Overview

- Rendering topics
  - Z-buffering
  - Back-Face Culling
  - Ray Tracing (Ray Casting)
  - Radiosity

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 \( \mathbf{N}_k (x_k, y_k, z_k) \) to the face.
- If \( z_k < 0 \), face is a Back-Face - don't draw it
  - More general test looks at \( \mathbf{N}_k \cdot \mathbf{V} \)
  - \( \mathbf{V} \) - View vector
- The only test necessary for a single convex polyhedron

Mesh/Faceted Model

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 .. ZMAX] corresponding to depth of each point.
- If the point is closer than one in the buffers, it will replace the buffered values

Z-interpolation

- 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

Z-buffering

\[
\text{for } (y = 0; y < \text{YMAX}; y++) \\
\text{for } (x = 0; x < \text{XMAX}; x++) \\
\quad \text{if } (pz > Z(x,y)) \{ \text{/* New point is closer */} \\
\quad \quad Z(x,y) = pz; \\
\quad \quad F(x,y) = \text{polygons color at pixel coordinates } (x,y) \\
\quad \} \\
\} \\
\}
\]

Z-buffering w/ front/back clipping

\[
\text{for } (y = 0; y < \text{YMAX}; y++) \\
\text{for } (x = 0; x < \text{XMAX}; x++) \\
\quad \text{if } (pz > \text{Z(x,y)}) \{ \text{/* New point is closer */} \\
\quad \quad Z(x,y) = pz; \\
\quad \quad F(x,y) = \text{polygons color at pixel coordinates } (x,y) \\
\quad \} \\
\]
Back-Face Culled & Z-Buffered Wire-Frame

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

Important Reminder
- Recall: near and far planes are transformed by $N_{par}$ and $N_{per}$
- Parallel projection
  - Near: $z = 0$
  - Far: $z = -1$
- Perspective projection
  - Near: $z = \frac{w_z - w_{z'}}{w_{z'}}$
  - Far: $z = -1$

See the Difference
Ray Casting (Ray Tracing)

- Determines visible surfaces by tracing rays of light from the viewer’s eye to the objects in the world.

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

Computing Intersections

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

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 \( (x,y,z) \)
  \( (x-a)^2 + (y-b)^2 + (z-c)^2 = r^2 \)
- With some straightforward algebraic transformations:
  \[ ((Ax)^2 + Ay)^2 + 2(Ax)(Ay - a) + (Ay - a)^2 \]
  \[ + (x-a)^2 + (y-b)^2 + (z-c)^2 - r^2 = 0 \]
- 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
  \[ P(t) = \alpha \beta \gamma (c-a) \]
- 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: 
  \[
  z = \frac{A_y y + B_y y + D}{-A_x x - B_x x - C_x}
  \]
- 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

To Ray Trace, We Need Refraction

- Snell’s Law: \( n_i \sin \theta_i = n_r \sin \theta_r \)
- Let \( n = n_i / n_r = \sin \theta_i / \sin \theta_r \)
- Let \( m = (\cos \theta_i \cdot n - i) / \sin \theta_i \)
- Then...
  \[
  t = \sin \theta_i \cdot m - \cos \theta_i \cdot n = (\sin \theta_i / \sin \theta_i) (\cos \theta_i \cdot n - i) - \cos \theta_i \cdot n = (n \cos \theta_i - \cos \theta_i) \cdot n - \eta_i \cdot i
  \]
  \[
  t = n \cos \theta_i - \frac{1 - n^2}{1 - n^2} \cdot i
  \]
- \( \cos \theta_i = \sqrt{1 - \sin^2 \theta_i} \)
- Can be negative for grazing angles when \( n^2 > 1 \), say when going from glass to air, resulting in total internal reflection (no refraction).
Refraction in Action

More Refractions

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
  - 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

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

Radiosity vs. Ray Tracing

Ray traced image. A standard ray tracer cannot simulate the interreflection of light between diffuse surfaces.
Image rendered with radiosity. Note color bleeding effects.

Original sculpture by John Ferren lit by daylight from behind.
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, mottled surfaces such as paper, bricks, carpet, etc.

Lambert’s Law for Diffuse Reflection

$\mathbf{I} = \mathbf{I}_0 \cos \theta$

$I$: resulting intensity
$I_0$: light source intensity
$k_d$: (diffuse) surface reflectance coefficient
$k_d \in [0,1]$
$\theta$: angle between normal & light direction

Continuous Radiosity Equation

$B_x = E_x + \rho \int G(x,x') V(x,x') B_{x'}$

$G$: geometry term
$V$: visibility term

No analytical solution, even for simple configurations

Discrete Radiosity Equation

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

$F_{ij}$: form factor

- discrete representation
- iterative solution
- costly geometric/visibility calculations
The Radiosity Matrix

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

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

\[
\begin{bmatrix}
1 - \rho F_{1i} & -\rho F_{12} & \cdots & -\rho F_{1n} \\
-\rho F_{21} & 1 - \rho F_{22} & \cdots & -\rho F_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
-\rho F_{ni} & -\rho F_{n2} & \cdots & 1 - \rho F_{nn}
\end{bmatrix}
\begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_n
\end{bmatrix}
= \begin{bmatrix}
E_1 \\
E_2 \\
\vdots \\
E_n
\end{bmatrix}
\]

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

Calculating the Form Factor \( F_{ij} \)

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

\[ F_{ij} = \frac{1}{A_i} \int_{A_j} \int_{A_i} \cos \theta_i \cos \theta_j \frac{V_{ij}}{\pi r^2} dA_i dA_j \]

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

This method is fundamentally a Gauss-Seidel relaxation
Stages in a Radiosity Solution

- Input Geometry
- Reflectance Properties
- Camera Position & Orientation

Form Factor Calculation
- Why so costly?
  - Calculation & storage of \( n^2 \) form factors

Solve the Radiosity Matrix
- < 10%

Radiosity Solution
- Visualization (Rendering)
- ~ 0%

Radiosity Image

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

Factory simulation. Program of Computer Graphics, Cornell University. 30,000 patches.

Increasing the Accuracy of the Solution

- 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.

What’s wrong with this picture?
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
1 hour 57 minutes

[ Gibson 96 ]

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