Surface Detail 2 of 5: Textures
OpenGL Shading

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:
Next class: Sections 20.5 – 20.13, Eberly 2e
Lecture Outline

- Reading for Last Class: §2.7, Eberly 2e; Direct 3D Handout
- Reading for Next Class: §20.5 – 20.13, Eberly 2e
- Last Time: Intro to Illumination and Shading
  - Local vs. global models
  - Illumination (vertex shaders) vs. shading (fragment/pixel shaders)
  - Bidirectional reflectance distribution function (BRDF) \( \rho(p, \omega_i, \omega_o, \lambda) \)
  - Phong illumination equation: introduction to shading
- Texture Mapping Explained
  - Definitions
  - Design principles
- Texture Pipeline
- Using Simple Intermediate Surfaces (Cylinder, Sphere, Plane, Box)
- OpenGL Shading: Flat Shading, Smooth Shading (Gouraud)
### Where We Are

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Primary Source(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Course Overview</td>
<td>Chapter 1, Eberly 2nd</td>
</tr>
<tr>
<td>1</td>
<td>CG Basics: Transformation Matrices; Lab 0</td>
<td>Sections (§) 2.1, 2.2</td>
</tr>
<tr>
<td>2</td>
<td>Viewing 1: Overview, Projections</td>
<td>§ 2.3 – 2.2.4, 2.8</td>
</tr>
<tr>
<td>3</td>
<td>Viewing 2: Viewing Transformation</td>
<td>§ 2.3 esp. 2.3.4; FVFH slides</td>
</tr>
<tr>
<td>4</td>
<td>Lab 1a: Flash &amp; OpenGL Basics</td>
<td>Ch. 2, 1.6. Angel Primer</td>
</tr>
<tr>
<td>5</td>
<td>Viewing 3: Graphics Pipeline</td>
<td>§ 2.3 esp. 2.3.7, 2.6, 2.7</td>
</tr>
<tr>
<td>6</td>
<td>Scan Conversion 1: Lines, Midpoint Algorithm</td>
<td>§ 2.5.1, 3.1; FVFH slides</td>
</tr>
<tr>
<td>7</td>
<td>Viewing 4: Clipping &amp; Culling; Lab 1b</td>
<td>§ 2.3.5, 2.4, 3.1.3</td>
</tr>
<tr>
<td>8</td>
<td>Scan Conversion 2: Polygons, Clipping Intro</td>
<td>§ 2.4, 2.5 esp. 2.5.4, 3.1.6</td>
</tr>
<tr>
<td>9</td>
<td>Surface Detail 1: Illumination &amp; Shading</td>
<td>§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2</td>
</tr>
<tr>
<td>10</td>
<td>Lab 2a: Direct3D / DirectX Intro</td>
<td>§ 2.7, Direct3D handout</td>
</tr>
<tr>
<td>11</td>
<td>Surface Detail 2: Textures, Opacity, Shading</td>
<td>§ 2.6.4, 20.1, Primer</td>
</tr>
<tr>
<td>12</td>
<td>Surface Detail 3: Mappings, OpenGL Textures</td>
<td>§ 20.5 – 20.13</td>
</tr>
<tr>
<td>13</td>
<td>Surface Detail 4: Pixel/Vertex Shad.; Lab 2b</td>
<td>§ 3.1</td>
</tr>
<tr>
<td>14</td>
<td>Surface Detail 5: Direct3D Shading; OGLSL</td>
<td>§ 3.2 – 3.4; Direct3D handout</td>
</tr>
<tr>
<td>15</td>
<td>Demos 1: CGA, Fun; Scene Graphs: State</td>
<td>§ 4.1 – 4.3, CGA handout</td>
</tr>
<tr>
<td>16</td>
<td>Lab 3a: Shading &amp; Transparency</td>
<td>§ 2.6, 20.1, Primer</td>
</tr>
<tr>
<td>17</td>
<td>Animation 1: Basics, Keyframes; HW/Exam</td>
<td>§ 5.1 – 5.2</td>
</tr>
<tr>
<td>18</td>
<td>Exam review: Hour Exam 1 (evening)</td>
<td>Chapters 1 – 4, 20</td>
</tr>
<tr>
<td>19</td>
<td>Demo 2: SFX: Skinning, Morphing</td>
<td>§ 4.4 – 4.7</td>
</tr>
<tr>
<td>20</td>
<td>Demos 3: Surfaces, B-reps/Volume Graphics</td>
<td>§ 10.4, 12.7, Mesh handout.</td>
</tr>
</tbody>
</table>
Review:
Illumination & Shading

- **Lighting**, or *illumination*, is the process of computing the intensity and color of a sample point in a scene as seen by a viewer
  - lighting is a function of the geometry of the scene (including the model, lights and camera and their spatial relationships) and material properties
- **Shading** is the process of *interpolation* of color at points in-between those with known lighting or illumination, typically vertices of triangles or quads in a mesh
  - used in many real time graphics applications (e.g., games) since calculating illumination at a point is usually expensive
- On the GPU, lighting is calculated by a *vertex shader*, while shading is done by a fragment or *pixel shader*

Adapted from slides © 2010 van Dam *et al.*, Brown University
Review:
Phong Illumination Equation

- The full Phong model is a combination of the Lambertian and specular terms (summing over all the lights)

\[ I_\lambda = i_{d\lambda} k_{d\lambda} O_{d\lambda} + \sum_{\text{all lights}} f_{\text{att}} i_{d\lambda} [k_{d\lambda} (n \cdot L_m) O_{d\lambda} + k_{s\lambda} (R_m \cdot V)^a O_{s\lambda}] \]

- Subscript \( s \) represents specular (so \( k_s \) would be the specular coefficient)
- \( R_m \) is the reflected direction of the light ray about the surface normal
- \( f_{\text{att}} \) is the lighting attenuation function
  - function of distance from the light

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Review: Flat/Constant Shading

- We define a normal at each polygon (not at each vertex)
- **Lighting:** Evaluate the lighting equation at the center of each polygon using the associated normal
- **Shading:** Each sample point on the polygon is given the calculated lighting value

Adapted from slides © 2010 van Dam et al., Brown University
Review: Gouraud Shading

- We define a normal vector at each vertex
- **Lighting**: Evaluate the lighting equation at each vertex using the associated normal vector
- **Shading**: Each sample point’s color on the polygon is interpolated from the color values at the polygon’s vertices which were found in the lighting step

GL_SMOOTH

Adapted from slides © 2010 van Dam et al., Brown University
Review: Phong Shading

- Each vertex has an associated normal vector
- **Lighting:** Evaluate the lighting equation at each vertex using the associated normal vector
- **Shading:** For every sample point on the polygon we interpolate the normals at vertices of the polygon and compute the color using the lighting equation with the interpolated normal at each interior pixel

OpenGL implementation?
Stay tuned...
Texturing

Eduard Gröller
(today: Stefan Jeschke)

Institute of Computer Graphics and Algorithms
Vienna University of Technology

Adapted from slides
© 2002 Gröller, E. & Jeschke, S. Vienna Institute of Technology
Other Source Material on Texture Mapping

David W. Jacobs
Associate Professor, Computer Science Department and UMIACS, at the University of Maryland

Tobias Isenberg
Scientific Visualization and Computer Graphics Group
Department of Mathematics and Computing Science
University of Groningen

Formerly
Graphics Jungle Lab
Department of Computer Science
University of Calgary

CMSC 427 Computer Graphics
University of Maryland – College Park (UMD)
Fall 2007
Course: http://bit.ly/fXVA1A
Instructor: http://www.cs.umd.edu/~djacobs

CPSC 599.64/601.64 Computer Graphics
Fall 2005
Instructor: http://www.cs.rug.nl/~isenberg

• Computer Graphics II lecture by Stefan Schlechtweg, Department of Simulation and Graphics, Otto-von-Guericke University of Magdeburg, Germany
• CPSC 407 and CPSC 453 lectures by Brian Wyvill, Department of Computer Science, University of Calgary, Canada
Why Texturing?

- Idea: enhance visual appearance of plain surfaces by applying fine structured details

Adapted from slides © 2002 Gröller, E. & Jeschke, S. Vienna Institute of Technology
Introduction [1]: Motivation

- so far: detail through polygons & materials
- example: brick wall
- problem: many polygons & materials needed for detailed structures → inefficient for memory and processing
- new approach necessary: texture mapping
Introduction [2]: Properties and their Mappings

- several properties can be modified
  - color: diffuse component of surface
  - reflection: specular component of surface to simulate reflection (environment mapping)
  - normal vector: simulate 3D surface structure (bump mapping)
  - actual surface: raise/lower points to actually modify surface (displacement mapping)
  - transparency: make parts of a surface entirely or to a certain degree transparent
Concerning Textures

- Pattern of Intensity and color.
  - Can be generalized to 3D texture.
- How do we get them?
  - Take pictures.
  - Write a program (procedural textures).
  - Synthesize from examples
- How do we apply them? (Texture mapping)
  - Specify a mapping from texture to object.
  - Interpolate as needed.
  - This can be a challenging problem, but we’ll consider simpler version.
Without Textures
With Textures
Texture Image

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

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
Texture Mapping Overview [1]: Technique

- Texture mapping:
  - Implemented in hardware on every GPU
  - Simplest surface detail hack, dating back to the '60s GE flight simulator and its terrain generator

- Technique:
  - "Paste" photograph or bitmap (the texture, for example: a brick pattern, a wood grain pattern, a sky with clouds) on a surface to add detail without adding more polygons.
  - Map texture onto the surface get the surface color or alter the object's surface color
  - Think of texture map as stretchable contact paper

Adapted from slides © 2010 van Dam et al., Brown University
Texture Mapping Overview [2]: Motivation

- How do we add more detail to a model?
  - Add more detailed geometry; more, smaller triangles:
    - Pros: Responds realistically to lighting, other surface interaction
    - Cons: Difficult to generate, takes longer to render, takes more memory space
  - Map a texture to a model:
    - Pros: Can be stored once and reused, easily compressed to reduce size, rendered very quickly, very intuitive to use, especially useful on far-away objects, terrain, sky,...
    - Cons: Very crude approximation of real life. Texture mapped but otherwise unaltered surfaces still look smooth.
- What can you put in a texture map?
  - Diffuse, ambient, specular, or any kind of color
  - Specular exponents, transparency or reflectivity coefficients
  - Surface normal data (for bump mapping or normal mapping)
  - Projected reflections or shadows

Adapted from slides © 2010 van Dam et al., Brown University
Texture Mapping Overview [3]:

Mappings

- A function is a mapping
  - Takes any value in the domain as an input and outputs ("maps it to") one unique value in the co-domain.

- Mappings in "Intersect": linear transformations with matrices
  - Map screen space points (input) to camera space rays (output)
  - Map camera space rays into world space rays
  - Map world space rays into un-transformed object space for intersecting
  - Map intersection point normals to world space for lighting

- Mapping a texture:
  - Take points on the surface of an object (domain)
  - Return an entry in the texture (co-domain)

Adapted from slides © 2010 van Dam et al., Brown University
Texture Mapping How-To [1]:
Goals and Texture Elements (Texels)

- **texture**: typically 2D pixel image
- **texel**: pixel in a texture
- determines the appearance of a surface
- procedure to map the texture onto the surface needed
  - easy for single triangle
  - complex for arbitrary 3D surface
- **goal**: find easy way to do this mapping

Adapted from slides
© 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Texture Mapping How-To [2]: Adapting Polygons-to-Pixels Pipeline

- rendering pipeline slightly modified to use new texture mapping function
- algorithm: for each pixel to be rendered
  - find depicted surface point
  - find point in texture (texel) that corresponds to surface point
  - use texel color to modify the pixel’s shading
Texture Mapping How-To [3]:
Mapping Definition

- 2D texture: function that maps points on the \((u, v)\) plane to \((r, g, b)\) values:
  \((r, g, b) = c_{\text{tex}}(u, v)\)
- texture mapping function maps \((u, v)\) values to \((x, y, z)\) positions on objects:
  \((x, y, z) = F_{\text{map}}(u, v)\)
- we need to solve the inverse function to find \((u, v)\) values for a \((x, y, z)\) position:
  \((u, v) = F_{\text{map}}^{-1}(x, y, z)\)

\[
\begin{align*}
u &= s(x, y, z) \\
v &= t(x, y, z)
\end{align*}
\]
Texture Mapping How-To [4]: General Procedure

- general texture mapping pipeline:

\[(x, y) \rightarrow \text{determine surface position} \rightarrow (x, y, z) \rightarrow \text{find texture coordinates} \rightarrow (u,v) \rightarrow \text{find corresponding texel} \rightarrow (s,t) \rightarrow \text{possibly more processing} \rightarrow (s,t) \rightarrow \text{modify illumination} \rightarrow (r,g,b) \rightarrow \text{illumination} \rightarrow (r,g,b)\]

1. compute texture color for surface point
2. use to modify parameters in Phong illumination

Adapted from slides © 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Texture Mapping How-To [5]: Projective Textures, Functions

- goal: derive texture coordinates from 3D point
- \( P: \mathbb{R}^3 \rightarrow \mathbb{R}^2 \), so \( P(x, y, z) = (u, v) \)
- several typical possibilities
  - (manual) parameterization of the surface
  - use of inherent \((u, v)\) coordinates (e.g., freeform surfaces or primitive shapes)
  - two step technique

Adapted from slides
© 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Texture Mapping How-To [6]:
(Manual) Surface Parameterization

- simplest technique
- specification of texture coordinates during modeling
- \((u, v)\) coordinates specified for all vertices of a polygon
- interpolation between these values for points inside the polygon
  (e.g. barycentric interpolation for triangles)
Texture Mapping How-To [7]: Inherent \((u, v)\) Coordinates

- \((u, v)\) coordinates derived from parameter directions of surface patches (e.g., Bézier and spline patches)
- obvious \((u, v)\) coordinates derived for primitive shapes (e.g., boxes, spheres, cones, cylinders, etc.)
- used as defaults
Two-Step Approach [1]: Duality (Again)

• two steps:
  – mapping of 2D texture coordinates onto simple 3D surface (s-mapping)
  – mapping of the now 3D texture pattern onto complex object (o-mapping)

• in practice – inverse approach:
  – mapping of object point onto simple surface
    \( O: f(x_o, y_o, z_o) = (x_i, y_i, z_i) \)
  – mapping of surface point onto texture
    \( S: f(x_i, y_i, z_i) = (u, v) \)
Two-Step Approach [2]: Example – Cylindrical Mapping

- mapping onto cylinder surface given by height $h_0$ and angle $\theta_0$
  
  $$S : (\theta, h) \rightarrow (u, v) = \left( \frac{r}{c} (\theta - \theta_0), \frac{1}{d} (h - h_0) \right)$$

  using scaling factors $c$, $d$, and the radius $r$

- discontinuity along one line parallel to center axis

Adapted from slides © 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Two-Step Approach [3]: Example – Spherical Mapping

• mapping onto surface of a sphere given by spherical coordinates

\[ S : (r, \phi, \theta) \rightarrow (u, v) = \left( \frac{\theta}{2\pi}, \frac{(\pi/2) + \phi}{\pi} \right) \]

• no non-distorting mapping possible between plane and sphere surface

from R. Wolfe: Teaching Texture Mapping

Adapted from slides
© 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Two-Step Approach [4]: Example – Planar Mapping

- mapping onto planar surface given by position vector $\vec{v}_0$ and two vectors $\vec{s}$ and $\vec{t}$

\[ S : (x, y, z) \rightarrow (u, v) = \left( \frac{\vec{v} \cdot \vec{s}}{k}, \frac{\vec{v} \cdot \vec{t}}{k} \right) \]

- scaling factor $k$ and $\vec{v} = \vec{P}_1 - \vec{v}_0$ (describes point position w.r.t. the origin of the plane)
Two-Step Approach [5]: Example – Cuboid/Box Mapping

- enclosing box is usually axis-parallel bounding box of object
- six rectangles onto which the texture is mapped
- similar to planar mapping

from R. Wolfe: Teaching Texture Mapping

Adapted from slides
© 2005 Isenberg, T., University of Calgary (now at U. Groningen)
O Mapping [1]:
Object-to-Surface

- necessary for all named techniques
- four methods
  - reflected ray: trace a ray from viewer to object and reflect it onto the intermediate surface
  - object normal: intersection of normal vector of object with intermediate surface
  - object center: intersection of ray from object center through the object surface with the intermediate surface
  - normal of intermediate surface: trace this normal vector towards the object and determine intersection with it

Adapted from slides
© 2005 Isenberg, T., University of Calgary (now at U. Groningen)
O Mapping [2]: Illustrations

1. Reflected Ray
2. Object Normal
3. Object Center
4. Normal of Intermediate Surface

Adapted from slides © 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Correspondence Functions [1]:
Texture Coordinates

- projector functions yield \( (u, v) \) coordinates in texture parameter space
  \[ u = s(x, y, z) \]
  \[ v = t(x, y, z) \]
- typically values of \( u \) and \( v \) in \([0, 1]\)
Correspondence Functions [2]: Tiling, Mirroring, Clamping, Borders

- problem: what happens outside of [0, 1]?
- typical approaches
  - texture repetition (tiling) using modulo function
  - texture mirroring – better continuity at texture seams
  - clamping: repeat the last value of the texture edges for values outside of [0, 1]
  - border color: use a specified color for all non-defined values
Correspondence Functions [3]: Clamping and Borders Illustrated

Texture

Clamped texture applied to primitive

Texture

Texture with red border applied to primitive

from Microsoft Developer Network

Adapted from slides © 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Application of Texture Values: Combining Texturing and Lighting

- from an \((x, y, z)\) position we derived an \((r, g, b)\) color value from the texture, potentially with \(\alpha\) transparency value
- is typically used to modify illumination
- methods:
  - replace: surface color value is replaced with texture color
  - decal: \(\alpha\) blending of texture and original color
  - modulate: multiplication of original color value with texture color
The End...?

Done! ... well, almost

Adapted from slides
© 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Surface Detail: Imitating Complexity by Texturing Smooth Objects

Also possible: model very complex objects just by using simple textured geometry

Adapted from slides © 2002 Gröller, E. & Jeschke, S. Vienna Institute of Technology
More on Shading
(Surface Detail 2, 4, 5)

- Shading in OpenGL
  - Flat/constant: GL_CONSTANT
  - Gouraud: GL_SMOOTH
- Shading Languages
  - Renderman Shading Language (RSL) – http://bit.ly/g229q4
- Vertex vs. Pixel Shaders
- How to Write Shaders
Source Material on OpenGL Shading

15-462 Computer Graphics I
Lecture 8

Shading in OpenGL

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

February 14, 2002
Frank Pfenning
Carnegie Mellon University
http://www.cs.cmu.edu/~fp/courses/graphics/

Polygonal Shading
Light Source in OpenGL
Material Properties in OpenGL
Normal Vectors in OpenGL
Approximating a Sphere

[Angel 6.5-6.9]

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University.
http://bit.ly/g1J2nj
Shading in OpenGL [1]: Flat Shading

- Normal: given explicitly before vertex
  
  \[
  \text{glNormal3f}(nx, ny, nz); \\
  \text{glVertex3f}(x, y, z);
  \]

- Shading constant across polygon
- Single polygon: first vertex
- Triangle strip: Vertex n+2 for triangle n

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University.
http://bit.ly/g1J2nj
Shading in OpenGL [2]: Interpolative (aka Smooth), Gouraud

- **Interpolative Shading**
  - Enable with `glShadeModel(GL_SMOOTH);`
  - Calculate color at each vertex
  - Interpolate color in interior
  - Compute during scan conversion (rasterization)
  - Much better image (see Assignment 1)
  - More expensive to calculate

- **Gouraud Shading**
  - Special case of interpolative shading
  - How do we calculate vertex normals?
  - Gouraud: average all adjacent face normals
    \[ n = \frac{n_1 + n_2 + n_3 + n_4}{n_1 + n_2 + n_3 + n_4} \]
  - Requires knowledge about which faces share a vertex

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

http://bit.ly/g1J2nj
Shading in OpenGL [3]: Phong Shading

- Interpolate *normals* rather than colors
- Significantly more expensive
- Mostly done off-line (not supported in OpenGL)

... kind of

-WHH

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University.
http://bit.ly/g1J2nj
Shading in OpenGL [5]: Specifying & Enabling Light Sources

- **Enabling Light Sources**
  - Lighting in general must be enabled
    ```
    glEnable(GL_LIGHTING);
    ```
  - Each individual light must be enabled
    ```
    glEnable(GL_LIGHT0);
    ```
  - OpenGL supports at least 8 light sources

- **Specifying Point Light Source**
  - Use vectors {r, g, b, a} for light properties
  - Beware: light source will be transformed!
    ```
    GLfloat light_ambient[] = {0.2, 0.2, 0.2, 1.0};
    GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    ```

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University.
http://bit.ly/g1J2nj
Shading in OpenGL [6]:
Global Ambient Light

- Set ambient intensity for entire scene
  \[
  \text{GLfloat al[]} = \{0.2, 0.2, 0.2, 1.0\};
  \text{gLLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, al);};
  \]
- The above is default
- Also: local vs infinite viewer
  \[
  \text{gLLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE);};
  \]
- More expensive, but sometimes more accurate

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University.
http://bit.ly/g1J2nj
Shading in OpenGL [7]: Point Sources vs. Directional

- Directional Lights versus Point Lights

  - 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);
    ```

- Spotlights: Special Case of Point Lights

  - Create point source as before
  - Specify additional properties to create spotlight
    
    ```
    GLfloat sd[] = {-1.0, -1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, sd);
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
    glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 2.0);
    ```

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

http://bit.ly/g1J2nj
Shading in OpenGL [8]:
Example Material Properties

```c
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 */
```

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

- Last Time: Intro to Illumination and Shading
  - Local vs. global models
  - Illumination (vertex shaders) vs. shading (fragment/pixel shaders)
  - Phong illumination: derivation of ambient, diffuse, specular terms
  - Introduction to shading
- Texturing: Adding Detail, Raster Image, Color, etc. to CG Model
- Texture Pipeline
  - Part of polygons-to-pixels
  - Uses same spaces (coordinate systems) and more
- Using Simple Intermediate Surfaces (Cylinder, Sphere, Plane, Box)
- OpenGL Shading: Flat aka Constant, Interpolative (Specifically, Gouraud)
- Next: Patterns, Procedural Textures, Anisotropic Filtering
Texture Map / Texture Mapping
- Method of adding surface detail to CGI or 3-D model (Wikipedia)
- Kinds of surface detail
  - Detail: roughness, grain, bumps/dimples, etc.
  - Surface texture: finish, veneer, etc. (represented by raster image)
  - Color: monochrome, patterns, polychromatic

Coordinate Systems (Spaces)
- Model / Object: 3-D (x, y, z)
- World / Scene: 3-D (x, y, z)
- Camera / Eye: 3-D (u, v, n)
- Window / Screen: 2-D (u, v)
- Texture: 1-D, 2-D, or 3-D; (s, t) for 2-D

Texture Pipeline – End-to-End System for Calculating, Applying Textures

Gouraud Shading – Interpolative Shading with Color Interpolation