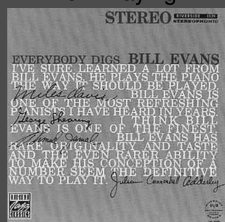
Now Playing:



Oleo
The Bill Evans Trio
from *Everybody Digs Bill Evans*
Released December 15, 1958

Movie: Lifted

Pixar, 2007

Beyond Raytracing: Radiosity

Rick Skarbez, Instructor
COMP 575
November 13, 2007

Announcements

- Programming Assignment 4 (Ray tracer) is out, due Tuesday 11/20 by 11:59pm
  - Any questions?

Final Project Notes

- You are required to submit a written proposal document for your project
  - Even if you met with me in person
  - If you have not done this, please do so immediately

Final Project Notes

- When should the project be due?
  - Can be as late as, say, Wednesday Dec. 12
    - Note that this is in the middle of finals
- Should we do class/public presentations?

Final Project Notes

- I would like to have a project "checkpoint"
  - You meet with me to show what you have so far / discuss any problems / plan your attack
  - This would be part of the project grade
  - I'd like to do this at a time where you still have time to make changes if necessary
    - I suggest November 30

Last Time

- Started talking about advanced ray tracing techniques
  - Acceleration data structures
    - To improve performance
  - Distributed ray tracing techniques
    - To achieve some neat visual effects

Must go faster...

- So what we've done so far works
  - We can render any scene just fine
    - At least, any scene that doesn't use additional effects
- But it's really, really slow:
  - Loop
    - For each pixel
      - For each object
        - For each light
  - Reflection/refraction/shadows make it even worse

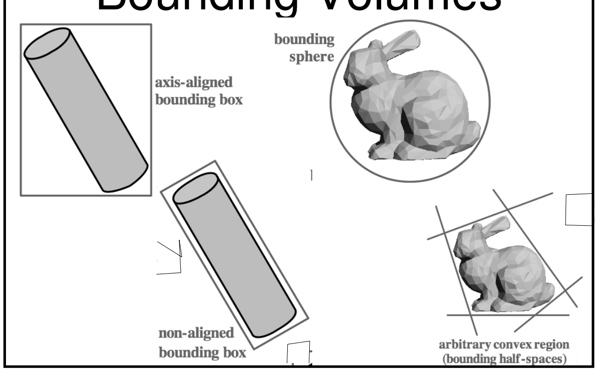
Accelerating Ray Tracing

- Reducing and/or simplifying intersection tests is the biggest "bang for your buck" in terms of performance
  - Computing ray intersections is slow
- 2 main ways
  - Use bounding volumes
  - Use a spatial data structure

Bounding Volumes

- Here's the idea:
  - Some shapes are harder to intersect with than others
    - Consider a box vs. a complex polygonal model
  - So, for every object, find the smallest simple object that encloses it
  - Test for intersection against the simple object
  - If there is one, only then do you test the original object

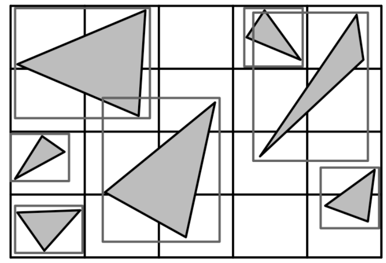
Bounding Volumes



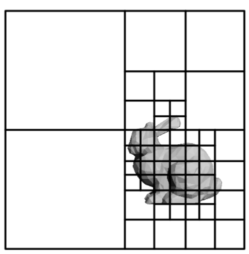
Spatial Data Structures

- The idea is that we only need to test if a ray hits an object if the ray passes through the region of space that the object is in
  - If a ray is going left, and the object is on the right, there is no need to test for intersection

Regular Grids



Adaptive Grids



BSP Trees

- Can create a BSP tree for your scene
- Then only have to test against objects that are on the "right" side of a split plane

Acceleration Review Over

- Any questions?

Distributed Ray Tracing

- We started talking about this last time
- Cook argues that classical ray tracing (*i.e.* everything we've done so far) only represents <u>sharp</u> pheno
  - Unrealistic sharp shadow infinite depth of focus, etc
- How can we do better?

Distributed Ray Tracing

Cook et al., 1984

- So what are some of the effects we can expect this way?
  - Antialiasing
    - Distribute rays across each pixel
  - Glossy reflections
    - Distribute multiple reflection rays instead of just one

Stochastic Ray Tracing

- So what are some of the effects we can expect this way? (cont'd)
  - Soft shadows
    - Distribute multiple rays to an area light source
  - Depth of field
    - Distribute rays across a lens
  - Motion blur
    - Distribute rays <u>over time</u>

Glossy Reflections

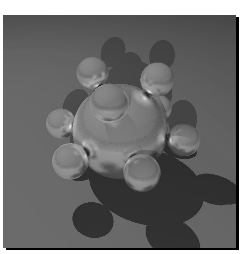
- Shooting a single reflection ray simulates perfect reflection
  - i.e. a mirror
- Many real surfaces are reflective, but not mirror-like
  - i.e. many metals
  - This is called *gloss*

Glossy Reflections

- To get glossy reflections, don't just shoot one ray
  - Shoot multiple rays, and perturb them slightly
    - This simulates taking the integral over a solid angle

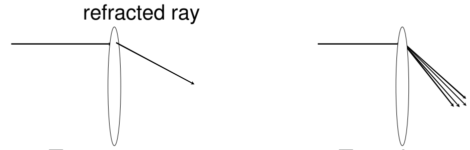
Mirror　　　　Glossy

Glossy Reflection Examples



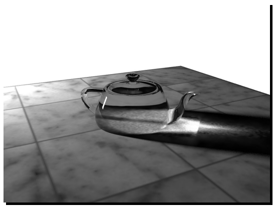
Translucency

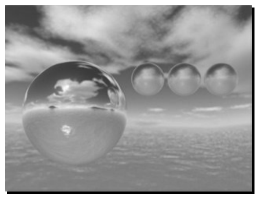
- Translucency is sort of the dual of glossy reflection
  - Instead of distributing rays around the reflection ray, distribute them around the refracted ray



Transparent　　　　Translucent

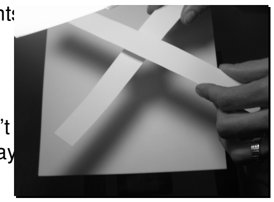
Translucency Examples





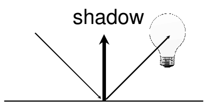
Soft Shadows

- In most graphics applications (and in our ray tracer so far), we've assumed point light sources
  - In the real world, lights have area
  - This leads to soft shadows in the real world, which we can't yet simulate in our ray tracer

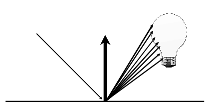


Soft Shadows

- To get soft shadows, don't just shoot one ray
  - Shoot multiple rays distributed across the surface of the light
  - Sum their contributions to find the amount of shadow

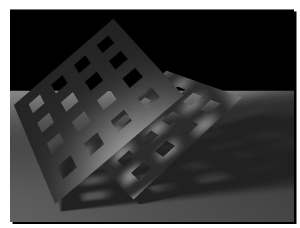


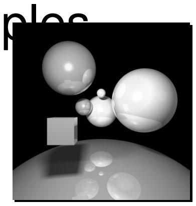
Hard Shadows

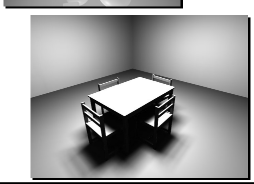


Soft Shadows

Soft Shadow Examples

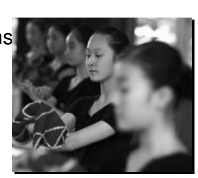






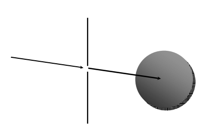
Depth of Field

- Our ray tracer up to this point simulates a pinhole camera
  - Real world cameras have lens, differing aperture sizes, differing exposure times, etc.
  - We're going to focus (no pun intended) on depth of field

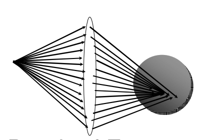


Depth of Field

- To get depth of field, generate multiple rays for each pixel
  - Distribute them across the surface of the lens

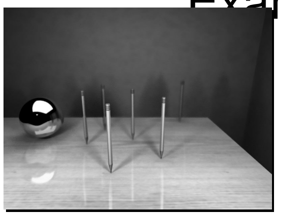


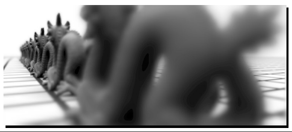
Perfect Focus



Depth of Focus

Depth of Field Examples





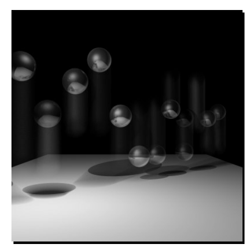
Motion Blur

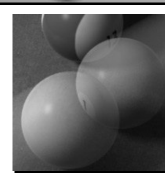
- Motion blur in the real world happens when objects are moving while the camera shutter is open
  - Effectively, the same point on the object is seen along multiple ra
    the camera

Motion Blur

- To get motion blur, you need to distribute your rays over time
  - As an object moves, it will get hit by different camera rays
  - Moving objects get averaged with the environment
- What happens to stationary objects?
  - Additional rays can still be used for antialiasing, depth of field effects, etc.

Motion Blur Examples





Distributed Ray Tracing Review

- We introduced the concept of distributed ray tracing
  - <u>NOTE</u>: Don't confuse this with the way the word "distributed" is commonly used in CS
- Showed some examples of how it can be used to generate more realistic images
- Basic idea: Replace a single ray with many

Done with (Standard) Raytracing

- So that's all we have to say about standard (one-way) ray tracing
- <u>Basic technique</u>: Shoot rays from the eye, trace them back to the lights
- Gives us shadows, reflection, refraction
- Distributed ray tracing gives us even more
  - Gloss, translucency, soft shadows, lens effects

So, what else is there?

Classifying Light Transport Paths

Heckbert, SIGGRAPH 90

- Paul Heckbert proposed a way of classifying light transport paths
  - And thereby stating which cases a renderer can (or can't) handle

Heckbert's Notation

- L : a light
- E: the eye
- S: a specular surface
- D: a diffuse surface
- G: a glossy surface
  - Not always included
- An example: the path from a light, to a diffuse surface, to the eye can be written LDE

An Aside: Regular Expressions

- Some useful notation:
  - For a symbol *k*
    - *k*+ : *k* appears 1 or more times
    - *k** : *k* appears 0 or more times
    - *k*? : *k* appears 0 or more times
    - *k* | *k'* : either *k* or *k*' appears

Better Example

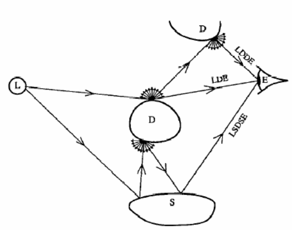


Figure 2: *Selected photon paths from light (L) to eye (E) by way of diffuse (D) and specular (S) surfaces. For simplicity, the surfaces shown are entirely diffuse or entirely specular; normally each surface would be a mixture.*

Classifying Renderers

- Optimal:
  - L(D|S)*E
  - Handles any number of diffuse or specular bounces between the light and the eye
  - Can we actually accomplish this?

Classifying Renderers

- Classical ray tracing
  - L(D)?(S)*E
  - Can handle one diffuse surface
    - Takes its color directly from the light
  - Can handle arbitrarily many specular bounces

The Rendering Equation

Jim Kajiya, 1986

- Remember this?

$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + \int_\Omega f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}')(\vec{w}' \cdot \vec{n}) d\vec{w}'$$

- The rendering equation describes the observed color of light from any point
- Actually solving it would give you every possible lighting effect
  - Let's review

The Rendering Equation

Jim Kajiya, 1986

$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + \int_\Omega f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}')(\vec{w}' \cdot \vec{n}) d\vec{w}'$$

- In short, the light out from a point in a specific direction depends on:
  - The light it emits in that direction
  - The light incident on that point from every direction, affected by
    - How reflective the material is for that pair of directions
    - How close the incoming direction is to the surface normal

The Rendering Equation

- As we've discussed before, it is far too complicated to compute the full solution to the rendering equation
  - Ray tracing simplifies by only considering light incident on a point from
    - Light sources
    - Points made visible by reflection / refraction
  - There are other simplifications that can be made, though

Radiosity

- Radiosity is an alternative lighting solution
  - It is nearly the opposite of raytracing, in terms of what effects each method is good at
    - Radiosity yields "global illumination", that is to say, diffuse-diffuse interactions
      - But not reflection or refraction
- Radiosity for lighting grew out of a similar technique used for simulating heat transfer

Classifying Renderers

- Radiosity
  - LD*E
  - Can handle arbitrarily many diffuse-diffuse interactions
  - No reflections
    - Note that this makes the radiosity solution for a scene view independent

Radiosity Assumptions

- Essentially, radiosity treats all surfaces in a scene as emitters (or potential emitters)
  - All surfaces are opaque
  - All surfaces are diffuse
  - Objects are in a vacuum (a pretty fair assumption)

Radiosity Benefits

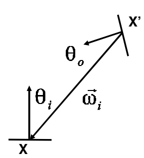
- Our first real "global illumination" solution
  - Now we can handle diffuse-diffuse interactions
  - Don't have to do "ambient light" hacks anymore
- Solved in object space
  - Totally view independent
  - Can precompute radiosity and "bake it in" to a texture

How Radiosity Simplifies the Rendering Equation

- Instead of considering incoming light at a point over all possible angles
  - Think about it in terms of the light that is outgoing from other surfaces

$$L_i(\mathbf{x}, \vec{\omega}_i) = L_o(\mathbf{x}', \vec{\omega}_o{}')V(\mathbf{x}', \mathbf{x})$$

  - Here, $\omega_o = -\omega_i$
  - $V(\mathbf{x}', \mathbf{x})$ is the visibility term
    - 0 or 1, depending on whether point **x** is visible from **x**'



How Radiosity Simplifies the Rendering Equation

- This observation allows us to rewrite the rendering equation (without the emitter component) as

$$L_o(\mathbf{x}, \vec{\omega}_o) = \int_\Omega f_r(\mathbf{x}, \vec{\omega}_o, \vec{\omega}_i) L_o(\mathbf{x}', \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

- The next step is to make the integral over surfaces, instead of angles

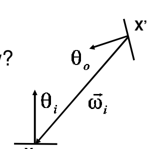
Converting Angles to Areas

- The solid angle subtended by a distant patch is related to its size and its distance

$$d\vec{\omega} = \frac{dA}{r^2}$$

- We can rewrite this
  - Remember Lambert's cosine law?
  - There is a similar effect here

$$d\vec{\omega}_i = \frac{\cos\theta_o \, dA'}{|\mathbf{x}' - \mathbf{x}|^2}$$



How Radiosity Simplifies the Rendering Equation

- So now we can rewrite the rendering equation as

$$L_o(\mathbf{x}, \vec{\omega}_o) = \int_\Omega f_r(\mathbf{x}, \vec{\omega}_o, \vec{\omega}_i) L_o(\mathbf{x}', \vec{\omega}_o{}') V(\mathbf{x}', \mathbf{x}) \frac{\cos\theta_o \cos\theta_i}{|\mathbf{x}' - \mathbf{x}|^2} dA'$$

- Note that $V(\mathbf{x}', \mathbf{x}) \frac{\cos\theta_o \cos\theta_i}{|\mathbf{x}' - \mathbf{x}|^2} dA'$ is a constant dependent only on the geometry

The Geometry Term

- For simplicity, we define
  $$G(\mathbf{x}',\mathbf{x}) = G(\mathbf{x},\mathbf{x}') = \frac{\cos\theta_o \cos\theta_i}{|\mathbf{x}'-\mathbf{x}|^2} V(\mathbf{x}',\mathbf{x})$$
  - Note the symmetry (G = G')
- Now we can rewrite the rendering equation again
  $$L_o(\mathbf{x},\vec{\omega}_o) = \int_S f_r(\mathbf{x},\vec{\omega}_o,\vec{\omega}_i) L_o(\mathbf{x}',\vec{\omega}_o') G(\mathbf{x},\mathbf{x}') dA'$$

One More Simplification...

- Remember that we said that radiosity assumes only diffuse surfaces
  - This means that the reflected color is not dependent on the relationship between incoming and outgoing angles
    - That is, $f_r(\mathbf{x}, \omega_i, \omega_o)$ is a constant
      - Define $\rho(\mathbf{x}) = f_r(\mathbf{x}, \omega_i, \omega_o)$

The Diffuse Assumption

- Note that angles are now irrelevant
  - We've succeeded in rewriting things only in terms of surfaces

$$L_o(\mathbf{x}) = \rho(\mathbf{x}) \int_S L_o(\mathbf{x}') G(\mathbf{x},\mathbf{x}') dA'$$

- Can do one more rewrite: expressing in terms of radiosities

Convert to Radiosities

- Define $B = \int_\Omega L_o \cos\theta d\omega$
  - That is, B [...] outgoing light from a point
- Then $L = B / \pi$, and we can rewrite again [...]
  $$B(\mathbf{x}) = \rho(\mathbf{x}) \int_S \frac{B(\mathbf{x}')G(\mathbf{x},\mathbf{x}')}{\pi} dA'$$

Final Radiosity Equation

- For convenience, move the $(1 / \pi)$ term into G
- Bring back the emissive term, and we have
  $$B(\mathbf{x}) = E(\mathbf{x}) + \rho(\mathbf{x}) \int_S B(\mathbf{x}') G(\mathbf{x},\mathbf{x}') dA'$$
- Now we have radiosity at each point expressed <u>only</u> in terms of radiosity at each other point

And now for some hand-waving...

- The derivation from here on out is pretty intense
  - The math is helpful if you're trying to implement, but a bit too rigorous to just give you a general idea
    - I won't cover it here
  - If you want a more detailed discussion, see Prof. Lastra's slides from COMP 870 last year
    - http://www.cs.unc.edu/~lastra/Courses/COMP870_F2006/Slides/07-Radiosity_1.ppt

Radiosity Method

1. Subdivide the model into elements.
2. Select locations (nodes) on elements at which to solve for radiosity.
3. Select basis functions to approximate radiosity across the element, based on values at nodes. Most common is to assume constant value of radiosity across the element, so a single node is placed in the middle.
4. Select finite error metric. This will result in a set of linear equations.

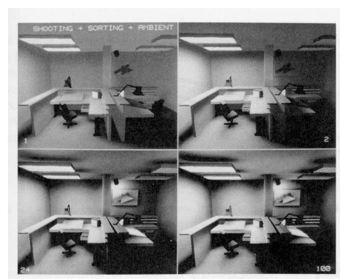
Radiosity Method

1. Compute coefficients of linear system. These are based on the geometric relationships between elements, called the form factors.
2. Solve the system of linear equations.
3. Reconstruct the radiosity function. Used to just assign radiosity values to vertices. Now textures common.
4. Render – often Gouraud interpolation of radiosity values at vertices.

In Short

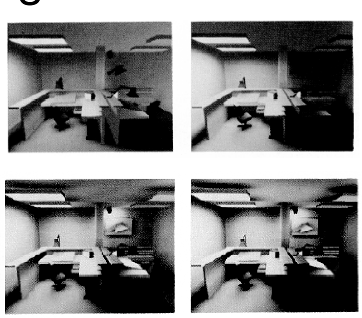
- Build a really big linear system
  - Radiosity for each patch is one variable
- Solve the whole gosh-darn thing

Progressive Radiosity

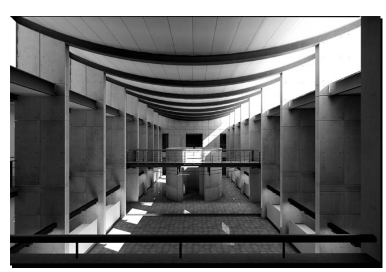


Displayed Image after 1, 2, 24, and 100 Steps

Progressive Radiosity



Some Results



Some Results

Some Results

Some Results

Some Results

Next Time

- Doing it all
  - Techniques that can produce the benefits of both raytracing and radiosity
    - Bi-directional ray tracing
    - Photon mapping