Overview

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

Mesh/Faceted Model

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

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

Normal for Triangle

- For plane
  \[ \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0 \]  
  \[ \mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) \]

- Normalize \( \mathbf{n} \)
- Note that right-hand rule determines outward face
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.

```
for (y = 0; y < YMAX; y++)
    for (x = 0; x < XMAX; x++) {
        F[x][y] = BACKGROUND_COLOR;
        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 with Front/Back Clipping

```
for (y = 0; y < YMAX; y++)
    for (x = 0; x < XMAX; x++) {
        Z[x][y] = -1.0; /* 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 */
            Z[x][y] = pz;
            F[x][y] = polygon's color at pixel coordinates (x,y)
        }
    }
```

Z Interpolation

- We can simplify the calculation of z by exploiting the fact that triangles are planar.
  - Interpolate z values along the edges.
  - Interpolate z values along scan line.
  - Special cases: horizontal edge, degenerate triangle & single vertex.
Z Interpolation

- \( z_a = z_1 + \frac{|P_a - P_1|}{|P_2 - P_1|}(z_2 - z_1) \)
- \( z_b = z_1 + \frac{|P_b - P_1|}{|P_3 - P_1|}(z_3 - z_1) \)
- \( z_p = z_a + \frac{|P_p - P_a|}{|P_b - P_a|}(z_b - z_a) \)

- \( P_1 = (x_1, y_1) \)
- \( P_2 = (x_2, y_2) \)
- \( P_3 = (x_3, y_3) \)

Back-Face Culled & Z-Buffered Wire-Frame

See the Difference

See the Difference

Depth Cueing

- Objects that are closer are brighter
- Objects farther away are darker
- Color = BaseColor \( * \frac{z - \text{far}}{\text{near} - \text{far}} \)
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

- Determines visible surfaces by tracing rays of light from the viewer's 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

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:
  \[ x = x_0 + \Delta x t, y = y_0 + \Delta y t, z = z_0 + \Delta z t \]

Relation of t to Intersection

We want the smallest positive t - call it \( t_1 \)

\[ t_1 = \begin{cases} \frac{\Delta x}{2A} - \frac{\sqrt{\Delta x^2 - 4AC}}{2A} & \text{if discriminant > 0} \\ \frac{\Delta x}{2A} + \frac{\sqrt{\Delta x^2 - 4AC}}{2A} & \text{if discriminant < 0} \end{cases} \]
Ray-triangle Intersection

- Insert ray equation into barycentric expression of triangle
- $P(u,v,w) = (1-u-v) P_1 + u P_2 + v P_3$
- Intersection if $0 \leq u \leq 1$, $0 \leq v \leq 1$, and $0 \leq w \leq 1$

Computing Intersections with Polygon

- First intersect ray with plane
- The substitution results in:
  $$Ax + By + Cz + D = 0$$
- 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

- Boat reflected in wavy water rendered in OpenGL using an environment map

Compiled from: Lecture notes of Dr. John C. Hart @ University of Illinois
To Ray Trace, We Need Refraction

- Snell’s Law: $\sin \theta_i = \frac{1}{\eta} \sin \theta_t$
- Let $\eta = \frac{\sin \theta_i}{\sin \theta_t}$
- Let $m = \frac{(\cos \theta_i \cdot \eta - \eta)}{\sin \theta_i}$

Then…

$$t = \sin \theta_t m - \cos \theta_t n$$

$$t = \left( \sin \theta_i \cdot \frac{1}{\eta} \cdot (\cos \theta_i \cdot \eta - \eta) \right) - \cos \theta_t n$$

$$t = \left( \cos \theta_i \cdot \frac{1}{\eta} \right) - \cos \theta_t n$$

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

Compiled from: Lecture notes of Dr. John C. Hart @ University of Illinois
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. 54,000 patches. Note indirect lighting from ceiling.

Global Illumination with Radiosity

Slides from Fredo Durand and Barb Cutler
MIT
Pat Hanrahan, Stanford U.
Radiosity Overview

- Classic radiosity = finite element method
- Assumptions
  - Diffuse reflectance
  - Usually polygonal surfaces
- Advantages
  - Soft shadows and indirect illumination
  - View-independent solutions
  - Precompute for a set of light sources
  - Useful for walkthroughs

Why Radiosity?

- Sculpture by John Ferren
- Diffuse panels

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.
**Continuous Radiosity Equation**

\[ B_c = E_c + \sum \{ G(x,x') V(x,x') B_c \} \]

Radiosity (B): sum of emission (E) and reflection.

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.

\[ B_i = E_i + \rho \sum_j F_{ij} B_j \]

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

---

**The Radiosity Matrix**

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

---

**Solve \( \{F\} \{B\} = \{E\} \)**

Direct methods: \( O(n^3) \)
- Gaussian elimination

Iterative methods: \( O(n^2) \)
- Energy conservation
- diagonally dominant → iteration converges

**Results**

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

---

*References*
- The image contains references to various sources which are not visible in the image.
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?)

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

Stages in a Radiosity Solution

Input Geometry → Form Factor Calculation → Radiosity Solution → Visualization (Rendering) → Radiosity Image

Reflectance Properties → Solve the Radiosity Matrix → Radiosity Solution

Camera Position & Orientation → Visualization (Rendering)

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

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
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.

Adaptive Subdivision of Patches

- Coarse patch solution (147 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

[Discontinuity Meshing](http://graphics.stanford.edu/courses/cs348b-02/lectures/lecture17/walk029.html)

[Results](http://www.lightscape.com)

[Discontinuity Mesh](http://graphics.stanford.edu/courses/cs348b-02/lectures/lecture17/walk030.html)
Results

Lightscape  http://www.lightscape.com

Results

Lightscape  http://www.lightscape.com