u,v)=\dfrac{\partial \mathbf{P}(u,v)}{\partial u} \qquad \mathbf{T}(u,v)=\dfrac{\partial \mathbf{P}(u,v)}{\partial v}$

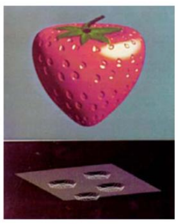
$\mathbf{N}(u,v)=\mathbf{S}\times\mathbf{T}$

- Displacement

$\mathbf{P}'(u,v)=\mathbf{P}(u,v)+h(u,v)\mathbf{N}(u,v)$

- Perturbed normal

$$\mathbf{N}'(u,v)=\mathbf{P}'_u\times\mathbf{P}'_v = \mathbf{N}+h_u(\mathbf{T}\times\mathbf{N})+h_v(\mathbf{S}\times\mathbf{N})$$

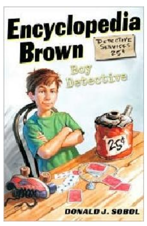


From Blinn 1976

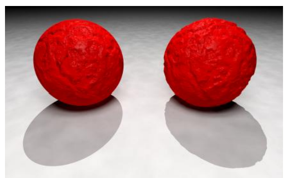
Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University
http://bit.ly/hZfsjZ (CS 348B)

Bump Mapping [3]: Advanced Methods & Research

- Bump Mapping Tutorial for OpenGL (Baker, 2003): http://bit.ly/fun4a5



Hey, wait a minute!

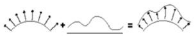


… what's wrong with the one on the left?

- 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?



$\mathbf{P}(u,v)$

$\mathbf{S}(u,v) = \frac{\partial \mathbf{P}(u,v)}{\partial u} \quad \mathbf{T}(u,v) = \frac{\partial \mathbf{P}(u,v)}{\partial v}$

$\mathbf{N}(u,v) = \mathbf{S} \times \mathbf{T}$

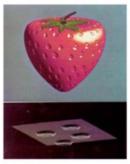
- Displacement

$\mathbf{P}'(u,v) = \mathbf{P}(u,v) + h(u,v)\mathbf{N}(u,v)$

- Perturbed normal

$\mathbf{N}'(u,v) = \mathbf{P}'_u \times \mathbf{P}'_v$

$= \mathbf{N} + h_u(\mathbf{T} \times \mathbf{N}) + h_v(\mathbf{S} \times \mathbf{N})$



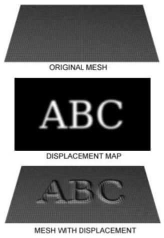
From Blinn 1976

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

Adapted from slides © 1995 – 2009 P. Hanrahan, Stanford University
http://bit.ly/hZfsjZ (CS 348B)

Displacement Mapping [2]: Techniques

- Displacement Map: Similar to Bump Map – Contains Delta Values



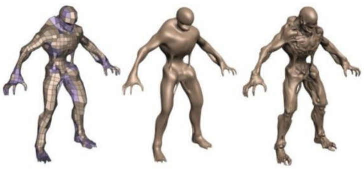
Displacement Mapping © 2005 Wikipedia
http://en.wikipedia.org/wiki/Displacement_mapping

Displacement Mapping: For Real!

- Displacement Mapping: Uses Open GL Shading Language (GLSL)
- Tutorial using GLSL (Guinot, 2006): http://bit.ly/dWXNya

Displacement Mapping [3]: Advanced Methods & Research

- When To Consider Using Displacement Mapping
  - Very "deep" texture effect: veins, ridges, *etc.*
  - Shadows expected



The "Imp" © Kenneth Scott, id Software 2008

The "Imp" © 2008 K. Scott, id Software
Bjorn3D, http://bit.ly/i78SiP

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/

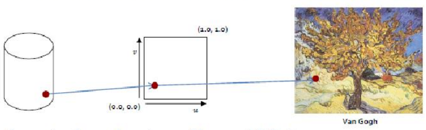
Texture Mapping

Beautification of Surfaces

Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

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:
    - Map a point on the geometry to a point on the unit square
    - Map the unit square point to point on the texture

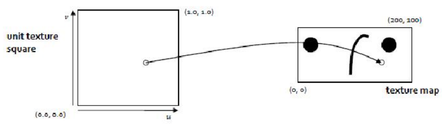


  - Second mapping is much easier, we'll present it first.

Adapted from slides © 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

Texture Mapping Technique [2]

- Mapping a point in the unit u, v square to a texture of arbitrary dimension:
  - 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 $(u * l, v * h)$.

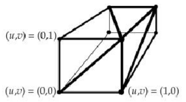


  - Above: (0.0, 0.0) -> (0, 0); (1.0, 1.0) -> (200, 100); (.7, .45) -> (140, 45)
  - Once we have coordinates for the texture, we just need to look up the color of the texture at these coordinates
  - Coordinates not always a discrete point on texture as they come from continuous space. May need to average neighboring texture pixels (i.e. filter)

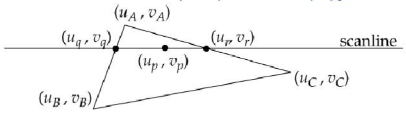
Adapted from slides © 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

Texture Mapping Technique [3]

- Texture mapping polygons
  - (u, v) 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



Interpolation Trick: Barycentric Coordinates

- Consider interpolating between two values along a line
  - Given two colors C_a and C_b, you can compute any value along the "line" between the two colors by evaluating:

$$C(t) = (1-t)C_a + tC_b \quad 0 \le t \le 1$$

  - This equation can be written as:

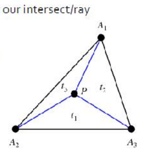
$$C(s,t) = sC_a + tC_b \qquad s + t = 1 \qquad s, t \ge 0$$

  - s and t are the Barycentric Coordinates of the line segment between C_a and C_b
  - 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

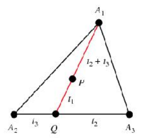
$$C(s,t,u) = sC_a + tC_b + uC_c \qquad s + t + u = 1 \qquad s, t, u \ge 0$$

Applying Barycentric Coordinates

- When you intersect a ray with a polyhedral object (not needed for our intersect/ray projects):
  - return the vertex data of the triangle intesected
  - return the Barycentric coordinates (t_1, t_2, t_3) of the intersection point
  - These coordinates can be used to interpolate between vertex colors, normals, texture coordinates, or other data
  - What weights do we hang on each vertex such that the triangle would be perfectly balanced on a pin at point P
  - Alternatively, think of a mobile suspended from P with 2 arms A_1Q and A_2A_3.



  - Compute Q as intersection of line through A_1 and P and line through A_2 and A_3
  - $t_3' = |Q - A_2|$
  - $t_2' = |Q - A_3|$
  - $t_1' = |P - Q|$
  - $(t_1, t_2, t_3) = (t_1', t_2', t_3')/(t_1 + t_2 + t_3)$

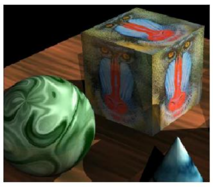


  - Another way of thinking about this is by triangle area. The weight at A_1 should be proportional to the area of the triangle P, A_2, A_3, and so on...

© 1999 – 2011 Wolfram Research
http://bit.ly/dGGdlc

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)

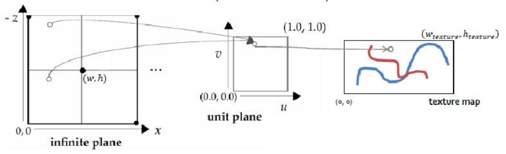


  - Can cause unwanted texture scaling
  - Texture filtering is an option in most graphics libraries
  - OpenGL allows you to choose filtering method. (GL_NEAREST, GL_LINEAR, etc...)

Texture Mapping Technique [5]

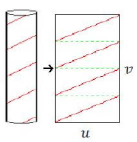
- Texture mapping large quads:
  - How to map a point on a very large quad to a point on the unit square?
  - Tiling: texture is repeated over and over across infinite plane
  - Given coordinates (x, y) of a point on an arbitrarily large quad that we want to tile with quads of size (w, h), the (u, v) coordinates on the unit square representing a texture with arbitrary dimensions are:

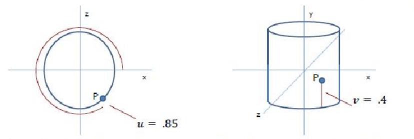
$$(u, v) = \left(\frac{(x \,\%\, w)}{w}, \frac{(y \,\%\, h)}{h}\right)$$



Texture Mapping Technique [6]

- How to texture map cylinders and cones:
  - Given a point P on the surface:
    - If it's on one of the caps, map as though the cap is a plane
    - If it's on the curved surface:
      - Use the position of the point around the perimeter to determine u
      - Use the height of the point to determine v

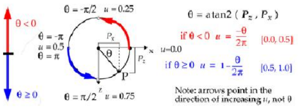




  - Mapping v is trivial, [-.5, .5] gets mapped to [0.0, 1.0] just by adding .5

Texture Mapping Technique [7]

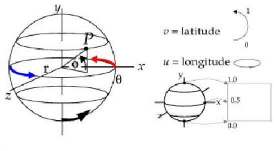
- Computing the u coordinate for cones and cylinders:
  - We need to map all the points on the perimeter of the object to [0, 1].
  - The easiest way is to say $u = \frac{\theta}{2\pi}$, but computing θ can be tricky



  - Standard atan function computes a result for θ but its always between 0 and π and it maps two positions on the perimeter to the same θ value.
    - Example: atan(1, 1) = atan(-1, -1) = $\frac{\pi}{2}$
  - atan2(x, y) yields values between $-\pi$ and π, but isn't continuous. See above diagram.

Texture Mapping Technique [8]

- Texture mapping spheres:
  - Find (u, v) coordinates for P
  - We compute u the same we do for cylinders and cones
  - If $v = 0$ or $v = 1$, there is a singularity. Set u to some predefined value. (.5 is good)
  - v is a function of the latitude of P



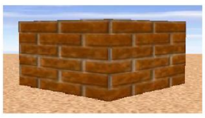
$$\phi = \sin^{-1}\frac{P_y}{r} \qquad -\frac{\pi}{2} \le \phi < \frac{\pi}{2} \qquad r = \text{radius}$$

$$v = \frac{\phi}{\pi} + .5$$

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


Texture


Without Tiling

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


With Tiling

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


Applied with stretching

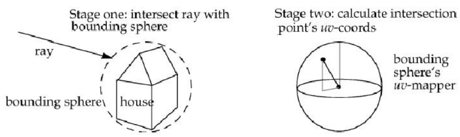
Texture Mapping Complex Geometry [1]

- 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 methods, 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 ray's object space intersection with bounding sphere
  - Convert to (u, v) coordinates

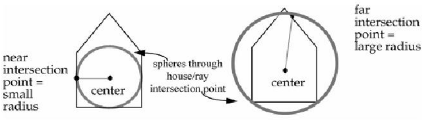


- We actually don't need a bounding sphere!
  - Once we have the intersection point with the object, we just treat it as though it were on the sphere. Same results, but be careful with radii.

37

Texture Mapping
Complex Geometry [3]

- When we treat the object intersection point as a point on a sphere, our "sphere" won't always have the same radius



- 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.

Adapted from slides © 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

38

Texture 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 caps
    - Planar: one dimension must be ignored

sphere mapped with spherical projection

sphere mapped with planar projection

- For best overall results, mapping techniques can be swapped

Adapted from slides © 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

39

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

Adapted from slides
© 2007 Jacobs, D. W., University of Maryland

40

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 *texels);***
  - Eg: ***glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 128, 128, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);***
  - ***format*** and ***type*** used to specify the way the texels are stored
  - ***internalFormat*** specifies how OpenGL should store the data internally
  - ***width*** and ***height*** have to be powers of 2; you can use ***gluScaleImage( )*** to scale

Adapted from slides
© 2007 Jacobs, D. W., University of Maryland

41

OpenGL Texturing [3]:
Specify How Texture Is Applied

- ***glTexParameter{if}(GLenum target, GLenum pname, TYPE param)***
- ***target*** can be: *GL_TEXTURE_1D, GL_TEXTURE_2D, …*

pname	***param***
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*

Adapted from slides
© 2007 Jacobs, D. W., University of Maryland

42

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).

Adapted from slides
© 2007 Jacobs, D. W., University of Maryland

OpenGL Texturing [5]: Create Texture Object

- *glGenTextures(GLsizei n, GLuint* textureIDs);*
  - Returns *n* currently unused texture ID 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 the ID = *textureID*
  - if *textureID* has been used before, the texture object with ID = *textureID* becomes active

Adapted from slides

OpenGL Texturing [6]: Putting It All Together

In initialization:

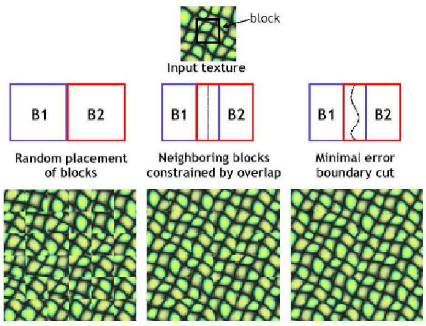
```
glGenTextures(...);
glBindTexture( ... );
glTexParameteri(...); glTexParameteri(...); ...
glTexImage2D(...);
glEnable(GL_TEXTURE_2D);
```

In display:

```
glBindTexture( ... ); // Activate the texture defined in initialization
glBegin(GL_TRIANGLES);
 glTexCoord2f(...); glVertex3f(...);
glTexCoord2f(...); glVertex3f(...);
glTexCoord2f(...); glVertex3f(...);
glEnd( );
```

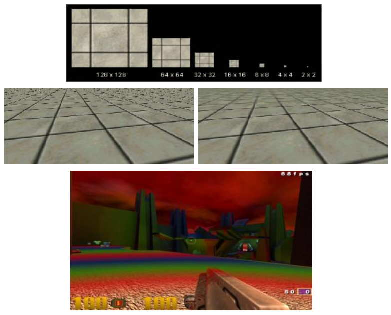
Adapted from slides

Preview: Texturing with Blocks



Adapted from slides

Preview: Mipmapping



Adapted from slides

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

The Lord of the Rings: The Fellowship of the Ring
© 2001 New Line Cinema

- 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

Terminology

- Texture Mapping - Adding Detail, Raster Image, Color, *etc.* to CG Model
  - Planar projection: apply flat texture to flat surface(s)
  - Enclosing volumes: cylinder, sphere, box
  - Mapping methods
    - reflected ray – bounce ray off object *O*
    - object normal – ray from face normal of object (polygon mesh)
    - object center – ray from center of object
    - intermediate surface normal – ray from inside of enclosing *S*
- Mappings: Apply Image or Simulated Surface Detail to Object
  - Shadow: cast planar projective shadows or calculate volume
  - Reflection/environment: take picture of scene from "inside" object
  - Transparency: take picture of scene "behind" object; refract
  - Bump: perturb color based on height map
  - Displacement: perturb face normals, recalculate lighting