Lecture 11 of 41
Surface Detail 2 of 5: Textures
OpenGL Shading

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

Reading:
Today: Sections 2.6.3, 20.3 – 20.4, Eberly
Next class: Sections 20.5 – 20.13, Eberly

Textures
OpenGL Shading

Lecture Outline
- Reading for Last Class: §2.7, Eberly 2nd
- Reading for Today: §2.6.3, 20.3 – 20.4, Eberly 2nd
- Reading for Next Class: §20.5 – 20.13, Eberly 2nd
- 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

Review:
Illumination & Shading

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

Review:
Phong Illumination Equation

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

Review:
Flat/Constant Shading

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

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…

Source Material on Texturing: Gröller & Jeschke (Vienna Tech)

Texturing
Eduard Gröller (today: Stefan Jeschke)
Institute of Computer Graphics and Algorithms
Vienna University of Technology

Why Texturing?

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

Other Source Material on Texture Mapping

David W. Jacobs
Associate Professor, Computer Science Department and iLabCS
# Bi 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

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**

**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

Reused with permission.
Texture Mapping Overview [1]: Technique

- Texture mapping:
  - Implemented in hardware on every GPU
  - Smallest surface detail handled, dating back to the '80s: GE flight simulator and its terrain generator

- Technique:
  - “Paste” photograph or bitmap (the texture, for example: a bird pattern, a wood grain pattern, a sky with clouds) on a surface to add detail without adding more polygons.
  - Maps 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:
  - Prior: Render parts individually: lighting, other surface interaction
  - Now: Display to generate, takes longer to render, takes more memory space
  - Map a texture to a model:
    - Prior: Can be stored once and reused, avoids compressed to reduce size, mesh very similar to one, especially surface far away objects, maps, etc...
    - Cons: Very crude approximation of real life. Texture mapped but otherwise unvaried 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

Adapted from slides © 2005 Isenberg, T., University of Calgary (now at U. Groningen)

Texture Mapping How-To [3]: Mapping Definition

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

Adapted from slides © 2005 Isenberg, T., University of Calgary (now at U. Groningen)
Texture Mapping How-To [4]: General Procedure

- general texture mapping pipeline:
  - determine surface position
  - end texture coordinates
  - end corresponding color
  - possibly more processing
  - modify illumination
  1. compute texture color for surface point
  2. use to modify parameters in Phong illumination

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

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 new 3D texture pattern onto complex object (o-mapping)
- in practice – inverse approach:
  - mapping of object point onto simple surface
  - $f(x, y, z) = (x_1, y_1, z_1)$
  - mapping of surface point onto texture
  - $S: f(x, y, z) = (u, v)$

Two-Step Approach [2]: Example – Cylindrical Mapping

- mapping onto cylinder surface given by height $h_0$ and angle $\theta_0$
  - $S: (\theta, \phi) \rightarrow (x, y) = \left( \sqrt{r^2 - d^2} \sin(\phi) \, (h - h_0), \sqrt{r^2 - d^2} \cos(\phi) \right)$
  - using scaling factors $c, d$, and the radius $r$
- discontinuity along one line parallel to center axis
Two-Step Approach [3]:
Example – Spherical Mapping

- mapping onto surface of a sphere given by spherical coordinates
  \[ S(x, y, z) \rightarrow (u, v) = \left( \frac{\theta}{\pi}, \frac{\phi}{2\pi} \right) \]
- no non-distorting mapping possible between plane and sphere surface

Two-Step Approach [4]:
Example – Planar Mapping

- mapping onto planar surface given by position vector \( \vec{w} \) and two vectors \( \vec{s} \) and \( \vec{t} \)
  \[ S(x, y, z) \rightarrow (u, v) = \frac{\vec{v} \cdot \vec{w}}{\vec{k} \cdot \vec{w}} \]
- scaling factor \( \vec{k} \) and \( \vec{v} = P - \vec{P}_0 \) (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

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

O Mapping [2]:
Illustrations

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

Correspondence Functions [1]:
Texture Coordinates

- projector functions yield \((u, v)\) coordinates in texture parameter space
  \[ u = f(x, y, z) \quad v = g(x, y, z) \]
- typically values of \( u \) and \( v \) in \([0, 1]\)
- correspondence functions transform these into texel positions
- rotations, translations, scaling possible
- in most simple cases only scaling necessary
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

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

Surface Detail: Imitating Complexity by
Texturing Smooth Objects

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

More on Shading
(Surface Detail 2, 4, 5)

- Shading in OpenGL
  - FlatConstant: `GL_CONSTANT`
  - Gouraud: `GL_SMOOTH`
- Shading Languages
  - Renderman Shading Language (RSL) – [link]
  - OpenGL Shading Language (GLSL) – [link]
  - Microsoft High-Level Shading Language (HLSL) – [link]
  - nVidia Cg – [link]
- Vertex vs. Pixel Shaders
- How to Write Shaders
Source Material on OpenGL Shading

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.

Shading in OpenGL

February 14, 2003  
Frank Pfenning  
Carnegie Mellon University  

15-462 Computer Graphics I  
Lecture 8

Shading in OpenGL  

- Normal: given explicitly before vertex  
  - glNormal3f(nx, ny, nz);  
  - 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.

Shading in OpenGL

- Flat Shading

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

Shading in OpenGL

- Interpolative (aka Smooth), Gouraud

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

Shading in OpenGL

- Phong Shading

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

Shading in OpenGL

- Specifying & Enabling Light Sources

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

Shading in OpenGL

- Global Ambient Light

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University.
Shading in OpenGL [7]:
Point Sources vs. Directional Light

- Directional Lights:
  - Directional light given by "position" vector
  - Point source given by "position" point
- Spotlights: Special Case of Point Lights
  - Create point source as before
  - Specify additional properties to create spotlight

Shading in OpenGL [8]:
Example Material Properties

- Directional Lights versus Point Lights
- Spotlights: Special Case of Point Lights

Summary

- 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

Terminology

- Texture Map / Texture Mapping
- 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)
- Texture: 1-D, 2-D, or 3-D, (s, t) for 2-D
- Texture Pipelining – End-to-End System for Calculating, Applying Textures
- Gouraud Shading – Interpolative Shading with Color Interpolation