CS 430/536
Computer Graphics I

Backface Culling,
Z-buffering,
Ray Casting, Ray Tracing
Radiosity
Week 8, Lecture 15

David Breen, William Regli and Maxim Peysakhov
Geometric and Intelligent Computing Laboratory
Department of Computer Science
Drexel University
http://gicl.cs.drexel.edu
Overview

- Rendering topics
  - Z-buffering
  - Back-Face Culling
  - Ray Tracing (Ray Casting)
  - Radiosity
Mesh/Faceted Model
Back-Face Culling

• Assumptions:
  – Object approximated as closed polyhedron
  – Polyhedron interior is not exposed by the front cutting plane
  – Eye-point not inside object
  – Polygons not facing the viewer called Back-Facing
  – Right-hand vertex ordering defines outward normal

• Back-Face Culling is a technique for eliminating polygons for these kinds of models

• On average eliminates half of the polygons
  – Could be done for performance reasons
Back-Face Culling

• After canonical transformation, examine normal $N_k(x_k, y_k, z_k)$ to the face.

• If $z_k < 0$, face is a Back-Face - don’t draw it
  ▲ More general test looks at $N_k \cdot V$
  ▲ $V$ - View vector

• The only test necessary for a single convex polyhedron
Back-Face Culled Wire-Frame
Z-buffering

- Z-buffering (depth-buffering) is a visible surface detection algorithm
- Implementable in hardware and software
- Requires data structure (z-buffer) in addition to frame buffer.
- Z-buffer stores values $[0 \ldots ZMAX]$ corresponding to depth of each point.
- If the point is closer than one in the buffers, it will replace the buffered values
Z-buffering
Z-buffering

for (y = 0; y < YMAX; y++)
    for (x = 0; x < XMAX; x++) {
        F[x][y] = BACKGROUND_VALUE;
        Z[x][y] = ZMIN;
    }

for (each polygon)
    for (each pixel in polygon’s projection) {
        pz = polygon’s z-value at pixel coordinates (x,y)
        if (pz > Z[x][y]) { /* New point is closer */
            Z[x][y] = pz;
            F[x][y] = polygon’s color at pixel coordinates (x,y)
        }
    }
Z-buffering w/ front/back clipping

for (y = 0; y < YMAX; y++)
    for (x = 0; x < XMAX; x++) {
        F[x][y] = BACKGROUND_VALUE;
        Z[x][y] = -1; /* Back value in NPC */
    }

for (each polygon)
    for (each pixel in polygon’s projection) {
        pz = polygon’s z-value at pixel coordinates (x,y)
        if (pz < FRONT && pz > Z[x][y]) { /* New point is behind front plane
            & closer than previous point */
            Z[x][y] = pz;
            F[x][y] = polygon’s color at pixel coordinates (x,y)
        }
    }
• We can simplify the calculation of $z$ by exploiting the fact that triangle is planar.
  – Interpolate $z$ values along the edges
  – Interpolate $z$ values along scan line
  – Special cases: horizontal edge & single vertex

\[
\begin{align*}
  z_a &= z_1 - (z_1 - z_2) \frac{y_1 - y_s}{y_1 - y_2} \\
  z_b &= z_1 - (z_1 - z_3) \frac{y_1 - y_s}{y_1 - y_3} \\
  z_p &= z_b - (z_b - z_a) \frac{x_b - x_p}{x_b - x_a}
\end{align*}
\]
Z Interpolation

- \( Z_a = z_1 + \left( \frac{|P_a - P_1|}{|P_2 - P_1|} \right)(z_2 - z_1) \)
- \( Z_b = z_1 + \left( \frac{|P_b - P_1|}{|P_3 - P_1|} \right)(z_3 - z_1) \)
- \( Z_p = z_a + \left( \frac{|P_p - P_a|}{|P_b - P_a|} \right)(z_b - z_a) \)
Back-Face Culled & Z-Buffered Wire-Frame
See the Difference
See the Difference
See the Difference
Depth Cueing

- Objects that are closer are brighter
- Objects farther away are darker
- Color = BaseColor*(z - far)/(near - far)

www.dm.unibo.it/~casciola

www.siggraph.org
Important Reminder

• Recall: near and far planes are transformed by $N_{\text{par}}$ and $N_{\text{per}}$

• Parallel projection
  – Near: $z = 0$
  – Far: $z = -1$

• Perspective projection
  – Near: $z = -\frac{\text{vrp}_{\text{z}} + F}{\text{vrp}_{\text{z}} + B}$
  – Far: $z = -1$
Ray Casting (Ray Tracing)

- Determines visible surfaces by tracing rays of light from the viewers eye to the objects in the world.
Ray Casting

- Determines visible surfaces by tracing rays of light from the viewers eye to the objects
- View plane is divided on the pixel grid
- The eye ray is fired from the center of projection through each pixel
Ray Casting

Select the center of projection and viewplane window
for (each scan line) {
  for (each pixel on the scan line) {
    Find ray from the center of projections through the pixel
    for (each object in the scene) {
      if (object is intersected and intersection is closest
         considered so far) {
        Record the intersection and the object name
      }
    }
    Set pixel color to that of the closest intersected object
  }
}

1994 Foley/VanDam/Finer/Huges/Phillips ICG
Computing Intersections

- Using parametric equation of the line:
  \[ x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0), \quad z = z_0 + t(z_1 - z_0) \]
- Simplifying for speed:
  \[ \Delta x = (x_1 - x_0), \quad \Delta y = (y_1 - y_0), \quad \Delta z = (z_1 - z_0) \]
- Resulting ray:
  \[ x = x_0 + t\Delta x, \quad y = y_0 + t\Delta y, \quad z = z_0 + t\Delta z \]
Computing Intersections with Sphere

- Sphere with radius $r$ and center $(a, b, c)$ is:
  \[(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2\]
- Intersection is found by substituting values for $(x, y, z)$
  \[(x_0 + t\Delta x - a)^2 + (y_0 + t\Delta y - b)^2 + (z_0 + t\Delta z - c)^2 = r^2\]
- With some straightforward algebraic transformations:
  \[(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] + (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0\]
- Equation is quadratic in terms of $t$
- Solve with quadratic formula
  - No real roots (square root is negative). No intersection
  - One real root. Ray grazes the sphere
  - Two real roots. There are two points of intersection
Ray-triangle Intersection

- Insert ray equation into barycentric expression of triangle
- $P(t) = a + \beta (b-a) + \gamma (c-a)$
- Intersection if $\beta + \gamma < 1$; $0 < \beta$ and $0 < \gamma$
Computing Intersections with Polygon

- First intersect ray with plane \[ Ax + By + Cz + D = 0 \]

- The substitution results in: \[ t = \frac{-Ax_0 + By_0 + Cz_0 + D}{(A\Delta x + B\Delta y + C\Delta z)} \]
- If denominator is 0, ray is parallel to the plane
- Project polygon and point orthographically on the coordinate plane
- Polygon containment test can be performed in 2D
Polygon Containment Test

• Jordan Curve Theorem:
  – Point is inside if, for any ray, there is an odd number of crossings
  – Otherwise it is outside
• Be careful with all the special cases
• Wide variety of other techniques exist
Why Trace Rays?

• More elegant
• Testbed for techniques:
  – modeling (reflectance, transport)
  – rendering (e.g. Monte Carlo)
  – texturing (e.g. hypertexture)
• Easiest photorealistic renderer to implement
Ray Tracing

- Extension of ray casting
- Idea: Continue to bounce the ray in the scene
- Shoot rays to light sources
- Simple and powerful
- Reflections, shadows, transparency and multiple light sources
- Can be used to produce highly realistic images
Ray Traced Image

Blessed State, by Ken Musgrave
To Ray Trace, We Need Refraction

- Snell’s Law: \( \eta_i \sin \theta_i = \eta_t \sin \theta_t \)
- Let \( \eta = \eta_i / \eta_t = \sin \theta_t / \sin \theta_i \)
- Let \( m = (\cos \theta_i \mathbf{n} - \mathbf{i}) / \sin \theta_i \)
- Then...

\[
t = \sin \theta_t m - \cos \theta_t n
= (\sin \theta_t / \sin \theta_i) (\cos \theta_i \mathbf{n} - \mathbf{i}) - \cos \theta_t \mathbf{n}
= (\eta \cos \theta_i - \cos \theta_t) \mathbf{n} - \eta \mathbf{i}
\]

\[
t = \left( \eta (\mathbf{n} \cdot \mathbf{i}) - \sqrt{1 - \eta^2 (1 - (\mathbf{n} \cdot \mathbf{i})^2)} \right) \mathbf{n} - \eta \mathbf{i}
\]

\[
\cos \theta_t = \sqrt{1 - \sin^2 \theta_t}
= \sqrt{1 - \eta^2 \sin^2 \theta_i}
\]

Can be negative for grazing angles when \( \eta > 1 \), say when going from glass to air, resulting in total internal reflection (no refraction).
Refraction in Action
More Refractions

Photon Mapping!

Justin Hansen
University of Utah

Henrik Wann Jensen
UC, San Diego
Efficiency Considerations

- Partition the bounding box on a regular grid with equal sized extents
- Associate each partition with the list of objects contained in it
Efficiency Considerations

- Ray only intersected with the objects contained in the partitions it passes through
- If partitions are examined in the order ray travels, we can stop after first intersection is found
- Need to check if intersection itself is in the partition
Advanced Rendering Techniques

Image produced by Horace Ip
Advanced Rendering Techniques

Image produced by Dmitriy Bespalov
Global Illumination with Radiosity

Slides from Fredo Durand and Barb Cutler
MIT
Why global illumination with radiosity?

- Simulate light inter-reflections (indirect lighting)
  - e.g. in a room much of the light is indirect

Museum simulation.
Program of Computer Graphics,
Cornell University.
50,000 patches.
Note indirect lighting from ceiling.
Direct illumination
Global Illumination
Why Radiosity?

- Sculpture by John Ferren
- *Diffuse* panels

**photograph:**

**diagram from above:**

All visible surfaces, white.
Radiosity vs. Ray Tracing

Original sculpture by John Ferren lit by daylight from behind.

Ray traced image. A standard ray tracer cannot simulate the interreflection of light between diffuse surfaces.

Image rendered with radiosity. Note color bleeding effects.
Radiosity vs. Ray Tracing

• Ray tracing is an *image-space* algorithm
  – If the camera is moved, we have to start over

• Radiosity is computed in *object-space*
  – View-independent (just don't move the light)
  – Can pre-compute complex lighting to allow interactive walkthroughs
Radiosity Overview

- Surfaces are assumed to be perfectly Lambertian (diffuse)
  - reflect incident light in all directions with equal intensity
- The scene is divided into a set of small areas, or patches.
- The radiosity, $B_i$, of patch $i$ is the total rate of energy leaving a surface. The radiosity over a patch is constant.
Diffuse reflection

The incident light is scattered equally in all directions.

This is characteristic of dull, matt surfaces such as paper, bricks, carpet, etc.
Lambert’s Law for Diffuse Reflection

$\begin{align*}
I &= I_L k_d \cos \theta \\
&= I_L k_d (\mathbf{n} \cdot \mathbf{L})
\end{align*}$

$I$: resulting intensity
$I_L$: light source intensity
$k_d$: (diffuse) surface reflectance coefficient
$k_d \in [0,1]$
$\theta$: angle between normal & light direction
Continuous Radiosity Equation

\[ B_{x'} = E_{x'} + \rho_{x'} \int G(x,x') \ V(x,x') \ B_x \]

- \( B_{x'} \): reflectivity
- \( E_{x'} \): incident energy
- \( \rho_{x'} \): reflectivity at point \( x' \)
- \( G(x,x') \): form factor
- \( V(x,x') \): visibility term
- \( B_x \): radiosity at point \( x \)

G: geometry term

V: visibility term

No analytical solution, even for simple configurations
Discrete Radiosity Equation

Discretize the scene into $n$ patches, over which the radiosity is constant

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j$$

- discrete representation
- iterative solution
- costly geometric/visibility calculations

reflectivity
form factor
The Radiosity Matrix

\[ B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j \]

\( n \) simultaneous equations with \( n \) unknown \( B_i \) values can be written in matrix form:

\[
\begin{bmatrix}
1 - \rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\
-\rho_2 F_{21} & 1 - \rho_2 F_{22} & & \\
& \ddots & \ddots & \\
-\rho_n F_{n1} & \cdots & \cdots & 1 - \rho_n F_{nn}
\end{bmatrix}
\begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_n
\end{bmatrix}
= 
\begin{bmatrix}
E_1 \\
E_2 \\
\vdots \\
E_n
\end{bmatrix}
\]

A solution yields a single radiosity value \( B_i \) for each patch in the environment, a view-independent solution.
Solving the Radiosity Matrix

The radiosity of a single patch $i$ is updated for each iteration by gathering radiosities from all other patches:

\[
\begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_i \\
\vdots \\
B_n
\end{bmatrix}
= 
\begin{bmatrix}
E_1 \\
E_2 \\
\vdots \\
E_i \\
\vdots \\
E_n
\end{bmatrix}
+ 
\begin{bmatrix}
\rho_i F_{i_1} \\
\rho_i F_{i_2} \\
\vdots \\
\rho_i F_{i_m}
\end{bmatrix}
\begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_i \\
\vdots \\
B_n
\end{bmatrix}
\]

This method is fundamentally a Gauss-Seidel relaxation.
Calculating the Form Factor $F_{ij}$

- $F_{ij} = \text{fraction of light energy leaving patch } j \text{ that arrives at patch } i$
- Takes account of both:
  - geometry (size, orientation & position)
  - visibility (are there any occluders?)
Calculating the Form Factor $F_{ij}$

- $F_{ij} = \frac{\text{fraction of light energy leaving patch } j \text{ that arrives at patch } i}{\theta_i \theta_j}$

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} V_{ij} \, dA_j \, dA_i$$
Form Factor from Ray Casting

- Cast $n$ rays between the two patches
  - $n$ is typically between 4 and 32
  - Compute visibility
  - Integrate the point-to-point form factor
- Permits the computation of the patch-to-patch form factor, as opposed to point-to-patch
Hemicube Algorithm

- A hemicube is constructed around the center of each patch
- Faces of the hemicube are divided into "pixels"
- Each patch is projected (rasterized) onto the faces of the hemicube
- Each pixel stores its pre-computed form factor. The form factor for a particular patch is just the sum of the pixels it overlaps
- Patch occlusions are handled similar to z-buffer rasterization
Stages in a Radiosity Solution

1. Input Geometry
2. Reflectance Properties
3. Camera Position & Orientation
4. Form Factor Calculation
5. Solve the Radiosity Matrix
6. Visualization (Rendering)
7. Radiosity Solution
8. Radiosity Image

Why so costly?
- Calculation & storage of $n^2$ form factors

- $> 90\%$
- $< 10\%$
- $\sim 0\%$
Progressive Refinement

- Goal: Provide frequent and timely updates to the user during computation
- Key Idea: Update the entire image at every iteration, rather than a single patch
- How? Instead of summing the light received by one patch, distribute the radiance of the patch with the most *undistributed radiance*. 
Progressive Refinement w/out Ambient Term
Progressive Refinement with Ambient Term
Results

30,000 patches.
Increasing the Accuracy of the Solution

What’s wrong with this picture?

- The quality of the image is a function of the size of the patches.
- The patches should be adaptively subdivided near shadow boundaries, and other areas with a high radiosity gradient.
- Compute a solution on a uniform initial mesh, then refine the mesh in areas that exceed some error tolerance.
Adaptive Subdivision of Patches

Coarse patch solution (145 patches)

Improved solution (1021 subpatches)

Adaptive subdivision (1306 subpatches)
Discontinuity Meshing

• Limits of umbra and penumbra
  – Captures nice shadow boundaries
  – Complex geometric computation
  – The mesh is getting complex
Discontinuity Meshing Comparison

With visibility skeleton & discontinuity meshing
10 minutes 23 seconds

[Gibson 96]
1 hour 57 minutes
Discontinuity Meshing
Results

Lightscape  http://www.lightscape.com
Results

Lightscape  http://www.lightscape.com
Results

Lightscape  http://www.lightscape.com
Programming Assignment 5

• Extend XPM to 60 different RGB colors
• Read 3 models and assign each a color
• Implement Z-buffer rendering
• Implement front & back cutting planes
  – Only render parts of models between planes
• Implement linear depth-cueing
  – Color = base_color*(z-far)/(near-far)
• Re-use and extend 2D polygon filling