Chapter 7 - Light, Materials, Appearance

• Types of light in nature and in CG
• Shadows
• Using lights in CG
• Illumination models
• Textures and maps
• Procedural surface descriptions

Procedural Surface Descriptions

• Programming languages for surface descriptions
• Can influence various stages of the rendering pipeline
  – in particular: can implement textures and the Phong model
  – but also much more...
• Can describe real 3D structures
  – not just surface color
• State of the art in high end 3D graphics
  – e.g., RenderMan, used in PIXAR movies
  – also in OpenGL, DirectX
• Detailed implementation varies depending on the platform
  – in OpenGL: GL Shading Language (GLSL)
  – Used for vertex shaders and fragment shaders

Pipeline Architectures for 3D Graphics

• Pipelining improves throughput
  – At the same time product \( n \) undergoes procedure \( n \), and product \( n-1 \) undergoes procedure \( n-1 \)

• Pipelines in computer graphics target parallel processing

• Basic geometric pipeline:
  – Objects (vertices) to pixels
  – Intermediate product: fragment

• Fragment is a “potential pixel”
  – Information needed to update a pixel in frame buffer
Fixed-Function and Programmable Pipelines

• Traditional pipelining: “Fixed Function Pipeline”
  – Order and functionality of steps in the pipeline are fixed
  – Many options and parameters to adjust behavior of (hardware) implementation

• Modern pipelining architecture: “Programmable Pipelines”
  – General scheme of pipeline is fixed
  – Main steps are programmable by developer

• OpenGL Architecture:
  – Vertex processing and fragment processing are programmable (“Shader”)
  – Using GLSL; alternative languages exist, e.g. Nvidia Cg
OpenGL: Vertex and Fragment Shaders

- **Vertex Shader** can do the following:
  - transform the vertex position using the modelview and projection matrices
  - transform normals, and if required normalize them
  - generate and transform texture coordinates
  - lighting per vertex or compute values for lighting per pixel
  - color computation

- **Fragment Shader** can do the following:
  - compute colors, and texture coordinates per pixel
  - apply a texture
  - fog computation
  - compute normals if you want lighting per pixel

- This, and more details at: [http://www.lighthouse3d.com/opengl/glsI/](http://www.lighthouse3d.com/opengl/glsI/)
Chapter 8 - Shading and Rendering

- Local Illumination Models: Shading
- Global Illumination: Ray Tracing
- Global Illumination: Radiosity
- Non-Photorealistic Rendering

The 3D rendering pipeline (our version for this class)

- 3D models in model coordinates
- 3D models in world coordinates
- 2D Polygons in camera coordinates
- Pixels in image coordinates

- Scene graph
- Camera
- Animation, Interaction
- Rasterization
- Lights
Local Illumination: Shading

• Local illumination:
  – Light calculations are done locally without the global scene
  – No cast shadows
    (since those would be from other objects, hence global)
  – Object shadows are OK, only depend on the surface normal

• Simple idea: Loop over all polygons
• For each polygon:
  – Determine the pixels it occupies on the screen and their color
  – Draw using e.g., Z-buffer algorithm to get occlusion right

• Each polygon only considered once
• Some pixels considered multiple times
• More efficient: Scan-line algorithms
Scan-Line Algorithms in More Detail

- Using $u$ for horizontal pixel dimension, $v$ for vertical pixel dimension
- Edge Table (ET):
  - List of all non-horizontal edges, sorted by smaller $u$ value of end point
  - Refers to polygons to which the edge belongs
- Polygon Table (PT):
  - List of all polygons with plane equation parameters, color information and inside/outside flag (see earlier in the lecture)
- Active Edge Table (AET):
  - List of all edges crossing the current scan line

```plaintext
for $v = 0..V$ (all scan lines):
    Compute AET, reset flags in PT;
    for all crossings in AET:
        update flags;
        determine currently visible polygon $P$;
        set pixel color according to info for $P$ in PT;
```

- Each polygon considered only once
- Each pixel considered only once
Reminder: Phong’s Illumination Model

\[ I_o = I_{amb} + I_{diff} + I_{spec} = I_a k_a + I_i k_d (\vec{l} \cdot \vec{n}) + I_i k_s (\vec{r} \cdot \vec{v})^n \]

• Prerequisites for using the model:
  – Exact location on surface known
  – Light source(s) known
• Generalization to many light sources:
  – Summation of all diffuse and specular components created by all light sources
• Light colors easily covered by the model
• Do we really have to compute the formula for each pixel?
Flat Shading

- Determine one surface normal for each triangle
- Compute the color for this triangle
  - using e.g., the Phong illumination model
  - e.g. for the center point of the triangle
  - using the normal, camera and light positions
- Draw the entire triangle in this color

- Neighboring triangles will have different shades
- Visible „crease“ between triangles

- Cheapest and fastest form of shading
- Can be a wanted effect, e.g. with primitives
Mach Band Effect

• Flat Shading suffers from an optical illusion
  – Human visual system accentuates discontinuity at brightness boundary
  – Darker stripes appear to exist at dark side, and vice versa

![How the eye works](source: keithwiley.com)

Source: keithwiley.com
Gouraud Shading

• Determine normals for all mesh vertices
  – i.e., triangle now has 3 normals

• Compute colors at all vertices
  – using e.g., the Phong illumination model
  – using the 3 normals, camera and light positions

• Interpolate between these colors along the edges

• Interpolate also for the inner pixels of the triangle

• Neighboring triangles will have smooth transitions
  – If normals at a vertex are the same for all triangles using it

• Simplest form of smooth shading
  – Specular highlights only if they fall on a vertex by chance
Phong Shading

• Determine normals for all mesh vertices
• Interpolate between these normals along the edges
• Compute colors at all vertices
  – using e.g., the Phong illumination model
  – using the interpolated normal, camera and light positions

• Neighboring triangles will have smooth transitions
  – If normals at a vertex are the same for all triangles using it

• Has widely substituted Gouraud shading
  – Specular highlights in arbitrary positions
  – Have to compute Phong illumination model for every pixel
Shading in OpenGL

• Standard built-in shaders:
  – Flat shading
  – Gouraud shading (default)

• Shader set by `glShadeModel` function
  – `GL_FLAT`, `GL_SMOOTH`

• Additional global settings in `glLightModel*`
  – E.g. properties of ambient light
  – E.g. enforcement of separate treatment for specular and nonspecular colors
    (as needed when using texture patterns on specular surfaces)
Chapter 8 - Shading and Rendering

• Local Illumination Models: Shading
• Global Illumination: Ray Tracing
• Global Illumination: Radiosity
• Non-Photorealistic Rendering
Global illumination: Ray Tracing

• Global illumination:
  – Light calculations are done globally considering the entire scene
  – i.e. cast shadows are OK if properly calculated
  – Object shadows are OK anyway

• Ray casting:
  – From the eye, cast a ray through every screen pixel
  – Find the first polygon it intersects with
  – Determine its color at intersection and use for the pixel
  – Also solves occlusion (makes Z-Buffer unnecessary)

• Ray tracing: recursive ray casting
  – From intersection, follow reflected and refracted beams
  – up to a maximum recursion depth
  – Works with arbitrary geometric primitives

http://pclab.arch.ntua.gr/03postgra/mladenstamenico/ (probably not original)
source: Blender Gallery
Brainstorming: What Makes Ray Tracing Hard?
Optimizations for Ray Tracing

• Bounding volumes:
  – Instead of calculating intersection with individual objects, first calculate intersection with a volume containing several objects
  – Can decrease computation time to less than linear complexity (in number of existing objects)

• Adaptive recursion depth control
  – Maximum recursion limit is always necessary
  – Recursion should be stopped as soon as possible
  – E.g. stop if intensity goes below a threshold value

• Monte Carlo Methods
  – Improve complexity (cascading recursion = exponential)
  – Use one random ray for recursive tracing (instead of refracted/reflected rays)
  – Carry out multiple experiments (e.g. 100) and compute average values
Recent development: Real Time Ray Tracing

- Various optimizations presented over the last few years
- Real time ray tracing has become feasible
- Follow http://openrt.de/ (images from there)
Chapter 8 - Shading and Rendering

• Local Illumination Models: Shading
• Global Illumination: Ray Tracing
• Global Illumination: Radiosity
• Non-Photorealistic Rendering
Reminder: The rendering equation [Kajiya ‘86]

\[ I_o(x, \bar{\omega}) = I_e(x, \bar{\omega}) + \int_{\Omega} f_r(x, \bar{\omega}', \bar{\omega}) I_i(x, \bar{\omega}') (\bar{\omega}' \cdot \bar{n}) d\bar{\omega}' \]

- \( I_o \) = outgoing light
- \( I_e \) = emitted light
- Reflectance Function
- \( I_i \) = incoming light
- angle of incoming light

- Describes all flow of light in a scene in an abstract way
- doesn't describe some effects of light:
Global Illumination: Radiosity

- Simulation of energy flow in scene
- Can show „color bleeding“
  - blueish and reddish sides of boxes
- Naturally deals with area light sources
- Creates soft shadows
- Only uses diffuse reflection
  - does not produce specular highlights

http://www.webreference.com/3d/lesson46/
Radiosity Algorithm

• Divide all surfaces into small patches
• For each patch determine its initial energy
• Loop until close to energy equilibrium
  – Loop over all patches
    • determine energy exchange with every other patch
• „Radiosity solution“: energy for all patches
• Recompute if ______________________ changes


http://pclab.arch.ntua.gr/03postgra/mladenstamenico/ (probably not original)
Combinations

- Ray Tracing is adequate for reflecting and transparent surfaces
- Radiosity is adequate for the interaction between diffuse light sources
- What we want is a combination of the two!
  - This is non-trivial, a simple sequence of algorithms is not sufficient

Example for a state-of-the-art “combination” (more like another innovative approach): Photon Maps (Jensen 96)
  - First step:
    - Inverse ray tracing with accumulation of light energy
    - Photons are sent from light sources into scene, using Monte Carlo approach
    - Surfaces accumulate energy from various sources
  - Second step:
    - “Path tracing” (i.e. Monte Carlo based ray tracing) in optimized version
    (e.g. only small recursion depth)
Chapter 8 - Shading and Rendering

- Local Illumination Models: Shading
- Global Illumination: Ray Tracing
- Global Illumination: Radiosity
- Non-Photorealistic Rendering
Non-Photorealistic Rendering (NPR)

• Create graphics that look like drawings or paintings

• One method: stroke-based NPR
  – instead of grey shades, determine a stroke density and pattern
  – imitates pencil drawings or etchings (Kupferstich)

• Other methods: using image manipulation on rendered images
  – can in principle often be done in Photoshop

• Active field of research
  – many others

[http://www.katrinlang.de/npr/](http://www.katrinlang.de/npr/)