High Quality Rendering

CS559 Lecture Notes
Not for Projection
Mike Gleicher, November 2007

Rendering

- How to make an image (from a model)
- How we "draw" with computers
- Generally, term implies trying to make high-quality images
- Two main categories of approaches
  - Object-Based
  - Light-Based
- Distinction is a little fuzzier than that

Object-Based Rendering

- What we've been doing so far
- Draw each object independently
- Primitives and abstractions provided by hardware
  - Triangles, texture mapping, multi-pass, local shading, …
- Hacks to make better and better visual effects
- Pros: abstractions efficient on hardware
- Cons: it's a hack!
  - Can't achieve all effects (without more hacks)
  - Not accurate model of real world

Light-Based Rendering

- Model what happens with light in scene
- Assume that we have a model of the scene
- Figure out how light interacts with it
- Allows for global effects
  - Or at least non-local ones
- Simulate what really happens
  - To varying degrees of realism in the model

How the real world "renders"

- Photons (Rays) from source
- Bounce paths
- Some lucky photons make it to the eye (very few)
- Not a practical strategy – too inefficient

Ray Tracing

- Technically "Backward Ray Tracing"
  - From eye to light
  - There are cases where we actually do forward tracing
  - Terminology is confusing – I prefer "from the eye"
- Idea:
  - For each pixel (image space algorithm)
  - Figure out where the photon would have come from
  - Note: get projective transform from ray fan out
  - Note: could use real model of lens to determine ray directions
  - Note: Sampling Issue

Ray Tracing Pieces

- 1. Figure out what ray is
- 2. Figure out what ray hits (ray-object intersection)
- 3. Figure out where it could have come from
  - Recursive – since outgoing ray must have come from someplace
- Ray / Object Intersection
  - Straightforward mathematical calculation (root finding)
  - Tricky part: making it go fast
  - Accelleration structures:
    - Simplified models (bounding spheres/boxes)
    - Hierarchical models (check rough stuff first)
    - Spatial Data structures

Where did the ray come from?

- We know: outgoing direction, local surface geometry
- Specular bounce
  - Good for mirror reflection
- Real surfaces are diffuse – could come from any direction
  - Distribution of likelihoods
  - Different surfaces distribute light differently
  - Really requires an integral over incoming ray directions
  - **B**i-directional **R**eflectance **D**istribution **F**unction
  - Ideal case: sample all incoming directions

Hack ray-tracing

- Try to model the rays most likely to be important
- Mirror reflection bounce (or refraction bounce)
- Direction towards light sources
  - Probably important since they are bright
  - Check to see if path is clear (hit something = shadow)
  - Use local lighting model
- What does this give us?
  - Everything from local lighting
  - Shadows
  - Reflections and Refractions

Shadows

- Shadows of point lights give hard edges
  - Even in the real world!
  - Quite ugly
- Soft shadows are nicer
- Come from area light sources
  - Umbra / penumbra
- How to achieve?
  - More than one ray towards the light source
  - Sampling of directions

Distributed Ray Tracing

- Need to sample a **distribution** of ray directions
- Some uses:
  - Soft shadows (distribution of directions towards area light)
  - Anti-Aliasing (distribution of rays within the pixel)
  - Imperfect reflections (distribution of outgoing rays)
  - Motion Blur (distribution of times)
  - Depth of Field
  - All indirect light directions (for diffuse surfaces)
    - Get inter-object color transfer
  - Notice how quickly this becomes impractical

What can we do with Ray-Tracing?

- Given infinite rays, just about anything
- Realistically:
  - Can be clever about how to sample
  - But ultimately, limited in number of rays
- To understand limits, need to talk about light paths

Light Path Calculus

- Lights
- Diffuse Reflections
- Specular Reflections
- Eyes

- All paths L (D | S)* E
  - Regular expressions

- (Backward) Ray tracing can do:
  - L (D|S) S* E
- What ray tracing can't do
  - Anything else

Global Illumination

- Real world lots of diffuse objects
- Inter-reflections are really important
- Indirect lighting common and good

- Not handled by ray tracing!

- Truly requires a global solution
  - Light bounces many times (potentially)
  - All objects can influence all others (potentially)

- Not readily solved with ray-tracing

Radiosity

- A special case of global illumination

- Assume all objects are diffuse
  - View direction doesn't matter
- Polygonal patches that are constant light "output"
- L D+ E paths

- Output of patch = sum(input)
- Input = for each other patch
  - Form factor (how much can it see)
  - Diffuse lighting
- Big linear system of equations (each patch depends on others)

Radiosity

- Requires little patches
  - Patches too big, they look silly
- Only diffuse lighting
- Doesn't handle curved surfaces well
- Hard to determine all the factors exactly
- Doesn't scale (big linear system)

Examples of other things

- Caustics
  - Light bounces off mirror (or through lens) to light a diffuse object
  - L S* D E

- Semi-Diffuse objects (real objects)

Advanced "Physically-Based" Rendering

- Smart Sampling – of all possible paths
- Bi-Directional Ray Tracing
  - Do some "from the light" and store energy on surfaces
  - Photon Maps
- Random sampling
  - Over ray directions (send out lots of rays, both directions)
  - Over possible paths
- Complex reflection distribution functions
  - Require complex sampling mechanisms to express
  - Integration over incoming (or outgoing) ray directions