Global Rendering and Hidden Surface Removal

Computer Graphics 1, Fall 2003
Lecture 7
Chapter 6.10, (13), 5.6, 8.8 (3rd ed.)

Diffuse Light

Diffuse shading is based on the principle that more light hits the surface of an object where the angle between the surface normal and the vector from the intersection to the light is least.

Camera light source: no shadows

Shadows

Shadows are simple to compute in a ray tracer. A point is in shadow if a ray from that point to the light source intersects an object. In order to apply diffuse shading, the point of intersection of a ray and the object has already been calculated.

Distant light source: parallel rays
Point light source: see shadows

Spotlight: cone angle 30, high contrast

Spotlight - soft shadows

Effect of distance - no attenuation

Effect of distance - attenuation

Effect of distance – more attenuation

\[ \frac{1}{\text{distance}} \]

\[ \frac{1}{\text{(distance)}^3} \]
Global rendering

- Ray tracing
- Radiosity

Ray tracing

Intensity calculation:

$I = I_{\text{local}} + k_r I_{\text{reflected}} + k_t I_{\text{transmitted}}$

Advantages and disadvantages

- Ray tracing is attractive because
  - Shadows and reflections are easily incorporated
- Ray tracing is expensive because
  - the cost of computing ray-object intersections is very high
Reflections

Reflections are generated using ray tracing. In real life, the interaction of light with a reflective object occurs as follows:
1. a light source emits photons
2. photons collide with and bounce off objects (some energy is absorbed and we see colour)
3. photons travel into the eye

Ray Tracing Paths

Bounding Volumes

- Enclose groups of objects in a simple bounding volume (sphere or box)
- Test for intersection against bounding volume
  - if none, then ray does not intersect objects within
  - if intersection occurs, test against each object in turn

How do we subdivide space?

- One idea is octree partitioning
- We illustrate in 2D - known as quadtree

Octree

A tree structure emerges, with the nodes at the leaves of the tree containing (hopefully) a small number of objects, or empty

Ray Tracing through Octree

We trace ray from subregion to subregion, only doing intersection tests for the small number of objects in the subregion.

Find region corresponding to start point. Test ray for intersections with any objects.

If none, find next region - by calculating intersection with region boundaries. Advance a short way into next region - say to (x,y,z)

Find region which includes (x,y,z) and continue.
Finding Region Containing \((x,y,z)\)

We use octree itself to locate region. Starting at top, a simple test determines in which of eight (4 for quadtree) regions the point \((x,y,z)\) lies. Proceed down tree until leaf node reached.

Binary Space Partitioning

- An alternative to octrees is just to split into two at each step
  - separating plane typically chosen to divide space into regions of equal complexity
- Known as Binary Space Partitioning (BSP) trees

Octrees vs BSP trees

- Octrees:
  - good for scenes where density of objects varies widely
  - possible to have small objects in large regions
  - stepping from region to region slow because trees tend to be unbalanced
- BSP trees:
  - depth of tree small because tree balanced
  - memory costs lower
  - void areas smaller

Hidden surface removal/
Visible surface determination

The POVRAY Teapot

We do not wish to see things that are hidden behind other objects
Two main types of HSR

- Object space approach
  - Works on object level, comparing objects with each other
    - Painter's algorithm
    - Depth sort
- Image/pixel space approach
  - Works on pixel level, comparing pixel values
    - z-buffer algorithm
    - Ray-casting/ray-tracing

Painter's algorithm

- Works like a painter
  - Sort all objects according to their z position
  - Draw the farthest object first and the closest last (possibly on top of others)
  - Object based; compares objects with each other
  - Hard to implement in a pipeline fashion
  - Makes quite many errors
  - We draw unnecessary polygons
  - Sorting of almost sorted list is fast (remember bubble-sort basic courses!)

z-buffer algorithm

- fill z-buffer with infinite distance for all polygons
  - for each pixel
    - calculate z-value
      - if \( z(x,y) \) is closer than \( z\) - buffer \( (x,y) \)
      - draw pixel
      - \( z\) - buffer \( (x,y) = z(x,y) \)
  - end
  - end

- When calculating the \( z \)-value you must interpolate \( q = 1/z \) to get a correct result from a perspective view
- Image/pixel space
- Easy to implement in a pipeline structure (hardware)
- Always correct result

Culling

\[ y \cdot n > 0 \]

- COP

Back-face culling

- We will never see the “inside” of solid objects; there is no reason to draw the backsides of the face polygons
- We can check if we see the front side of a polygon by checking if the angle between the normal and the vector pointing towards the observer is smaller than 90 degrees
- After view transformation and perspective distortion, this simply becomes a check of the \( z \)-coordinate of the normal vector