Outline

- Radiosity Calculations
- Computing Form Factors
- Radiosity and Animations
- Progressive Refinements
- Combining Radiosity and Ray Tracing

Radiosity

Overview

- Radiosity is usually used to compute lighting in a closed scene, although it can be used in any environment. Whereas ray tracing tends to produce overly spectacular, shiny objects, radiosity will usually produce much more muted effects.
- One major advantage of radiosity over ray tracing is that most scene properties are computed one time, and so it can be much more effective than ray tracing for walkthrough animations of a scene
- A scene to be rendered using radiosity will be defined as a relatively large number of patches. Here these patches will be assumed to be flat, but radiosity can also be applied to curved surfaces
- Life is somewhat easier if we're restricted to flat surfaces, and this is the most common situation. In general, a polygonal surface from the scene will either be a single patch, or will be represented as a number of smaller patches

The Radiosity Equation

- The radiosity of a surface will be the energy, per unit area per unit time, which comes off the surface into the scene
- In practical terms the radiosity of the surface will just be the intensity of the light that leaves the surface in the current wavelength (usually R, G, or B)
- The radiosity of a surface k, B_k, will be the sum of E_k, the energy emitted by the patch per unit time and area, plus all of the reflected energy per unit time and area that comes from energy coming in from all patches in the scene.
- The total energy from a patch j that reaches a patch k is the patch's radiosity (which is a unit area) multiplied by its area, and then multiplied by the fraction of the energy leaving j that hits k
- so to get the incoming energy from j that hits a unit area of k we need to divide this by the area of patch k. Tying all of this together we get

$$B_k = E_k + \rho_k \Sigma_j (B_j A_j F_k / A_k)$$

where $0 \le \rho_k \le 1$ is the reflection coefficient of surface k, F_{jk} is the form factor from j to k which tells us what fraction of the energy leaving patch j that hits patch k, and A_i is the area of patch i

- If you consider the orientation of two patches relative to each other, it is fairly obvious, and harder to prove, that both patches had unit area, then the fraction of the energy leaving patch j and hitting patch k is the same as the fraction of the energy leaving patch k that will hit patch j.

- Factoring in the area of the patches gives $A_k F_{kj} = A_j F_{jk}$. Using this in the prior equation, we get the radiosity equation:

$$B_k = E_k + \rho_k \sum (B_j F_{kj})$$

If there are n patches this is a series of n linear equations in n unknowns $B_1, \cdots, B_n$, and so the radiosities can be found using usual equation solving techniques.

A Radiosity Example

- Let us imagine a very simple scene with just three patches, P_1, P_2, and P_3. The form factors are defined in the table below:

F_{ij}	1	2	3
1	0	1/2	1/4
2	1/2	0	0
3	1/4	0	0

- So, for example, $F_{31} = 1/4$. Note that none of the energy leaving P_2 hits P_3, and vice versa, and none of the patches send light directly to themselves (zeros on the diagonal)

- The patch areas are given by $A_1 = A_2 = 2$ and $A_3 = 4$. The patch reflectances are given by $\rho_1 = \rho_3 = 1/2$ and $\rho_2 = 1/4$. The only patch to emit light into the scene is P_2 with $E_2 = 2$, and so $E_1 = E_3 = 0$

- The radiosity equations are:

$$B_1 = 0 + \frac{1}{2}\left(\frac{1}{2}B_2 + \frac{1}{4}B_3\right)$$

$$B_2 = 2 + \frac{1}{4}\left(\frac{1}{2}B_1\right)$$

$$B_3 = 0 + \frac{1}{2}\left(\frac{1}{4}B_1\right)$$

- So we have three linear equations, which if we tidy them up are:

$$4B_1 - B_2 - B_3 = 0$$

$$8B_2 - B_1 = 16$$

$$8B_3 - B_1 = 0$$

- which have the solution $B_1 = \frac{8}{15}$, $B_2 = 2\frac{1}{15}$, $B_3 = \frac{1}{15}$, as the patch radiosities.

° Computing Form Factors

- The biggest challenge in radiosity is computing the form factors. F_{ij} is the fraction of the light that leaves patch i that reaches patch j

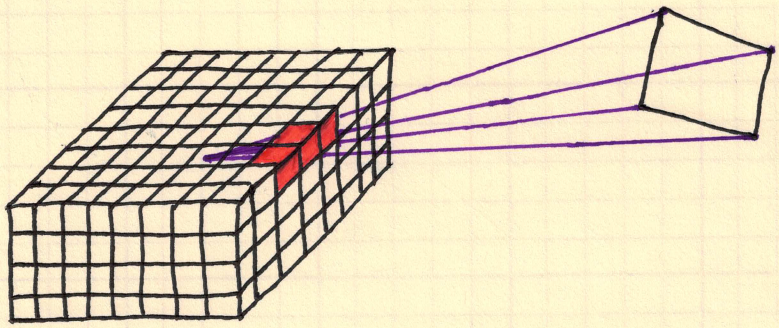
- When radiosity was first introduced form factors were computed using a really ugly double integral that was based on the angles between the patch normals and the line between the centers of the patches.

- Greenburg et al, developed two far more efficient ways to compute form factors, the hemisphere and the hemicube approaches. Both of these techniques have three advantages over the double integral technique; they are dramatically faster, they are much easier to understand, and they include hidden surface removal with virtually no extra cost.

- Although the hemisphere technique is slightly more accurate, the hemicube approach is the method of choice, because it is a lot faster, and for reasonable subdivisions of the hemicube it is accurate enough.

- The goal of the hemicube approach is to compute, for each patch k, all form factors of the form F_{jk}. I.e., the incoming form factors from all other patches

- To do this we place a unit hemicube (unit sides and half the height) on the center of the patch. We then, for every other patch, project the patch onto the hemicube by taking lines from its vertices to the center of our current cube



- When we do this projection it will cover part of the cube. If we subdivide the cube into $n \times n \times n/2$ cubelets, as shown above with $n=8$, we can compute which cubelets have their surface covered (or partially covered) by this projection.

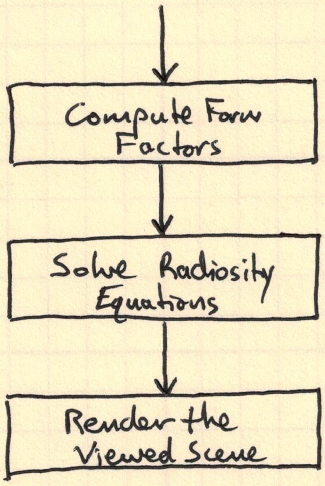
- Each cubelet surface will have an associated value called the delta form factor which is defined so that if one totals the delta form factors for a patch j projected onto k you will get the form factor, F_{jk}.

- The biggest inaccuracy that can occur here is that some cubelets will only be partially covered by the projection. We can alleviate this somewhat by assigning the appropriate fraction of the delta form factor for that cubelet to the form factor for that cubelet to the form factor total.

- Another (and better) approach is to go with a big enough value of n so that this leads to only minor inaccuracies, or (even better) to combine both approaches
- The size of the hemicube should be sufficiently small to guarantee that all other patches are outside the hemicube.
- A nice feature of the hemicube approach is that it provides a very cheap test to see whether there is an obscuring patch between the two patches we are considering
- If a cubelet is projected onto by two or more patches then we compute distances and only apply this delta form factor to the total of the patch that is closest.

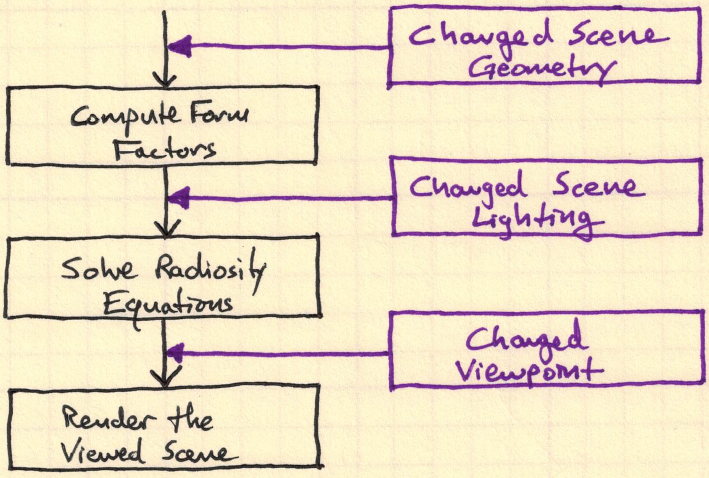
° Animations with Radiosity

- There are three major steps used when rendering a scene with radiosity, which are shown in the following figure:



- Say that we're doing a walkthrough of an architectural design for a house, and want to go from room to room through the virtual house

- As the following figure shows, the amount of new computation that we need to do depends on what we have changed in the scene:



- If the only difference is that the viewpoint changes, which is normal in a walkthrough, then after the initial setup computations we already know the colors (radiosities) of the whole structure, and so we need only to re-render with the new viewpoint.

- If the model lets us change the lighting (say, by turning on a light switch when entering a room), then the radiosity equations have been messed up and so we need to solve them again, but we don't need to redo the worst job, which is recomputing the form factors.

- Only if we change the scene geometry do we need to redo everything, we move a chair in the room, then we are forced to recompute everything. Though there are some tricks that we can use to avoid recomputing some form factors that we know can't have changed.

- There is one interesting assumption here, which is that the virtual person moving through the animation is infinitely small. Without that, as the person moves through the scene they will be getting between patches, and so blocking energy passing between those patches.

- They should also absorb light from the scene and reflect some light back into the scene. i.e., a moving person changes the scene geometry.

- Since we don't want to be recomputing the form factors and radiosity equations every time that we move in the scene, it is easier to assume that the viewer doesn't affect the scene

- So either think of the viewer as infinitely small, or as a vampire who does not interfere with radiosity calculations.

o Progressive Refinement and Other Strategies

- Obviously there are significant advantages to having as few patches as possible, as long as, we can achieve a smaller number of patches without losing visual quality.

- Some of the strategies associated with the radiosity technique begin with a small number of relatively large patches, and then subdivide these patches when they detect that visual anomalies are occurring.

- This can be at known problem areas like corners of rooms where rapid intensity changes occur close to the edges, or wherever the intensity difference between adjacent patches exceeds some threshold

- There are also variations of a strategy called progressive refinement. The basic idea is that if, say, one has 50,000 patches in a scene then solving 50,000 linear equations in 50,000 variables is a relatively expensive proposition.

- However, think about what is going on in the physical world. We have some light emitting polygons that throw light into the scene. This light then is reflected off of all polygons and the new reflected light is added to the scene.

- The light then reflects as well, and so overall image intensity increases again. In its simplest form progressive refinement just simulates the first few cycles of this, and so, for example, we might start with just the emitting surfaces, then calculate the first order reflections off surfaces, then the second order, etc.

- All we'll be missing is some intensity that hasn't been accounted for yet, but other than that after very few iterations the scene will look pretty good.

- In more complex forms of progressive refinement we can also estimate how much intensity is still missing at each step, and render this as ambient light throughout the scene.

- So at the first iteration we'll have bright emitters, and the rest of the scene will have uniform intensity. After the second iteration we'll be able to see significant differences between non-emitting patches, as ones more directly in the path of light sources will be brighter, and the scene will look more realistic.

- After very few additional iterations the scene will begin to look very good. Now we'll look at three iterations through our three polygon example from the earlier section to show how we approach the stable solution.

- The radiosity equations that we had in this example were:

$$B_1 = 0 + \frac{1}{4} B_2 + \frac{1}{4} B_3$$

$$B_2 = 2 + \frac{1}{8} B_1$$

$$B_3 = 0 + \frac{1}{8} B_1$$

- We can start all three variables at 0, and then assign those values into the right side of the equation to get new values. This will make $B_2 = 2$, and the other two will be 0.

- Repeating this process (and assigning in parallel) gives: $B_1 = 0.5$, $B_2 = 2$, $B_3 = 0$. Additional iterations will give the results in the table below:

B_1	0	0	1/2	1/2	17/32
B_2	0	2	2	2 1/16	2 1/8
B_3	0	0	0	1/16	1/16

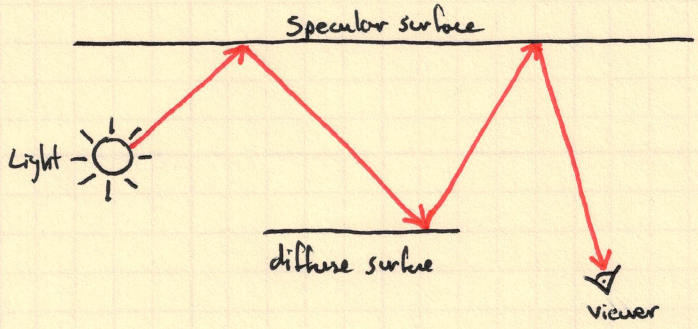
Which rapidly approaches the correct solutions of 8/15, 2 1/15, and 1/15. In this example we've taken the simplest approach, and haven't [redacted] included the estimated energy that hasn't yet been accounted for.

° Combining Radiosity and Ray Tracing

- Ray tracing works well with specular effects, where, when a ray intersects with a surface we can use a well defined reflection (and possibly refraction) ray to continue through the scene.
- If it hits a very diffuse surface, then it fails.
- Radiosity is the opposite. With radiosity surfaces are expected to distribute intensity to all surfaces that can see the current surface, which means that it is emulating a purely diffuse process.
- Unfortunately in real life we get a mixture of diffuse and specular reflections, and so neither approach is completely satisfactory
- Heckbert introduced a useful notation to describe what goes on in the real world. He has four events:

  L - Event where photons leave the light source
  E - Event where photons enter the viewer's eye
  S - Specular reflection event
  D - Diffuse reflection event

  The combination for an LSDSE event can be seen in the following diagram:



- Neither ray tracing nor radiosity can display this effectively. Using Heckbert's regular expression notation, ray tracing can handle interactions of the form LS*E or LDS*E, while radiosity can handle only LD*E.
- What we want is to handle any interactions of the form L(S|D)*E
- One approach is to perform radiosity, and then follow with a ray tracing second pass, but this will only give solutions to LD*S*E. Instead Heckbert proposed a two-pass bi-directional ray tracing system which achieves L(S|D)*E

- Heckbert's approach uses a data structure that he called a rex (adaptive radiosity texture) to store photo energy on diffuse surfaces.

- The basic idea is to have two passes, where the first, from the light source, generates $L(S*D)*$ paths and the second, from the eye, generates $DS*E$ paths.

- When combined this will give $L(S*D)*S*E$ paths. Which gives $(S...SDS...SD...S...SD)(S...S)$ where any S sequences can be empty, shows that this is equivalent to $L(S|D)*E$, which is what we want.

- The first phase distributes photons into the environment, and for any photon finds the closest intersection. If it is a reflecting surface then it is either reflected as usual or it is absorbed or refracted, based on probabilities based on surface properties.

- If it is a refracting surface the energy is absorbed and stored in the rex. This is continues, as in radiosity, both for light sources and for diffuse surfaces, using rex values, and always shooting from the surface which currently has the highest energy. So we get $L(S*D)*$ behavior.

- The second phase is similar to a traditional ray tracer, with rays from the eye thrown into the scene, which run until a diffuse surface is reached. The difference is that each time that an intersection is found we don't just send shadow feelers to the light sources, but also to the other surfaces and compute shadowing based on their rex values. This makes each ray have $DS*E$ behavior, which completes the goal.

For Next Time

Additional Notes