ILLUMINATION MODELS/ALGORITHMS

Local illumination - Fast
Ignore real physics, approximate the look
Interaction of each object with light
• Compute on surface (light to viewer)

Global illumination - Slow
Physically based
Interactions between objects
• Compute on surface (light to viewer)

WHAT WAS NON-PHYSICAL IN LOCAL ILLUMINATION?
- Ignore real physics, approximate the look
- Interaction of each object with light

HOW SHOULD GLOBAL ILLUMINATION WORK?
- Simulate light
  • As it is emitted from light sources
  • As it bounces off objects / gets absorbed/refracted
  • As some of the rays hit the camera

ROUGH ROUTINES
- RayTrace(r,scene)
- obj = FirstIntersection(r,scene)
  if (no obj) return ...
  else
    refract_color = Black;
    return Shade(reflect_color, refract_color, obj);

WHEN TO STOP?
- Algorithm above does not terminate
- Termination Criteria
  • No intersection
  • Contribution of secondary ray attenuated below threshold 
    • each reflection/refraction attenuates ray
  • Maximal depth is reached

REFLECTION
- Mirror effects
  • Perfect specular reflection

SUB-ROUTINES
- ReflectRay(r, obj) – computes reflected ray (use obj normal at intersection)
- RefractRay(r, obj) – computes refracted ray
  • Note: ray is inside obj
- Shade(reflect_color, refract_color, obj) – compute illumination given three components

REFRAC TION
- Interface between transparent object and surrounding medium
  • E.g. glass/air boundary
- Light ray breaks (changes direction) based on refractive indices $n_1, n_2$

BASIC RAY-TRACING ALGORITHM

RayTrace(r, scene)
  obj = FirstIntersection(r, scene)
  if (no obj) return ...
  else
    refract_color = Black;
    return Shade(reflect_color, refract_color, obj);

HOW TO STOP?
- Algorithm above does not terminate
- Termination Criteria
  • No intersection
  • Contribution of secondary ray attenuated below threshold 
    • each reflection/refraction attenuates ray
  • Maximal depth is reached
SIMULATING SHADOWS
• Trace ray from each ray-object intersection point to light sources
• If the ray intersects an object in between \( \Rightarrow \) point is shadowed from the light source

RAY TRACING: IDEA
\[
\text{Light Source} \rightarrow \text{Image Plane} \rightarrow \text{Pixel}
\]

RAY-TRACING: PRACTICALITIES
• Generation of rays
• Intersection of rays with geometric primitives
• Geometric transformations
• Lighting and shading
• Speed: Reducing number of intersection tests
  • E.g. use BSP trees or other types of space partitioning

RAY-TRACING: GENERATION OF RAYS
• Distance to image plane: \( \Delta \)
• Image resolution (in pixels): \( N_x, N_y \)
• Image plane dimensions: \( l, t, b \)
• Pixel at position \( (i, j) \) \( \Rightarrow \) \( \Delta l = \frac{j - 0}{N_y} \)
• For convex polygons
  \[ P_{ij} = O + (i + 0.5) \frac{r - 1}{N_y} \Delta l - (j + 0.5) \frac{r - 1}{N_y} \Delta b \]

RAY-INTERSECTIONS WITH OTHER PRIMITIVES
• Implicit functions:
  • Spheres at arbitrary positions
    • Same thing
  • Conic sections (hyperboloids, ellipsoids, paraboloids, cones, cylinders)
    • Same thing (see quadratic function)
  • Higher order functions (e.g. tori and other quartic functions)
    • In principle the same
    • But root finding difficult
    • Numerical methods

RAY-INTERSECTIONS WITH OTHER PRIMITIVES
• Polygons:
  • First intersect ray with plane
  • Linear implicit function
  • Then test whether point is inside or outside of polygon (2D test)
    • For convex polygons
      • Sufﬁces to test whether point is on the right side of every boundary edge

RAY TRACING: TRANSFORMATIONS
• Ray Transformation:
  • For intersection test, it is only important that ray is in same coordinate system as object representation
  • Transform all rays into object coordinates
  • Translation camera point and ray direction by \( \text{matrix} \) of model/view matrix
  • Shading has to be done in world coordinates (where light sources are given)
  • Transform object space intersection point to world coordinates
  • Then have to keep both world and object-space ray

RAY TRACING: DIRECT ILLUMINATION
• Light sources:
  • For the moment: point and directional lights
  • More complex lights are possible
    • Area lights
    • Fluoresent
**Problem:** Too regular

- $V$ visibility of light (0 or 1)
- **Spatial Subdivision**
- Need to integrate

**Ray Tracing**

- So far:
  - GPU
  - Path Tracing

**In 2D:**

- Hierarchical space subdivision
- Surround (Bounding Volumes: $\Omega$)

**Goal**

- For implicit surfaces
- In reality:
  - Both for ray
  - Lighting model
  - Area lights produce soft shadows: Radiosity

**Lighting and shading**

- **Speed:** Reducing number of intersection tests

**Integration over light source**

- Area lights produce soft shadows:
  - In 2D:
  - Viewpoint: $V$ visibility of light (0 or 1)
  - $F$ can now be different for every ray!

**Ray Tracing: Direct Illumination**

- Local surface information (normal...)
  - For implicit surfaces $F(x,y,z) = 0$ (normal $n(x,y,z)$ is gradient of $F$)
  - Example:
    - $n(x,y,z) = \nabla F(x,y,z) = \frac{\partial F(x,y,z)}{\partial x} \hat{x} + \frac{\partial F(x,y,z)}{\partial y} \hat{y} + \frac{\partial F(x,y,z)}{\partial z} \hat{z}$

**Spatial Subdivision Data Structures**

- Goal: reduce number of intersection tests per ray
- Lots of different approaches:
  - (Hierarchical) bounding volumes
  - Octree, k-D tree, BSP tree

**Optimized Ray-Tracing**

- Basic algorithm is simple but very expensive
- Optimize...
  - Reduce number of ray-object intersection calculations
  - Parallelize
  - Methods
    - Bounding boxes
    - Spatial subdivision
      - Volume, intersect/evaluate
      - Tree pruning

**Numerical Integration**

- Regular grid of point lights
- **Problem:** Too regular
  - New: 4 hard shadows
  - Need LOTS of points to avoid this problem

**Solution:** Monte-Carlo!