Chapter 11
Global Illumination

Part 1
Ray Tracing

Reading:
Angel’s Interactive Computer Graphics (6th ed.)
Sections 11.1, 11.2, 11.3
Can pipeline graphics renders images like these?

Picture from http://www.graphics.cornell.edu/online/realistic/
Global Illumination

• Local (direct) lighting
  – The calculations depend on only the surface material properties, surface’s local geometry, and parameters of the light sources
  – Can be added to a fast pipeline graphics architecture
  – No shadow, indirect lighting (reflection, refraction)
Global Illumination

- Global (indirect) lighting
  - Evaluate the light reflected from a surface point taking into account all illumination that arrives at the point, including
    - Direct light, indirect light reflected or refracted from other object
    - Lighting is a recursive process and amounts to an integral equation, the rendering equation
Global Illumination

• Global (indirect) lighting
  – Ray tracing
    • Specular interreflection/refraction
    • Hard shadow
    • Ok for dynamic scenes
  – Radiosity
    • Diffuse interreflection between polygons
    • Soft shadow
    • Good for static scenes
      – Need to compute the form factor between polygon pairs and radiosity for each polygon
        » Costly, done in preprocessing
      – Render using pipeline
Forward vs. backward ray tracing

Starting at the light

Starting at the eye
Recursive ray tracing

• HSR ray tracing or ray casting (one pass)
  – Computes the visible point for each pixel and then determine the local illumination.

• Recursive ray tracing
  – Incorporate the following in a single framework
    • HSR
    • Global specular interaction by simulating indirect reflection and refraction of light
    • Hard-shadow calculation
Recursive ray tracing
Examples -1 by J. L. Lin
Recursive ray tracing
Examples - 2 by J. L. Lin
Recursive ray tracing

Examples -3 by J. L. Lin
Recursive ray tracing
Examples -4 by J. L. Lin
Recursive ray tracing
Another example
Recursive ray tracing

- Ray: a ray from eye through a pixel
- Recursive ray tracing
  - At each ray-object intersection, a reflected and a refracted ray are fired to trace the light ray propagation.
Recursive ray tracing
Ray termination

- **Termination of recursive ray tracing**
  - Always terminate if a ray intersects a diffuse surface.
  - Can terminate when a pre-set depth of trace has been reached.
  - Can terminate when the energy of the ray has dropped below a threshold.
Recursive ray tracing

Ray tracing tree
Recursive ray tracing

Intensity calculation

Pixel’s RGB

- is the intensity at the first ray-object intersection.
- at each ray-object intersection, the RGB at that point is obtained by

\[ I = I_{local} + k_{rg} I_{reflected} + k_{ig} I_{refracted} \]
Recursive ray tracing

Main program

Select a COP and window on the view plane
for each scan line in the window
  for each pixel p in the scan line
    begin
      TraceRay(COP, p-COP, 1, pixel_color)
    end
Recursive ray tracing

TraceRay()

TraceRay(start, direction, depth, var color: colors)
begin
  depth > max_depth then color:=black
  else begin
    Intersect ray with all objects and find
    intersection point p that is closest to start;
    if no intersection then color:=background_color;
    else begin
      local_color:=....
      Compute reflection direction r
      TraceRay(p, r, depth+1, reflected_color);
      Compute refraction direction t
      TraceRay(p, t, depth+1, refracted_color);
      Combine(color, local_color, k_local,
              reflected_color, kreflection,
              refracted_color, k_refraction);
    end
  end
end
Recursive ray tracing - Shadow

- Fire a shadow ray at a visible ray-object intersection.
  - If the shadow ray is interrupted, the point is under shadow

- Shadow effect
  - Assume the objects are wholly opaque
  - Reduce the local term to the ambient value
  - Produces sharp shadow boundary
Recursive ray tracing

Shadow using light buffer

- **Light buffer**
  - A cube of six faces with the point light source as center, and whose faces are partitioned to cells

- **Set-up**
  - All polygons are projected to each face using the light position as the COP
  - Each cell contains a list of polygons that can be seen from the light source through the cell
  - The polygon are sorted in ascending order of the depth
Recursive ray tracing -8
Shadow using light buffer -2

• Shadow testing
  – Find the cell that the shadow ray passes
  – Check to see if there is a polygon on the cell’s sorted list that occludes the point
• Polygon may partially cover the cell
Speed-Up Methods

• Adaptive depth control
• Bounding volume and hierarchical BV
• First-hit speed up
• Spatial scene coherence
  – Uniform partition
  – Octree
  – Binary Space Partition
  – KD tree
Speed-Up Methods

Bounding volume

- Bounding box aims to avoid unnecessary intersection computations.
- Efficiency gained depends on how small the void area is.
- Examples of bounding volume
  - Bounding sphere
  - Bounding box
    - Axis-aligned bounding box (AABB)
    - Oriented bounding box (OBB)
Speed-Up Methods
Bounding volume

(a) Sphere
(b) AABB
(c) OBB
Speed-Up Methods
Hierarchical bounding volume

- Hierarchical bounding volume provides an even better void area ratio, and a hierarchical testing steps.
  - Hierarchical bounding sphere
  - Hierarchical axis-aligned bounding box
  - Hierarchical oriented bounding box
Speed-Up Methods
Bounding volume hierarchy

- Objects in close spatial proximity form a cluster, and recursive clustering form a BV hierarchy.

- Difficulties
  - Potential clustering and depth of the hierarchy depend on the nature of the scene.
Speed-Up Methods
Spatial partition

• Scene space is subdivided into regions such that each region either is empty or contains only a small set of objects.
  – Only regions intersected by the ray are checked and in a near-to-far order. The checking stops once an intersection is found.
  – Rendering time can be relatively constant and independent on the scene complexity.
Speed-Up Methods
Uniform partition

- Easy to implement, Easier for next-region search
- Hard to set a good cell-size
- More space requirement
  - Poor results for inhomogeneous scenes

Speed-Up Methods
Uniform partition

• Grid traversal
  – Requires enumeration of voxel along ray, by, ex., 3D-DDA
  – Simple and hardware-friendly

Speed-Up Methods
Uniform partition

- Objects in multiple voxels
  - Store only references
  - Original mailbox uses ray-id stored with each triangle
    - Simple, but likely to destroy CPU caches
  - Use mailboxing to avoid multiple intersection computations
    - Store (ray, object)-tuple in small cache
    - Do not do intersect test if found in cache
Speed-Up Methods
Uniform partition

Case 1.

The object may be tested twice.

This can be prevented by caching the point of intersection and the ray's ID with the object when the object is first encountered.
**Speed-Up Methods**  
**Uniform partition**

**Case 2.**

The ray intersects object A when visiting voxel 1. But this intersection is further away along the ray in voxel 2 where object B has a closer intersection. So object B is erroneously missed.

Object A: tested only once in voxel 1 and the intersection need to be cached as a mailbox.

Note in voxel 1: The voxel tracking process can only be terminated when the closest intersection in contained within the current voxel.
Speed-Up Methods
Octree

- Adjust to object distribution in the scene
  - Start with bounding box of entire scene
  - Recursively subdivide voxels into 8 equal sub-voxels
  - Stopping criteria
    - Too many triangles & maximum depth Problems

- Each leaf of the octree
  - Contains no object, or
  - Contains objects less than a preset number, or
  - Contains more objects than the preset number but has size less than a preset size.
Speed-Up Methods

Octree
Speed-Up Methods
Tracing with octree

• Tracking a ray in octree
  – Starts from the cell first intersected by the ray
  – The ray is tested for intersections with objects lie over this cell
    • The first intersection encountered is the one
    • Tracks to the next cell if there is no intersections.
Speed-Up Methods
Find the next cell

• A naive way
  – Derive a point on the ray a short distance into the next cell and search the cell containing the point by a upwardly and downwardly traversal on the octree.
  – Glassner’s hash table
    • Derive a hash table contains an entry for each cell
    • Point’s coordinate as a code to access the table

• Fujimoto’s SEADS
  – SEADS structure + 3D-DDA algorithm
Speed-Up Methods
SEADS and 3D-DDA

- **SEADS** (Spatially Enumerated Auxiliary Data Structure)
  - A 2x2x2 uniform grid by dividing the cell into equally sized cells.
  - Enable very fast tracking using 3D-DDA

- Octree + SEADS
  - Standard octree with each leaf a SEADS

- Traversal in octree
  - Internal nodes: standard octree traversal
  - Within leaf (SEADS): 3D-DDA
Speed-Up Methods

BSP

- Binary space partitioning tree (BSP tree)
  - Even more flexible than octree
  - In 3-D, split with cutting plane
  - Useful for
    - Painter’s algorithm
      - Sorts the polygons into a back-to-front order
    - Ray tracing
      - uses axis-aligned cutting plane for easier next-cell search
Speed-Up Methods
BSP
Scene spatial coherence
BSP tree

- Each internal node representing a single partition plane that partition the occupied space into two
  - A leaf node represents a cell that contains a small set of objects (1 or 2)

- The partition planes are usually axis-aligned
  - this simplifies the next-cell search

- Position of the partition plane depends on the object’s distribution within the scene
Speed-Up Methods

K-D tree

• **k-dimensional tree (k-d tree)**
  – Is a binary tree, split at arbitrary interior point
  – Split one dimension at a time
  – Good for neighborhood search
RRT’s problems in quality

• Point light source vs. area light source
• Aliasing (due to point sampling)
  – Area sampling: Cone tracing, beam tracing, pencil tracing
  – Stochastic sampling: Distributed ray tracing
• Specular-to-specular interaction
  – Perfect reflection/refraction, No specular spreading
  – Beam tracing, pencil tracing
  – Distributed ray tracing
• Specular-to-diffuse interaction
  – Backward ray/beam tracing
  – Photon mapping
• Diffuse-to-diffuse interaction
  – Combining ray tracing and radiosity
Quality improvements

- Pinhole cameras
  - No depth of field; everything in focus
- Instantaneous shutters
  - No motion blur; objects complete still
Recursive ray tracing -13
Problems

• RRT can not handle
  – Specular highlight
  – Some light propagations
    • Diffuse-to-diffuse propagation
    • Specular-to-diffuse propagation
4 types of light transport

(1) diffuse to diffuse

(2) diffuse to specular

(3) specular to diffuse

(4) specular to specular
Why ray tracing looks fake
Hard shadow vs. soft shadow
Why ray tracing looks fake
Sharp mirrors vs. blurry reflection
Why ray tracing looks fake
Sharp mirrors vs. blurry reflection

Figure 7: Left: perfect specular Right: glossy reflection.

[Interactive DRT, Boulos et al.]
Why ray tracing looks fake
Without/with depth of field

---

Jason Waltman / jasonwaltman.com
Why ray tracing looks fake
Without/with depth of field

Figure 5: Left: pinhole camera. Right: thin lens camera.

[Interactive DRT, Boulos et al.]
Why ray tracing looks fake
Motion blur
Why ray tracing looks fake

Motion blur

Figure 8: Left: A moving sphere without motion blur. Right: With motion blur.

[Interactive DRT, Boulos et al.]
Distributed Ray-Tracing[Cook `84]

- **Stochastic sampling**
  - **Pixel:** Anti-Aliasing
  - **Lens:** Depth-of-Field
  - **BRDF:** Glossy reflections
  - **Illumination:** extended light sources
  - **Time:** motion-blur