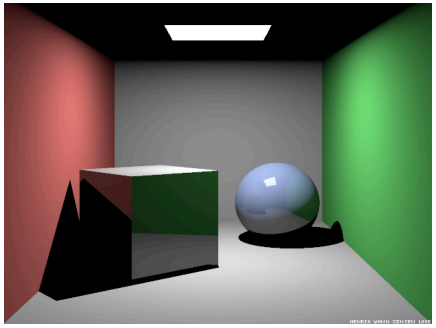
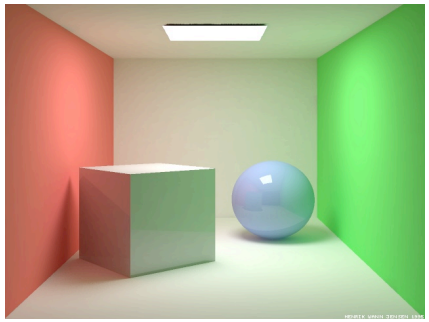
Radiosity



Ray-traced Cornell box, due to Henrik Jensen, http://www.gk.dtu.dk/~hwj



Radiosity Cornell box, due to Henrik Jensen, http://www.gk.dtu.dk/~hwj, rendered with ray tracer

Radiosity

Want to capture the basic effect that surfaces illuminate each other

Again, following every piece of light from a diffuse reflector is impractical--but combinations of brute force and clever hacks can be done

Another approach: Radiosity methods

Radiosity

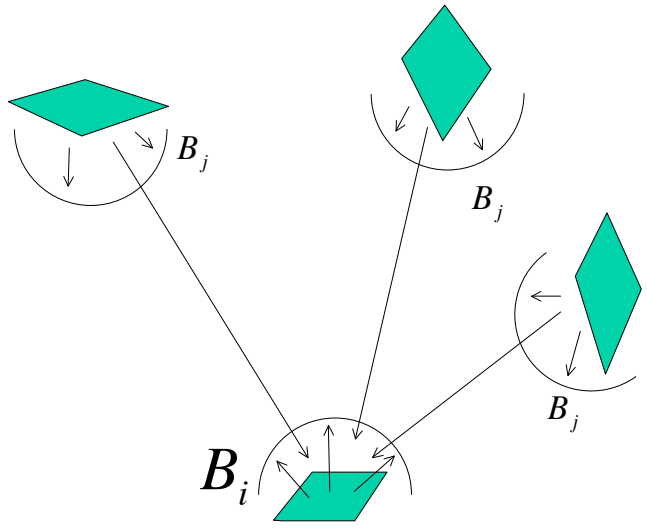
Think of the "world" as a bunch of patches. Some are sources, (and reflect), some just reflect. Each sends light towards all the others.

Consider one color band at a time (some of the computation is shared among bands).

Each surface, i, *radiates* reflected light, B_i

Each surface, *emits* light E_i (if it is not a source, this is 0).

Denote the albedo of surface i as ρ_i



Radiosity equation

$$B_i = E_i + \rho_i \sum_j F_{j \to i} B_j \frac{A_j}{A_i}$$

The form factor $F_{j \to i}$

is the fraction of light leaving dA_j arriving at dA_i taking into account orientation and obstructions

Useful relation

$$A_i F_{i \to j} = A_j F_{j \to i}$$

The equation now becomes

$$B_i = E_i + \rho_i \sum_j F_{i \to j} B_j$$

Rearrange to get

$$B_i - \rho_i \sum_j F_{i \to j} B_j = E_i$$

In matrix form

$$\begin{bmatrix} 1-\rho_1 F_{1\to1} & -\rho_1 F_{1\to2} & \dots & -\rho_1 F_{1\to n} \\ -\rho_2 F_{2\to1} & 1-\rho_2 F_{2\to2} & & -\rho_2 F_{2\to n} \\ & & & \\ -\rho_n F_{n\to1} & -\rho_n F_{n\to2} & & 1-\rho_n F_{n\to n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \\ E_n \end{bmatrix}$$

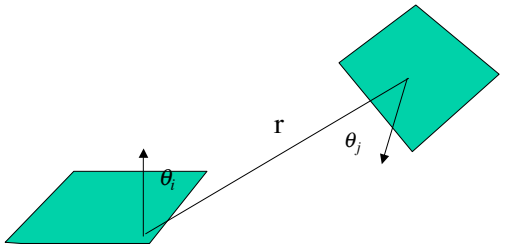
So, in theory, we just compute the Bi's by solving this (large!) matrix equation.

Optional

Optional

The fun part: Computing the $F_{i \to j}$

Without obstruction $dF_{dj \to di} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} dA_j$



Fancy methods exist for of computing and/or approximating storing form factors (e.g. hemisphere and hemicube methods)

Not covered in 2006

Fancy methods exist for of computing and/or approximating storing form factors (e.g. hemisphere and hemicube methods)

Hemicube: For a given patch, project other patches onto a cube surrounding it, which is an appropriately scaled version of the projection onto the hemisphere (use cube trick because planar projection is faster).

Hemicube projections can be tagged for distance (like Z-buffer) to deal with for occlusion.

Iterative Solution (gather)

The matrix equation can be solved iteratively (Gauss-Siedel method) starting with a crude estimate of the solution.

More intuitively, consider an estimate $\hat{B}_j$ for the B_j and plug them into our equation to re-estimated them.

$$B_i = E_i + \rho_i \sum_j F_{i \to j} \hat{B}_j$$

This is sometimes referred to as "gather", as we update the brightness of each patch in turn by gathering light from the other patches.

Iterative Solution (cast)

Iteration has the additional benefit that we can provide intermediate, approximate, solutions while the user waits.

But, in the previous version, each patch gets better in turn. Better to have all patches get a little better on each iteration.

This leads to the alternative approach where energy is cast from each patch in turn to update the others with:

$$B_j \quad due \ to \ B_i \ is \ p_j B_i F_{i \to j} \frac{A_i}{A_j} \quad \left(\text{do } \forall B_j \text{ then } \forall B_i\right)$$

A second advantage is that the casting can be done in order of brightness, which is obviously helpful.

Modeling

- Need to usefully represent objects in the world
- Need to provide for easy interaction
  - manual modeling
    - user would like to "fiddle" until it is right (e.g. CAD)
    - user has an idea what an object is like
  - fitting to measurements
    - laser range finder data
- Support rendering/geometric computations

Modeling tools

- Polygon meshes
- Fitting curves to points (from data)
- Fitting curves to points (user interaction)
- Generating shapes with sweeps
- Constructive solid geometry

Polygon Meshes

- Common, straightforward, often built in (e.g. torus mesh)
- Ready to render (many of the representations discussed soon are often be reduced to polygon meshes for rendering)
- Problems
  - Awkward to provide user editing
  - The number of polygons can be very large
    - Some kind of adaptive process makes sense
    - More polygons at high curvature points
    - More polygons where the object is larger
    - Extra care then needs to be taken to avoid temporal aliasing

Skipped in 06

Explicit curve representation

- Usual representation learned first
- Generally less useful in graphics, but know the term
- Explicit curve is a function of one variable. Examples
  - line, y=m*x + b
  - circle (need to glue two together) y = ±sqrt(r*r - x*x)
- Explicit surface is a function of two variables. Examples
  - plane z = m*x+ n*y + b

Skipped in 06

Implicit representation

- Also less useful for this section, but again, know the term
- An implicit curve is given by the vanishing of some functions
  - circle on the plane (2D), x*x + y*y - r*r = 0
  - twisted cubic in space,
    - x*y - z = 0, x*z - y*y=0, x*x - y = 0
- Similarly, an implicit surface is given by the vanishing of some functions
  - sphere in space x*x + y*y + z*z - r*r = 0
  - plane a*x + b*y + c*z + d = 0

Parametric representation

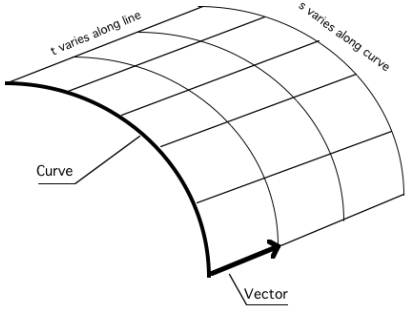
- A parametric **curve** is given as a function of one parameter. Examples:
  - circle as (cos(t), sin(t))
  - twisted cubic as (t, t*t, t*t*t)
- A parametric **surface** is given as a function of two parameters. Examples:
  - sphere as (cos(s) cos(t), sin(s)cos(t), sin(t))
- Advantage - easy to compute normal, easy to render, easy to put patches together, ranges can be easy (e.g. half circle)
- Disadvantage - intersecting with rays for ray tracing can be hard

Generating Surfaces

- We can construct surfaces from curves in a variety of user intuitive ways
  - Extruded surfaces
  - Generalized cones
  - Surfaces of revolution
  - Sweeping (generalized cylinders)
- In many the examples that follow, we will assume that we know how to generate a 3D parameteric curve (studied later)
  - e.g. twisted cubic as (t, t*t, t*t*t)

Extruded surfaces

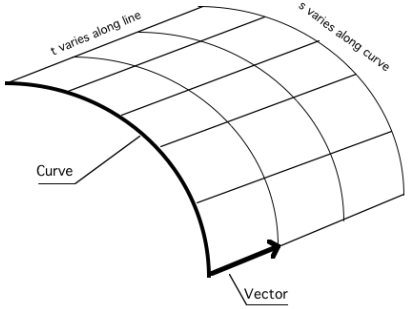
- Geometrical model - Pasta machine
- Take curve and "extrude" surface along vector
- Many human artifacts have this form - rolled steel, etc.



Parametric formula?

Extruded surfaces

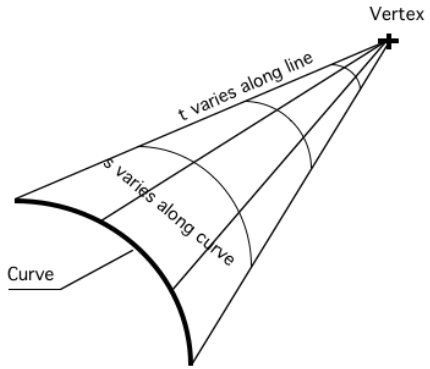
- Geometrical model - Pasta machine
- Take curve and "extrude" surface along vector
- Many human artifacts have this form - rolled steel, etc.



$$(x(s,t), y(s,t), z(s,t)) = (x_c(s), y_c(s), z_c(s)) + t(v_0, v_1, v_2)$$

Cones

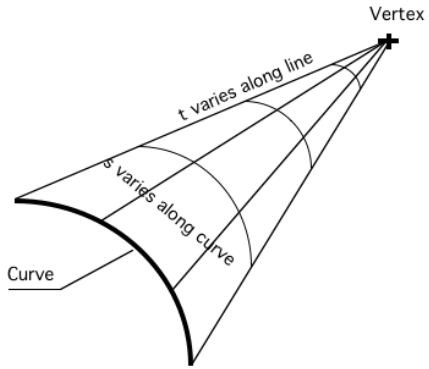
- From every point on a curve, construct a line segment through a single fixed point in space - the vertex
- Curve can be space or plane curve, but shouldn't pass through the vertex



Parametric formula?

Cones

- From every point on a curve, construct a line segment through a single fixed point in space - the vertex
- Curve can be space or plane curve, but shouldn't pass through the vertex



$$(x(s,t), y(s,t), z(s,t)) = (1-t)(x_c(s), y_c(s), z_c(s)) + t(v_0, v_1, v_2)$$