Course News

Assignment 2
- Due Monday, Feb 28

Homework 3
- Discussed in labs next week

Quiz 1
- Discussed in labs this week

Reading
- Chapter 9, 3

Out of Town Friday
- Anika will fill in for me

The Rendering Pipeline

Geometry Database → Model/View Transform → Perspective Transform → Clipping

→ Geometry Processing

→ Lighting

→ Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer

Types of Reflection

- Specular (a.k.a. mirror or regular) reflection causes light to propagate without scattering.

  - Diffuse reflection sends light in all directions with equal energy.

  - Mixed reflection is a weighted combination of specular and diffuse.

- Retro-reflection occurs when incident energy reflects in directions close to the incident direction, for a wide range of incident directions.

  gloss is the property of a material surface that involves mixed reflection and is responsible for the mirror-like appearance of rough surfaces.
Specular Reflection

Geometry of specular (mirror) reflection

\[ r = (n \cdot l)n - l + 2(n \cdot l)n \]

Intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.

Lambert’s “Law”

Glossy Materials – Empirical Approximation

Angular falloff

Phong Lighting

Most common lighting model in computer graphics (Phong Bui-Tuong, 1975)

\[ I_{\text{specular}} = k_r I_{\text{light}} \left( \cos \phi \right)^n \]

\( r \) : purely empirical constant, varies with surface
\( k_r \) : specular coefficient, highlight color
no physical basis, works ok in practice

Blinn-Phong model (Jim Blinn, 1977)

Variation with better physical interpretation

\[ I_{\text{out}}(x) = k_r (h \cdot n)^{1/3} \cdot I_{\text{in}}(x) \]

with \( h = (1 + v) / 2 \)
**Simple Light Sources**

*Types of light sources*
- Directional/parallel lights
  - E.g. sun
- Homogeneous vector (Homogeneous) point lights
  - Same intensity in all directions
  - Homogeneous point
- Spot lights
  - Limited set of directions
  - Point + direction + cutoff angle

**Light Sources**

*Area lights:*
- Light sources with a finite area
- Can be considered a continuum of point lights
- Not available in many rendering systems

**Light Source Falloff**

*Quadratic falloff (point- and spot lights)*
- Brightness of objects depends on power per unit area that hits the object
- The power per unit area for a point or spot light decreases quadratically with distance

\[
\text{Area } 4\pi r^2
\]

\[
\text{Area } 4\pi(2r)^2
\]

*Non-quadratic falloff*
- Many systems allow for other fallofs
- Allows for faking effect of area light sources
- OpenGL / graphics hardware
  - \( I_a \): intensity of light source
  - \( x \): object point
  - \( r \): distance of light from \( x \)
  - \( I_a(x) = \frac{1}{ar^2 + br + c} \cdot I_0 \)

**Light Sources**

*Ambient lights*
- No identifiable source or direction
- Hack for replacing true global illumination
  - (light bouncing off from other objects)

**Ambient Light Sources**

- Scene lit only with an ambient light source
**Directional Light Sources**
- Scene lit with directional and ambient light

**Point Light Sources**
- Scene lit with ambient and point light source

**Light Sources & Transformations**

**Geometry: positions and directions**
- Standard: world coordinate system
  - Effect: lights fixed wrt world geometry
  - Demo: [http://www.xmission.com/~nate/tutors.html](http://www.xmission.com/~nate/tutors.html)
- Alternative: camera coordinate system
  - Effect: lights attached to camera (car headlights)
  - Points and directions undergo normal model/view transformation

**Illumination calculations: camera coords**

**Lighting Review**

**Lighting models**
- Ambient
  - Normals don't matter
- Lambert/diffuse
  - Angle between surface normal and light
- Phong/specular
  - Surface normal, light, and viewpoint

**Lighting in OpenGL**

**Light source: amount of RGB light emitted**
- Value represents percentage of full intensity
  - E.g., (1.0, 0.5, 0.6)
- Every light source emits ambient, diffuse, and specular light

**Materials: amount of RGB light reflected**
- Value represents percentage reflected
  - E.g., (0.0, 1.0, 0.5)

**Interaction: multiply components**
- Red light (1,0,0) x green surface (0,1,0) = black (0,0,0)
**Lighting in Rendering Pipeline**

**Notes:**
- Lighting is applied to every vertex
  - i.e. the three vertices in a triangle
  - Per-vertex lighting
- Will later see how the interior points of the triangle obtain their color
  - This process is called shading
  - Will discuss in the context of scan conversion

---

**Scan Conversion**

Wolfgang Heidrich

---

**The Rendering Pipeline**

**Scan Conversion - Rasterization**

*Convert continuous rendering primitives into discrete fragments/pixels*

- Lines
  - Midpoint/Bresenham
- Triangles
  - Flood fill
  - Scanline
  - Implicit formulation
  - Interpolation

---

**Scan Conversion - Lines**
Scan Conversion of Lines - Digital Differential Analyzer

First Attempt:
```c
int dda( float xs, ys, xe, ye ) {
  // assume xs < xe, and slope m between 0 and 1
  float m = (ye-ys)/(xe-xs);
  float y = round( ys );
  for( int x=round(xs); x<=xe; x++ ) {
    drawPixel(x, round(y));
    y = y + m;
  }
}
```

Scan Conversion of Lines - Midpoint Algorithm

Moving horizontally along x direction
- Draw at current y value, or move up vertically to y+1?
  - Check if midpoint between two possible pixel centers above or below line

Candidates
- Top pixel: (x+1, y+1)
- Bottom pixel: (x+1, y)

Midpoint: (x+1, y+1)
Check if midpoint above or below line
- Below: top pixel
- Above: bottom pixel

Key idea behind Bresenham Alg.

Scan Conversion of Lines - Bresenham Algorithm (*63)

Use decision variable to generate purely integer algorithm
Explicit line equation:
\[ y = \frac{(y_e - y_s)(x - x_e) + y_s}{(x_e - x_s)} \]

Implicit version:
\[ L(x, y) \left( \frac{(y_e - y_s)}{(x_e - x_s)} - (y - y_s) \right) = 0 \]

In particular for specific x, y, we have
- \( L(x, y) > 0 \) if \((x,y)\) below the line, and
- \( L(x, y) < 0 \) if \((x,y)\) above the line
Scan Conversion of Lines

Bresenham Algorithm
- Decision variable: after drawing point (x,y) decide whether to draw
  - (x+1,y): case E (for "east")
  - (x+1,y+1): case NE (for "north-east")
- Check whether (x+1, y+1/2) is above or below line
  \[ d = L(x + 1, y + \frac{1}{2}) \]
- Point above line if and only if \( d < 0 \)

Scan Conversion of Lines

Bresenham Algorithm
- Problem: how to update \( d \)?
  - Case E (point above line, \( d < 0 \))
    \[ x = x + 1; \]
    \[ d = L(x+2, y+1/2) - d + (y_e - y)(x_e - x); \]
  - Case NE (point below line, \( d > 0 \))
    \[ x = x + 1; \]
    \[ y = y + 1; \]
    \[ d = L(x+2, y+3/2) - d + (y_e - y)(x_e - x) - 1; \]
- Initialization:
  \[ d = L(x+1, y+1/2) - (y_e - y)(x_e - x) - 1/2 \]

Scan Conversion of Lines

Bresenham( int xs, ys, xe, ye ) {
  int ye = ys;
  incrE = 2(ye - ys);
  incrNE = 2((ye - ye) - (xe-xs));
  for( int x = xs; x <= xe; x++ ) {
    drawPixel( x, y );
    if( d < 0 ) d += incrE;
    else { d += incrNE; y++; }
  }
}

Scan Conversion of Lines

Discussion
- Bresenham sets same pixels as DDA
- Intensity of line varies with its angle!

Discussion
- Good for hardware implementations (integer!)
- DDA
- May be faster for software (depends on system!)
- Floating point ops higher parallelized (pipelined)
  - E.g. RISC CPUs from MIPS, SUN
- No if statements in inner loop
- More efficient use of processor pipelining
Scan Conversion of Polygons

A General Algorithm
- Intersect each scanline with all edges
- Sort intersections in \( x \)
- Calculate parity to determine in/out
- Fill the ‘in’ pixels

Edge Walking
Past graphics hardware
- Exploit continuous \( L \) and \( R \) edges on trapezoid

\[
\text{scanTrapezoid}(x_L, x_R, y_L, y_R, \Delta x_L, \Delta x_R)
\]
**Edge Walking Triangles**

Split triangles into two regions with continuous left and right edges.

\[ \text{scanTrapezoid}(x_1, x_m, y_m, x_2, m_1, m_2) \]

\[ \text{scanTrapezoid}(x_1, x_2, y_1, y_2, m_1, m_2) \]

**Issues**

- Many applications have small triangles
- Setup cost is non-trivial
- Clipping triangles produces non-triangles
- This can be avoided through re-triangulation, as discussed

---

**Coming Up:**

**Friday**

- More scan conversion
- Lecture by Anika