Announcements

Written Assignment 2 due today

Questions?

Ray Tracing assignment out today, due April 10

- Start early—this is significantly more work than the previous two assignments
- The starter code is pretty simple—you're responsible for all program structure, so plan carefully before you start.
- Construct VERY simple test cases for debugging.
- Debug thoroughly before moving on to the next feature you want to add.

Radiosity

Ray Tracing and Radiosity
Form Factors
Enhancements
Two-pass Rendering

Outline

- A Brief Review/Introduction to Radiosity
- The Radiosity Equation, Form Factors
- Putting it all together, and Improving
- More Realism: A digression, and Two-Pass Rendering

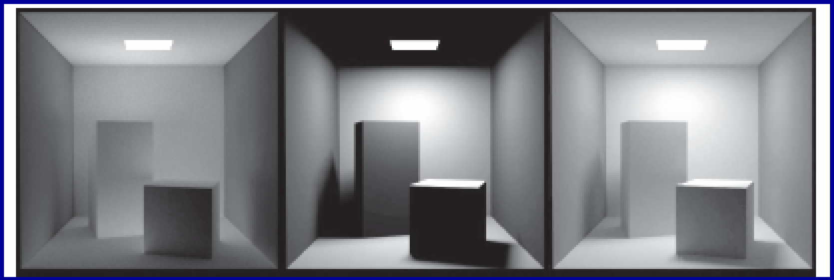
Review: Local vs. Global Illumination

Local illumination: Phong model
(OpenGL, most real-time graphics)

- Light to single surface point to viewer
- FAST
- Vastly simplified
- No representation of many natural phenomena (shadows, inter-reflections) without additional hacks

Review: Local vs. Global Illumination

- Global illumination: Ray tracing
  - Realistic specular reflection/transmission
  - Simplified diffuse reflection*
- Global illumination: Radiosity
  - Realistic diffuse reflection
  - Diffuse-only: No specular interaction*



indirect direct both

Beyond Ray Tracing

Ray tracing ignores the diffuse component of incident illumination

- to achieve this component requires sending out rays from each surface point for the whole visible hemisphere

Even if you could compute such a massive problem there is a conceptual problem—loops:

- point A gets light from point B
- point B also gets light from point A

Doing it Right

The real solution is to solve simultaneously for incoming and outgoing light at all surface points

- this is a massive integral equation

Radiosity deals with the relatively easy case of purely diffuse scenes

Or, you can sample many, many complete paths from light source to camera (photon mapping)

Advantages to diffuse-only model?

Specular interaction depends on viewer position—diffuse does not

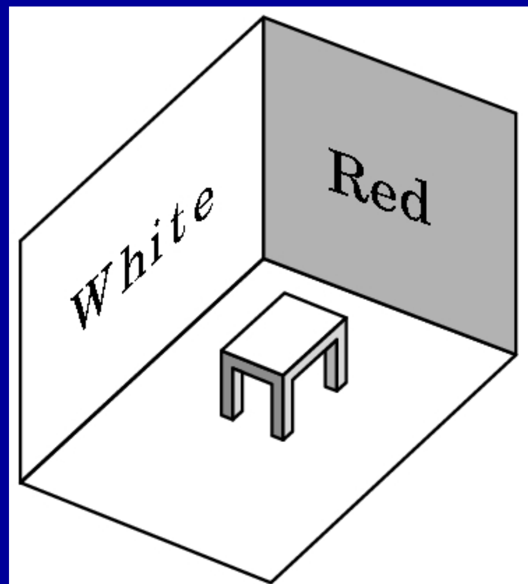
Result: The color seen at any point on any visible surface is independent of viewer position

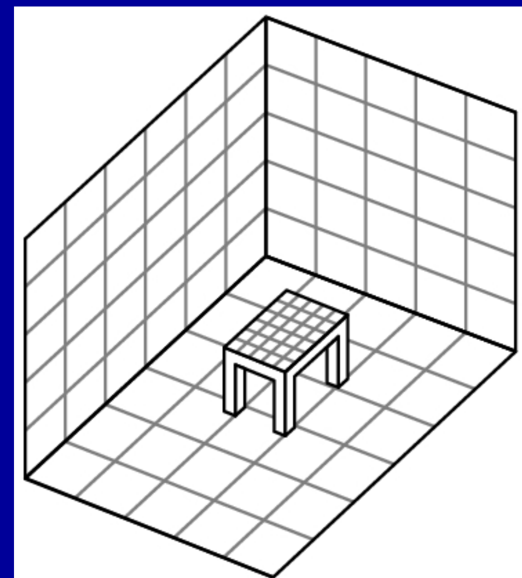
Radiosity produces a 3D model of surface patches with colors assigned to each

Can be rendered in OpenGL

Useful for architectual fly-throughs.

Radiosity

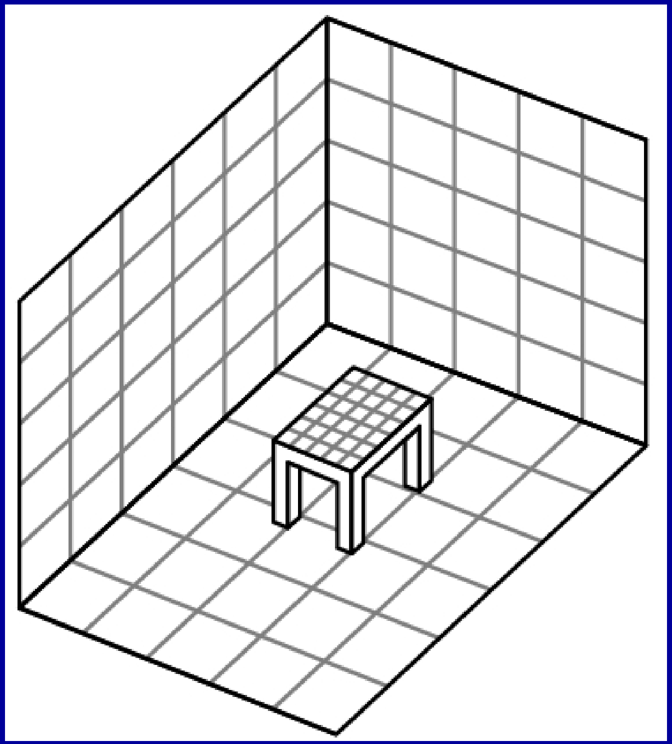




Simple scene with diffuse surfaces
White wall should show effect of being near red wall
Compute light reflected between each pair of patches

Radiosity

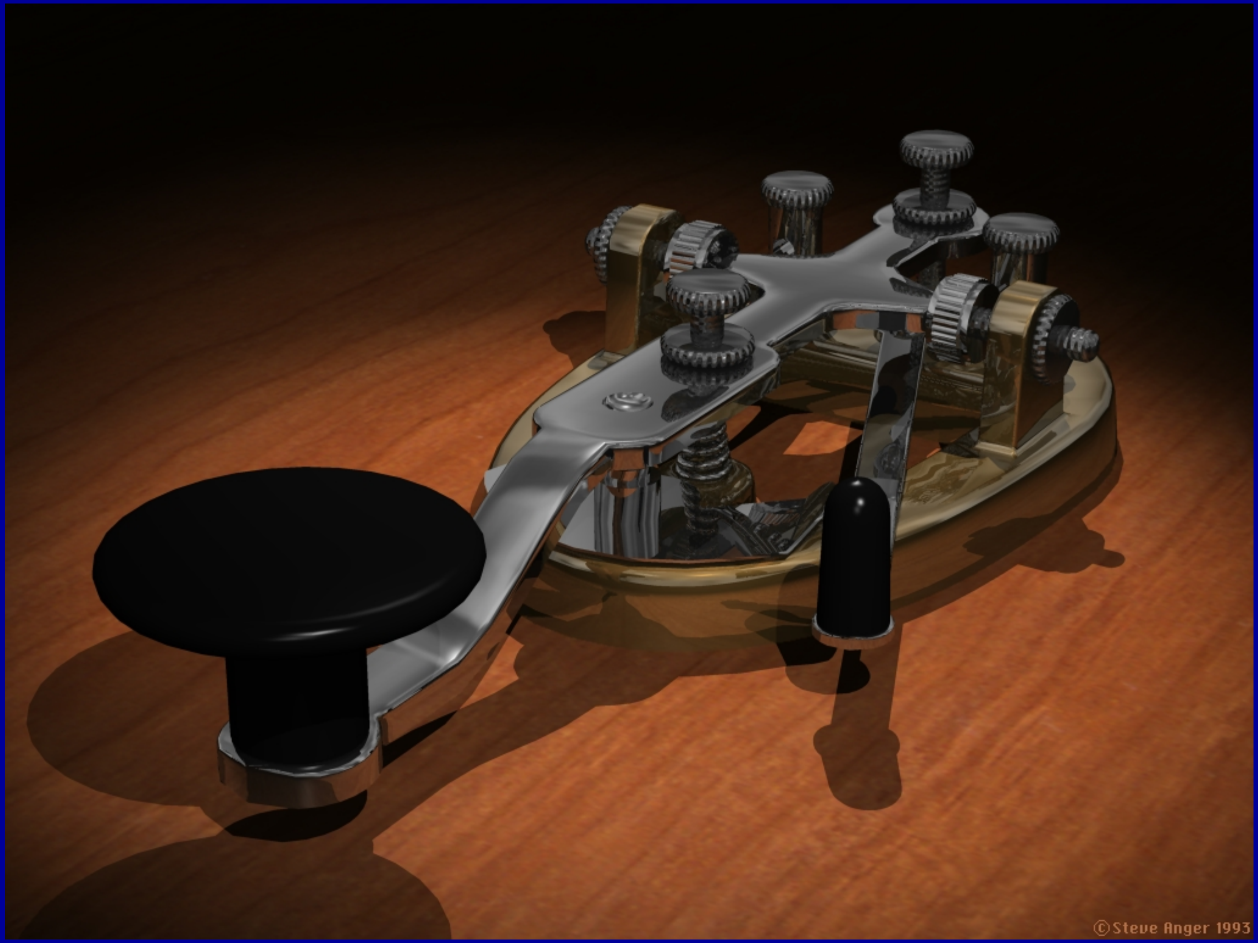
Closed environment (office, factory)
Compute interaction between all patches (over which intensity is assumed to be constant)
View independent
Difficult to do specular highlights



Radiosity Examples

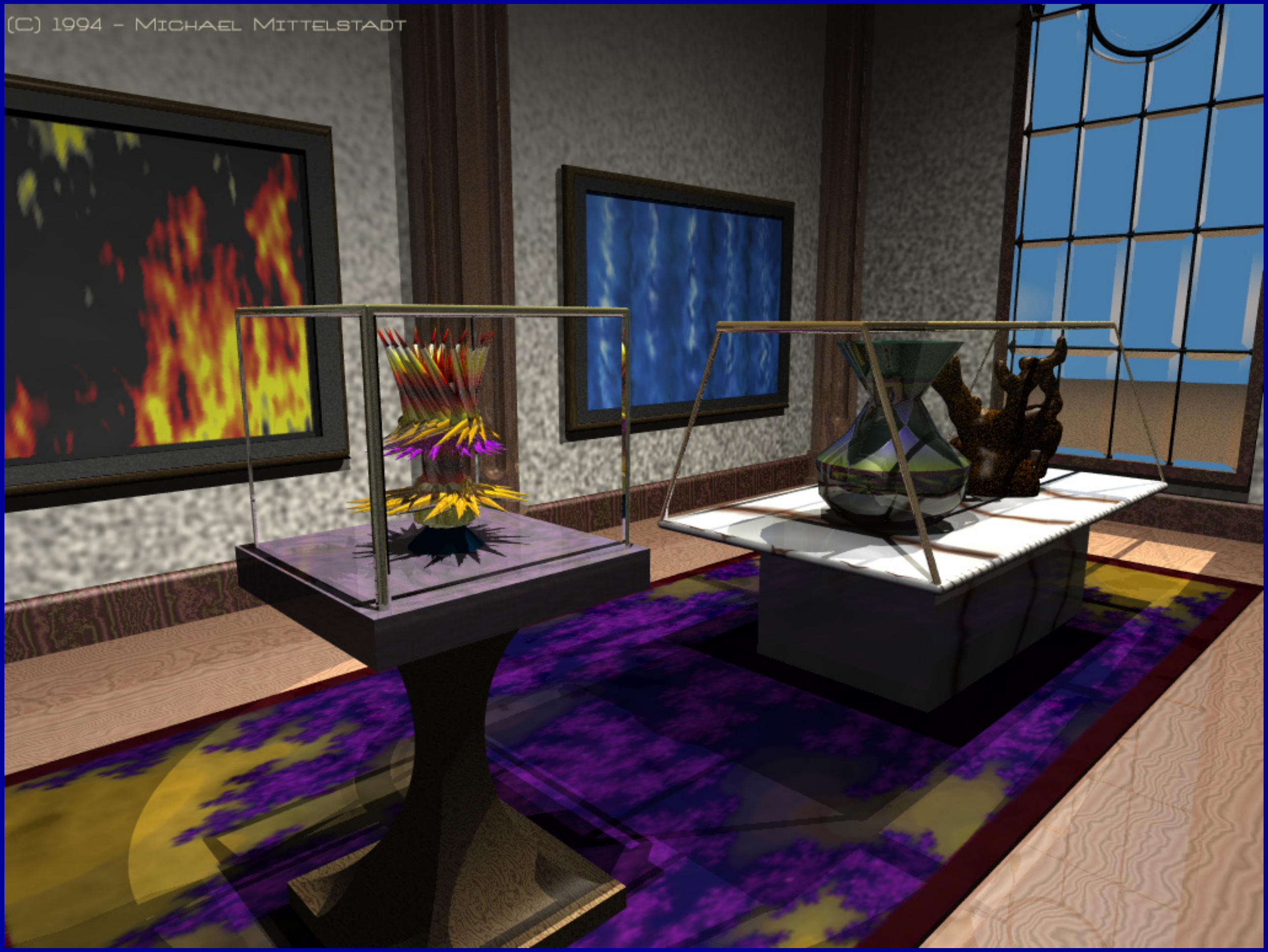
http://www.autodesk.com/us/lightscape/examples/html/index.htm

Raytracing Examples



http://www.povray.org/

Raytracing Examples



http://www.povray.org/

Raytracing Examples

http://www.povray.org/

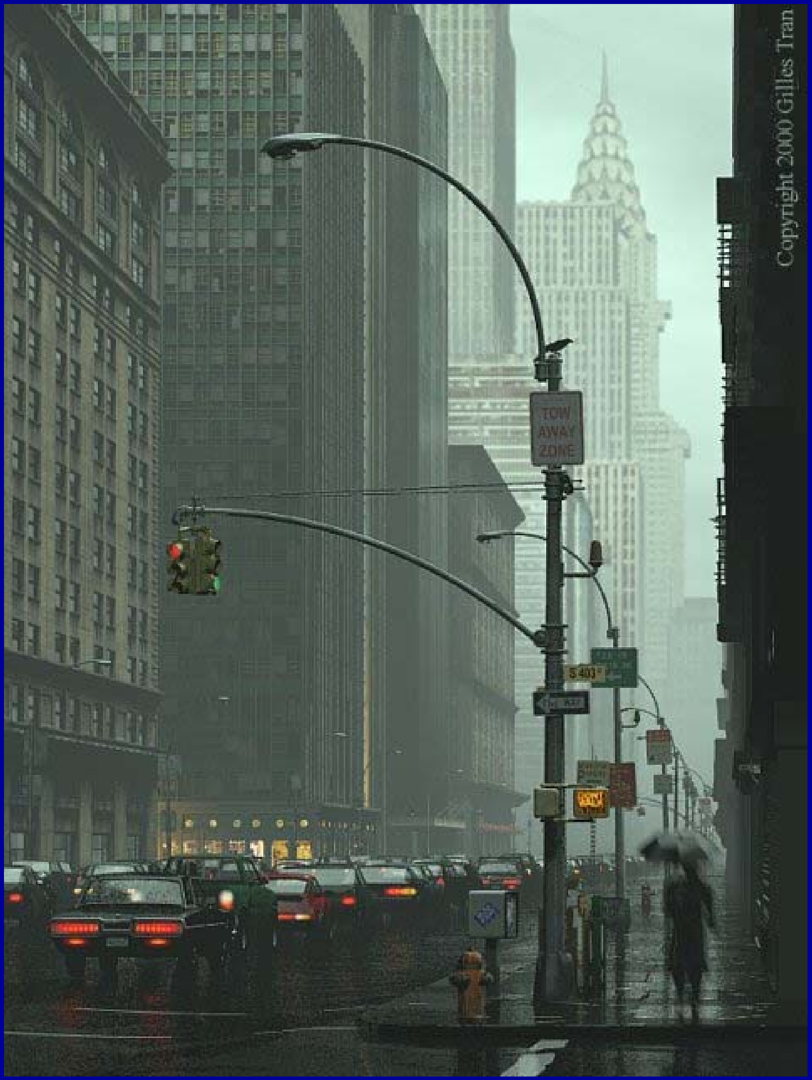
Raytracing Examples

http://www.povray.org/

Raytracing Examples

http://www.povray.org/

Raytracing Examples



 http://www.povray.org/

Raytracing Examples

http://www.povray.org/

Radiosity Examples

http://www.autodesk.com/us/lightscape/examples/html/index.htm

Radiosity Examples

http://www.autodesk.com/us/lightscape/examples/html/index.htm

Classical Radiosity in a Nutshell

Divide all surfaces into patches (squares are typical).

Determine a set of linear equations to model inter-reflection between all patches.

Solve set of simultaneous equations.

Render using standard hardware.

Assumptions of Classical Radiosity

- No participating media (no light interaction with air, fog, etc)
- Opaque surfaces—no transmission
- Radiosity is constant across element
- Colors (R, G, B) are independent

Assumptions of Classical Radiosity

Diffuse-only reflection and emission, so outgoing light radiates equally in all directions

Light radiating from a point on a surface is independent of position on the surface—constant “radiosity” across a single surface

Outline

- A Brief Review/Introduction to Radiosity
- The Radiosity Equation, Form Factors
- Putting it all together, and Improving
- More Realism: A digression, and Two-Pass Rendering

What *is* radiosity?

- Radiosity B(x) = dP/dA
  - P => Energy (light "intensity")
  - A => Area
- Integrating radiosity over a patch with respect to A will yield P for the patch
- Thus, radiosity is a representation of a patch's intensity of light per unit area

What *is* radiosity?

- Radiosity determined by the sum of the emitted and reflected energy: $$B_i A_i = E_i A_i + R_i \sum_j B_j A_j F_{ji}$$
- *i* identifies the patch whose radiosity is being determined
  j identifies a single other patch
  E is emitted energy (light sources)
  R is reflectance (how much incoming light is reflected)
  F is the form factor between two patches

What *is* radiosity?

- Radiosity determined by the sum of the emitted and reflected energy: $$B_i A_i = E_i A_i + R_i \sum_j B_j A_j F_{ji}$$

- Outgoing energy =
  Emitted energy + Reflected energy

Form Factor?

- F_{ij}: Fraction of light leaving patch *i* arriving at patch *j*
- Determined by properties of *i* and *j*:
  - Shape
  - Distance
  - Orientation
  - Occlusion by other patches

Form Factor Equation

$$A_i F_{ij} = \int_{x \in P_i} \int_{y \in P_j} \frac{\cos\theta \cos\theta'}{\pi * r^2} v(x, y)\, dy\, dx$$

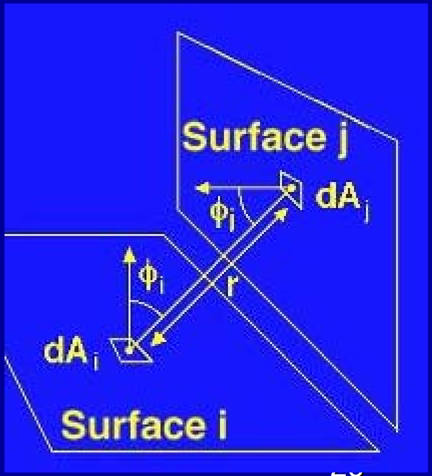
x and *y* are points in *i* and *j* respectively

r is distance from *x* to *y*

Thetas are angles between patch normals and line between *x* and *y*

v(*x*,*y*) is a visibility function

Can points *x* and *y* see each other?



Simplify

Form factors are symmetric:

$$A_i F_{ij} = A_j F_{ji}$$

$$B_i A_i = E_i A_i + R_i \sum_j B_j A_j F_{ji}$$

Divide radiosity equation by A_i

$$B_i = E_i + R_i \sum_j B_j A_j F_{ji} / A_i$$

$$B_i = E_i + R_i \sum_j B_j F_{ij}$$

Linear System

$$B_i = E_i + R_i \sum_j B_j F_{ij}$$

Our new equation gives the radiosity (*B*) of a single patch, so to specify the radiosity of all *n* patches we need *n* radiosity equations, one for each patch

Known values:
E (given), *R* (given), *F* (computable)

Unknown: *B*

n equations, *n* unknowns

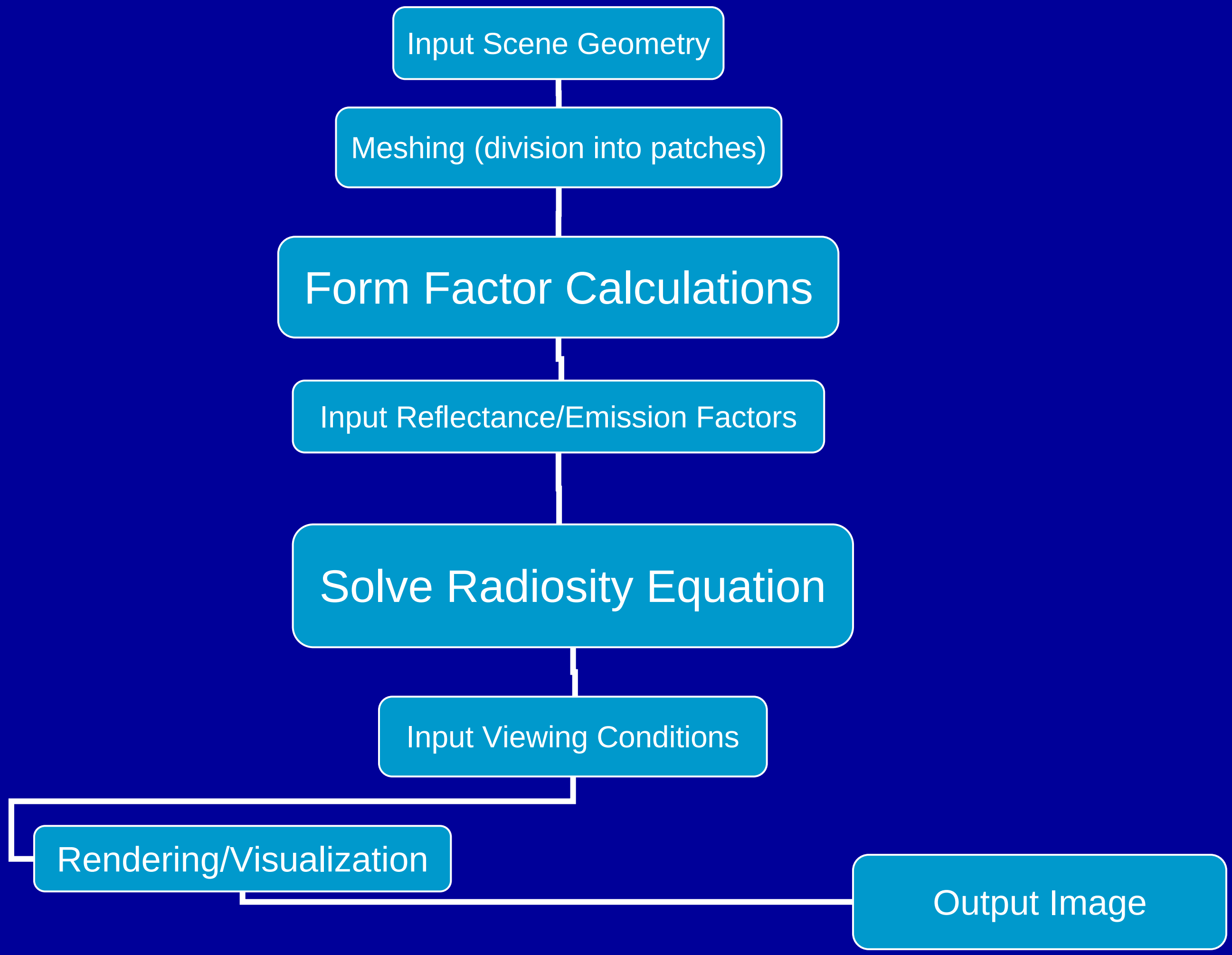
Linear System

Restate as a matrix equation…and solve

$$\begin{bmatrix} 1-R_1F_{11} & -R_1F_{12} & \ldots & R_1F_{1n} \\ -R_2F_{21} & 1-R_2F_{22} & \ldots & R_2F_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ -R_nF_{n1} & R_nF_{n2} & \ldots & 1-R_nF_{nn} \end{bmatrix} * \begin{bmatrix} B_1 \\ B_2 \\ \ldots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \ldots \\ E_n \end{bmatrix}$$

Each of our *n* linear equations contains *n* double integrals, one for each form factor.

The Radiosity "Pipeline"



Being Smart about Form Factors

Form factors depend only on scene geometry. If geometry is constant, they only need to be calculated once.

Solution of the radiosity system is independent of viewing conditions, so if only the viewer position changes, it only needs to be solved once—can walk around the scene in real-time after it's initially generated

Being Smart about Form Factors

Form factors are complicated. Full numeric approximation of these is expensive—many special cases may be solved analytically.

Because we assume that radiosity is constant across a patch, two patches are typically assumed to be fully inter-visible or not at all inter-visible. That means that patches have to be small enough to resolve shadows and other complexities

How to perform visibility testing?

Two basic methods, both of which have aliasing problems:

- Raycasting (typically slow)
- Hemicube method (z-buffer exploit)

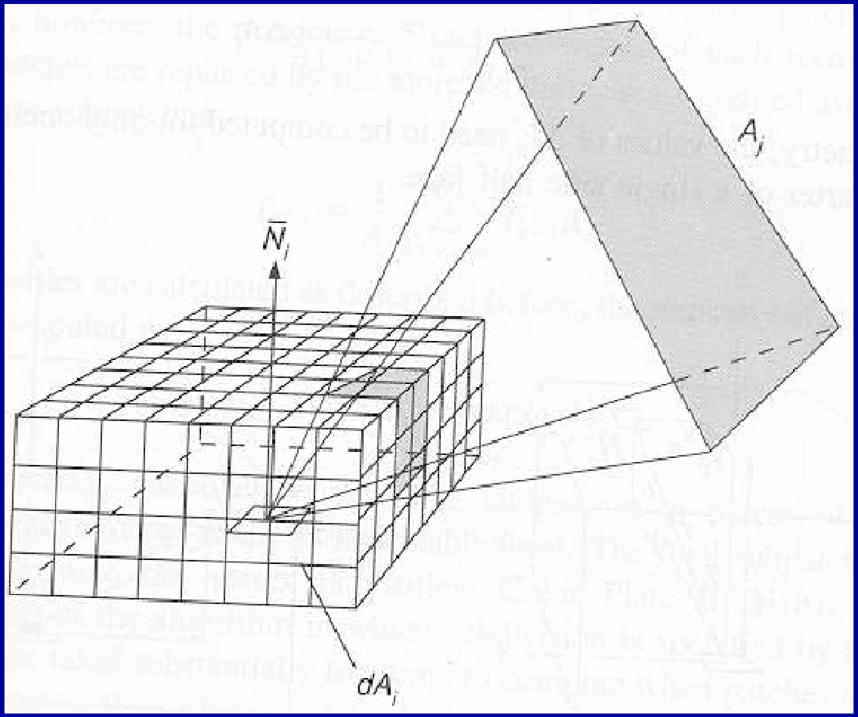
Anti-aliasing may be performed in both cases

Hemicube Visibility Testing

Render the entire scene from the perspective of the center of the current patch

Rather than color, store patch identifiers, using the z-buffer to determine visibility

Takes advantage of graphics hardware



R. Ramamoorthi

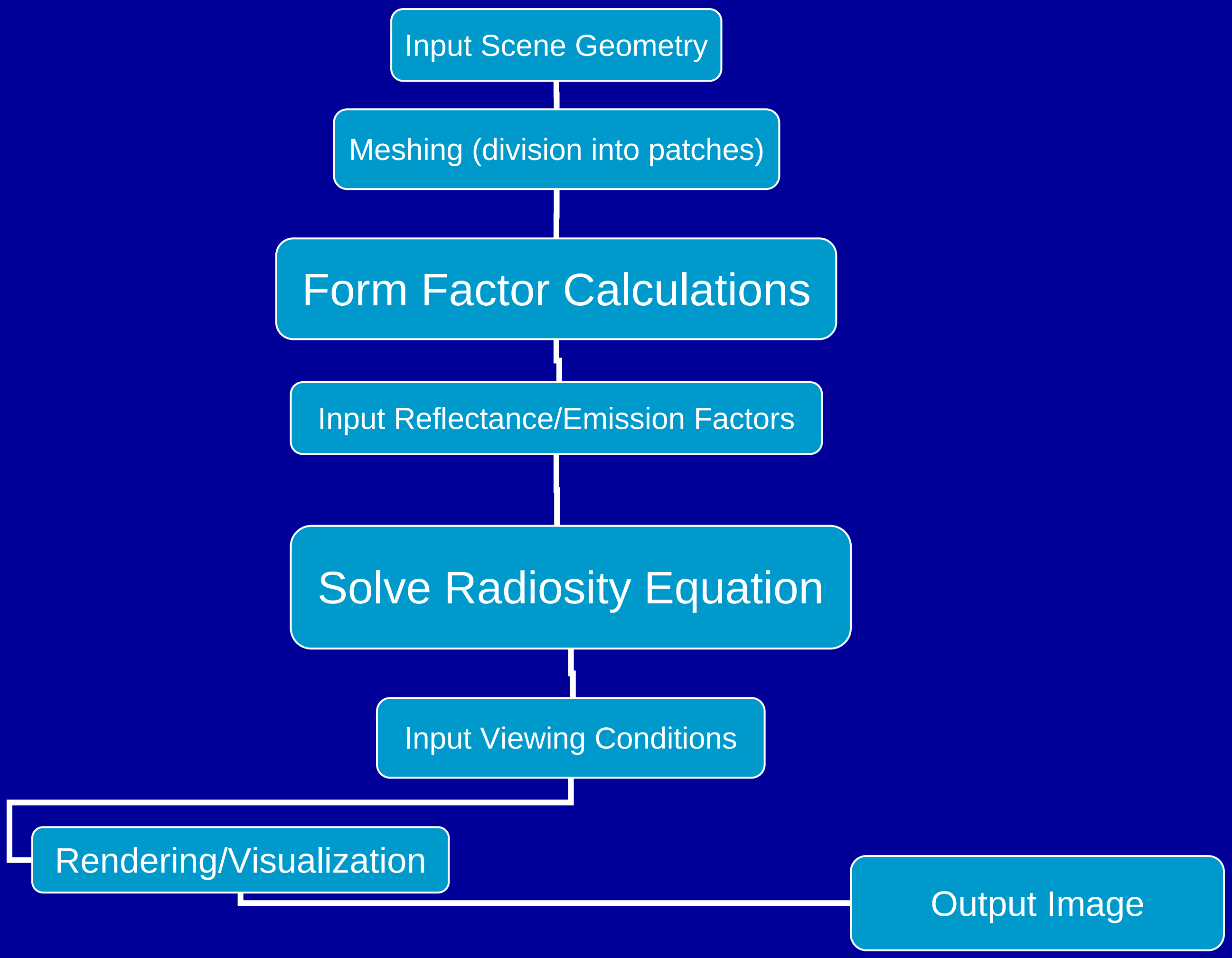
Outline

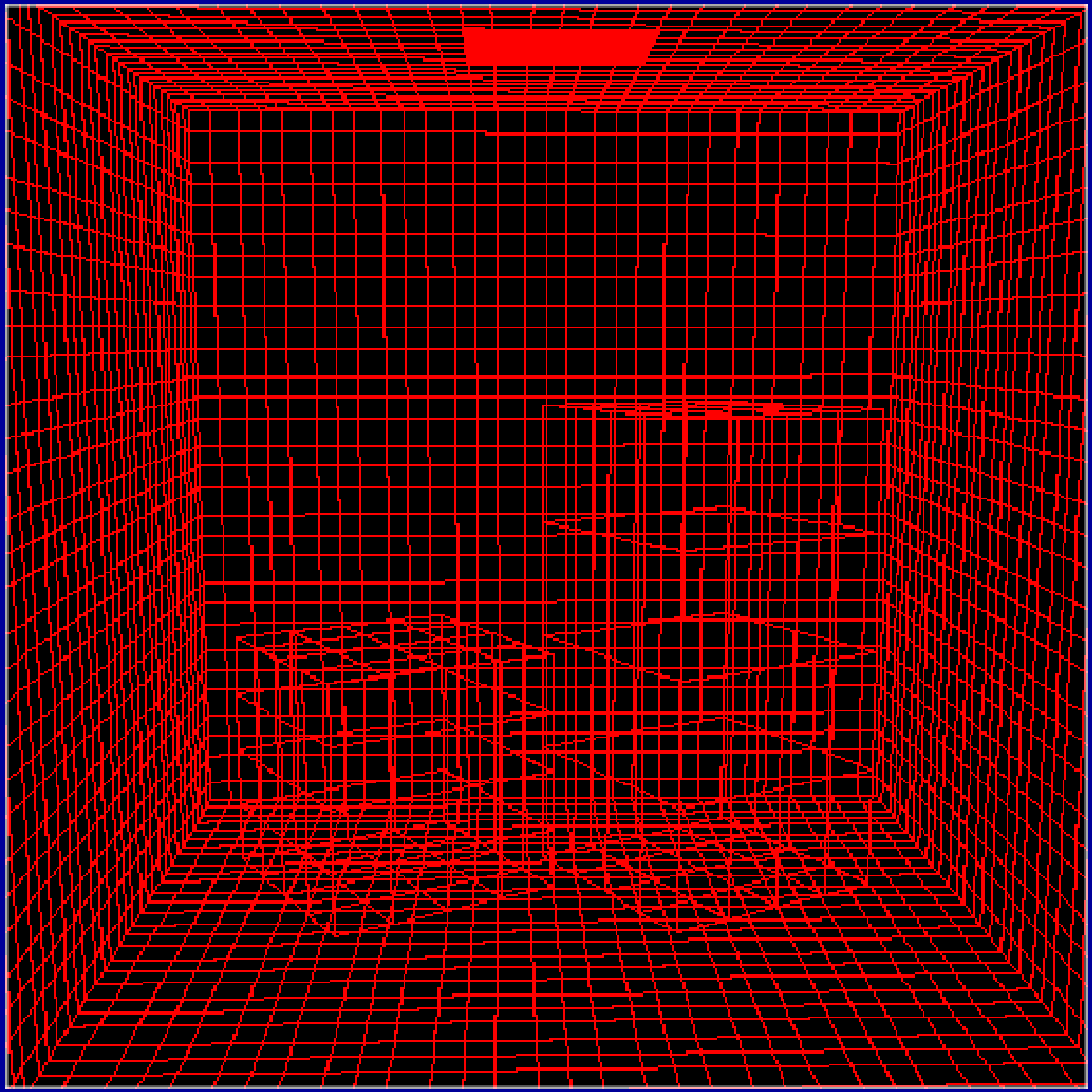
- A Brief Review/Introduction to Radiosity
- The Radiosity Equation, Form Factors
- Putting it all together, and Improving
- More Realism: A digression, and Two-Pass Rendering

Classical Radiosity in a Nutshell, Revised

- Divide all surfaces into patches.
- Calculate form factors between all patches.
  - Lighting and viewer independent
- Solve the radiosity equation.
  - Viewer independent
- Render using standard 3D hardware.

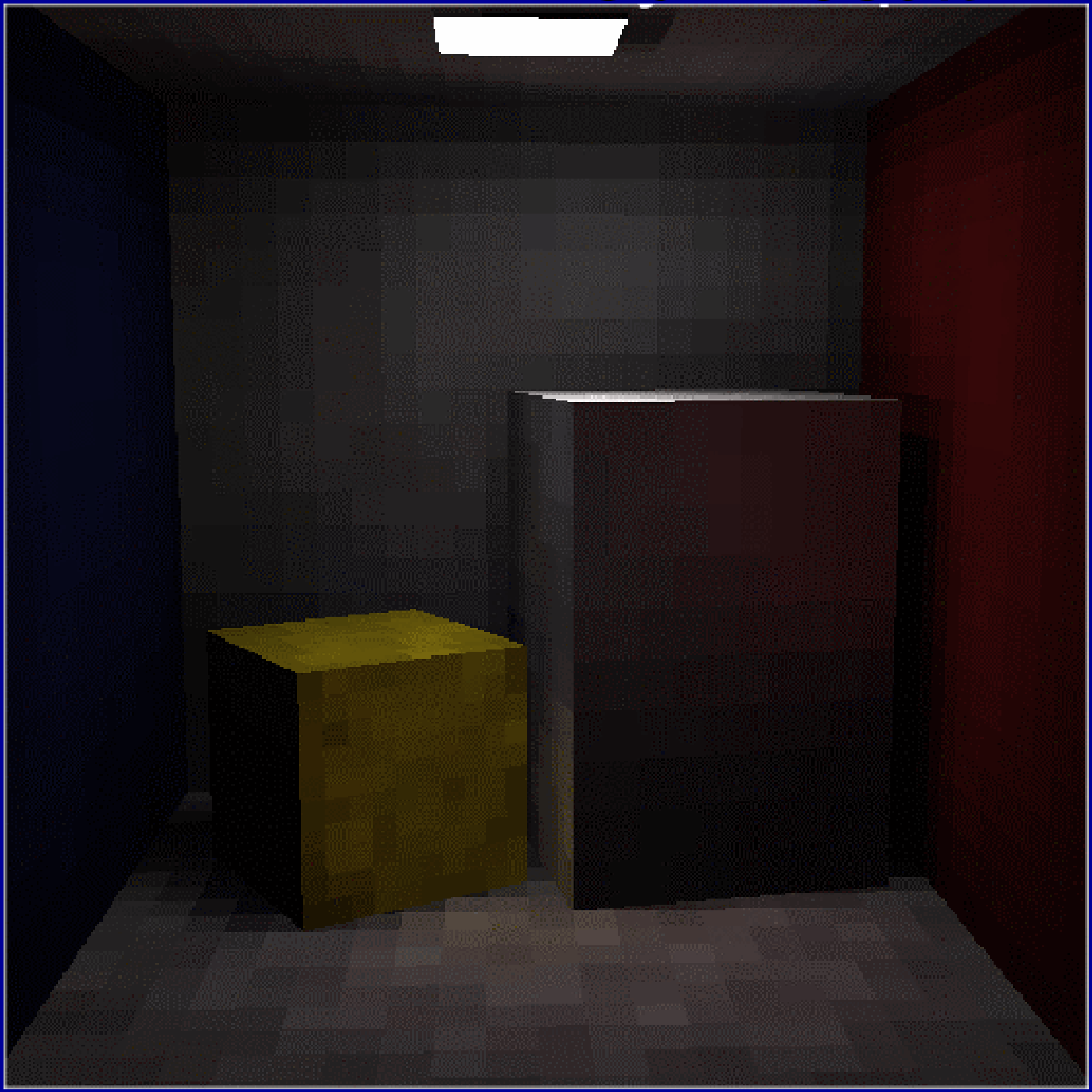
The Radiosity "Pipeline"





F. Pfenning

Our Result



What is right?
What is wrong?

What went right?

Inter-reflection effects—clearly visible between the box on the right and the wall

What went wrong?

Blocky-looking—patch boundaries extremely obvious

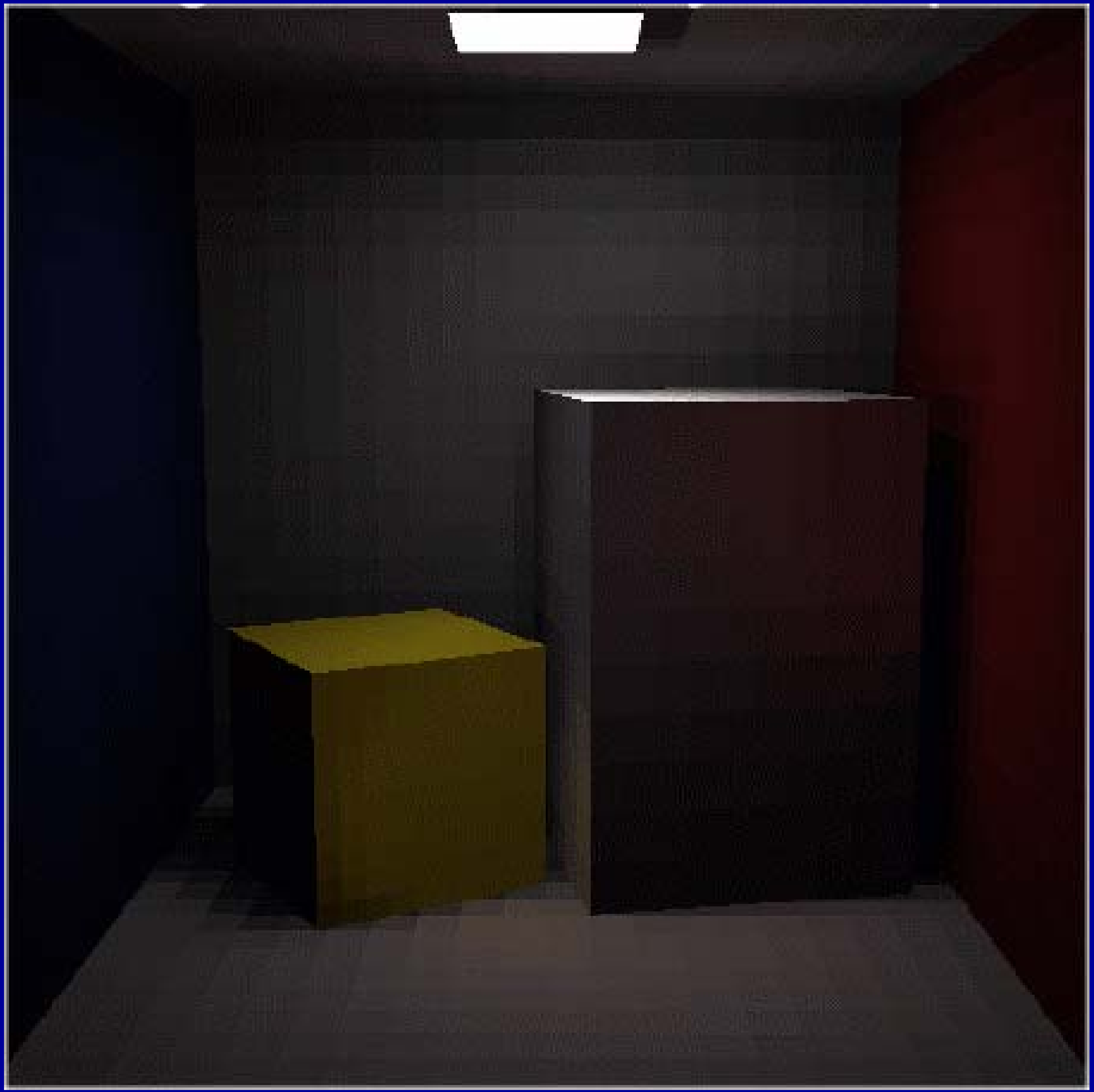
Causes of blockiness

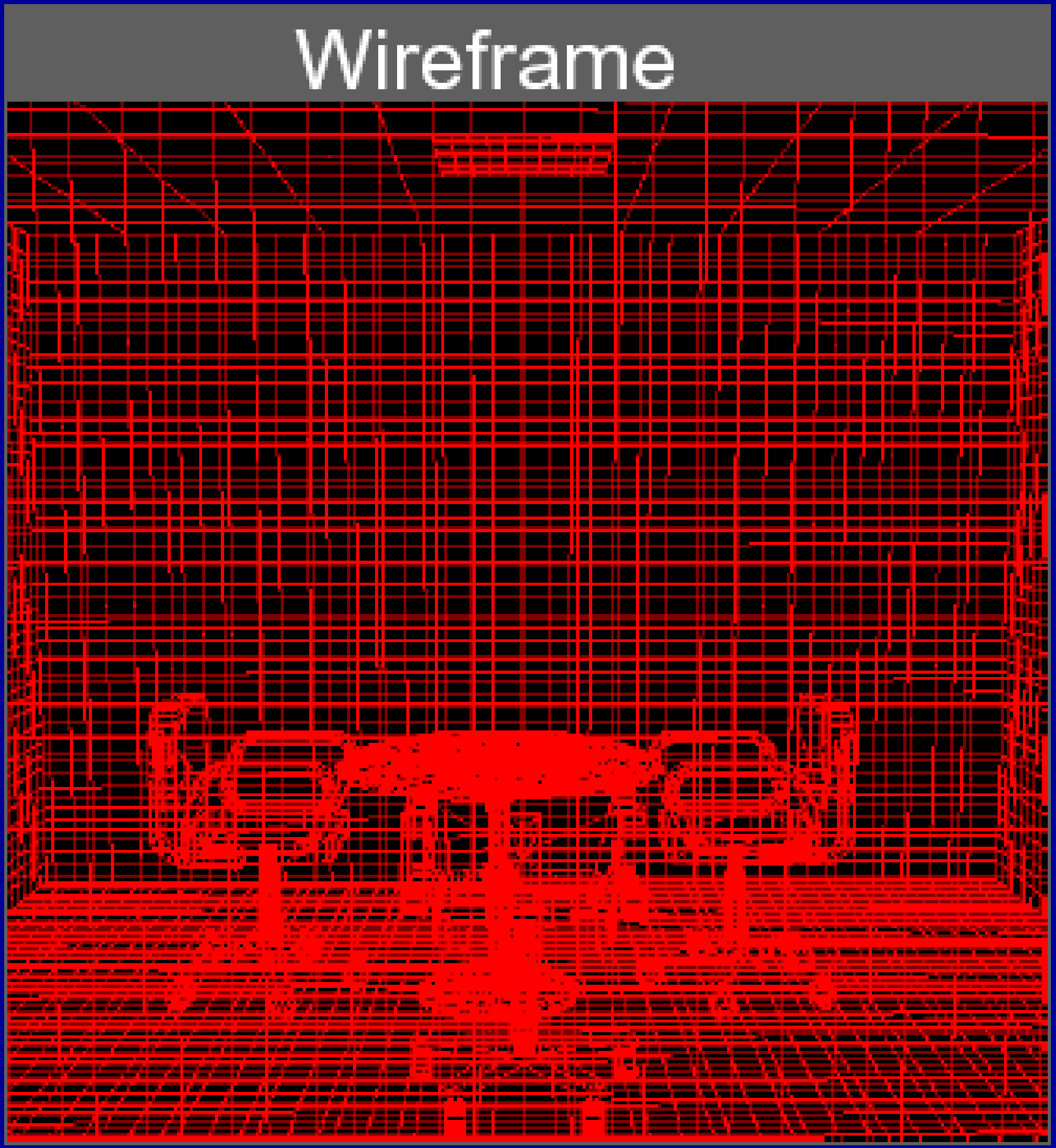
- Aliasing in hemicube method causes significant differences in radiosity between adjacent patches
- **Large patch size**

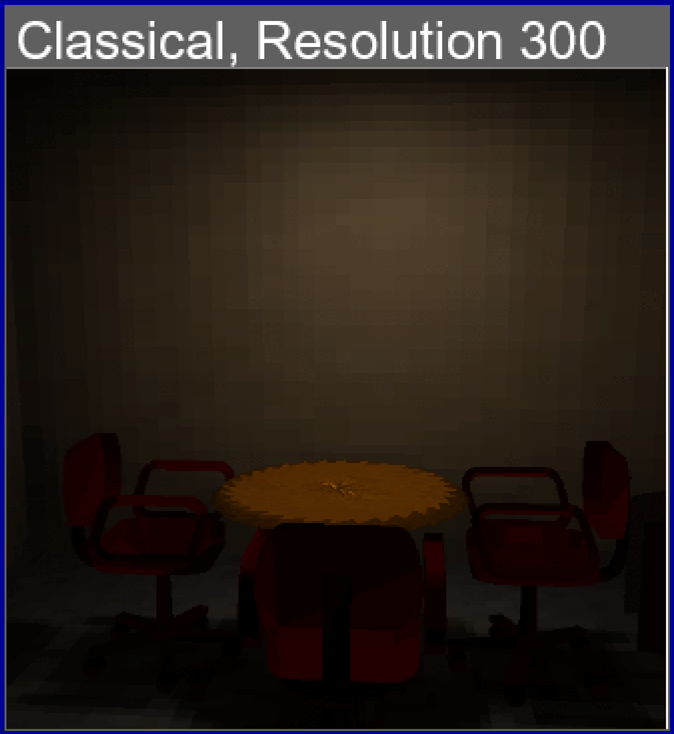
Fixes?

- Use antialiasing to clean up hemicube method
- Interpolation
  - Determine radiosity at each vertex of a patch and use bilinear interpolation to make things look smoother
- Increase patch resolution (decrease size)
  - Expensive if done uniformly – $O(n^2)$
  - How can we do this intelligently?

Antialiasing (on hemicube)




Wireframe


Classical, Resolution 300


Classical, Resolution 1200


Classical, Resolution 2500


Supersampling, Resolution 100


Classical, Resolution 2500, Interpolated


Supersampled, Res 100, Interpolated

Adaptive Subdivision

Introduce a patch substructure—divide each patch into smaller *elements*.

Keep distinction between patches and elements in order to avoid efficiency problems

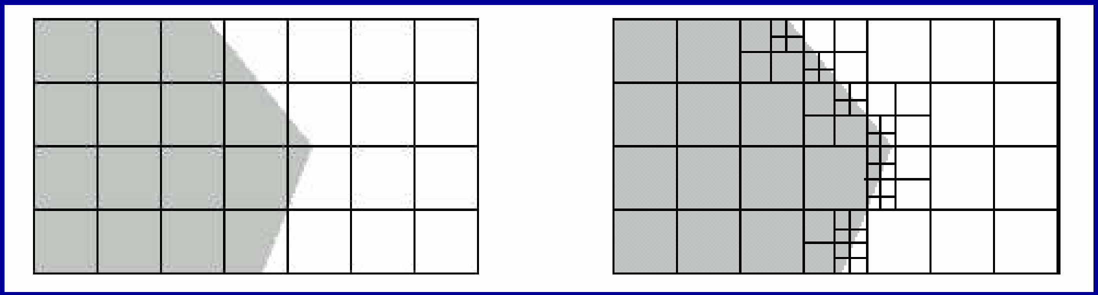
Adaptive Subdivision

Determine light transport one-way from patches onto elements, not analyzing element-to-element interaction

O(mn) for m elements and n patches. More expensive than the original n^2 approach, since $m >> n$, but much better than O(m^2).

Adaptive Subdivision

Subdivide elements adaptively:



Begin with elements identical to patches.

Determine radiosity of an element, then compare to neighbors to obtain an error value. If within some error threshold, assign constant radiosity (or optionally interpolate).

Otherwise, subdivide the element and recurse until the error threshold or a minimum element size is reached.

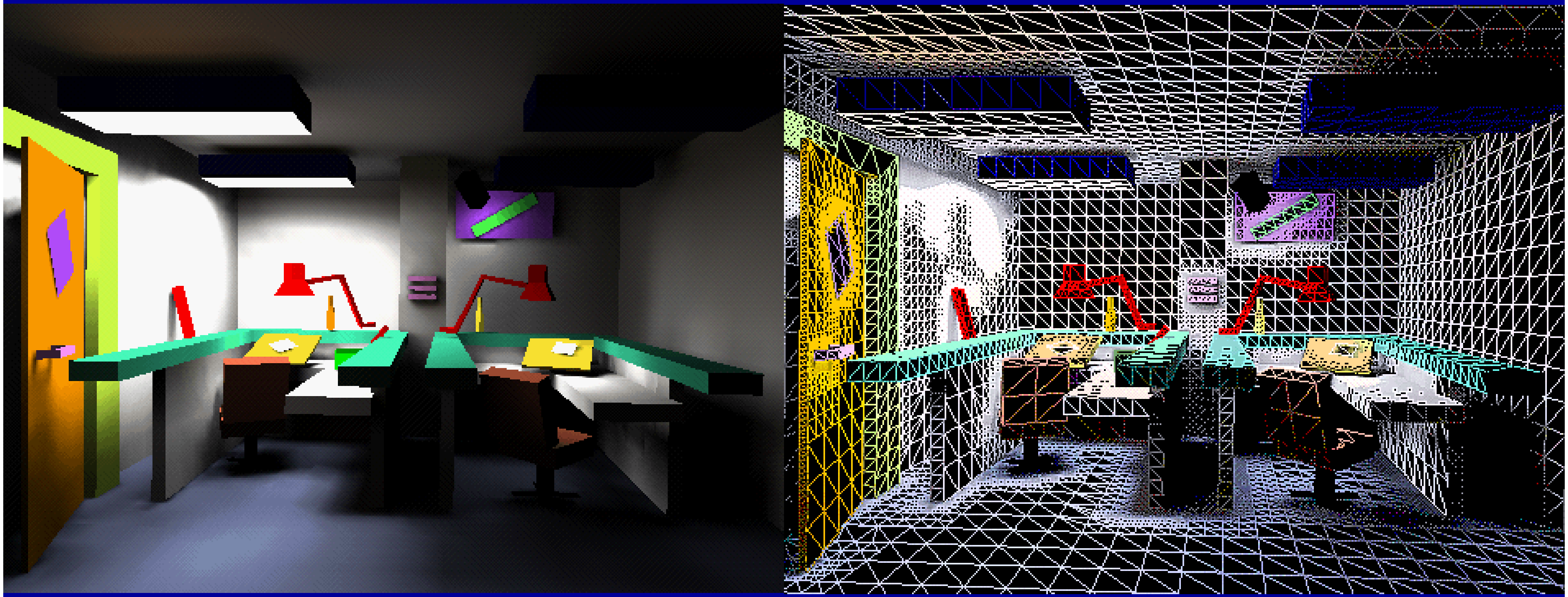
Adaptive Subdivision

Results in very smooth-looking results for a relatively small amount of extra work

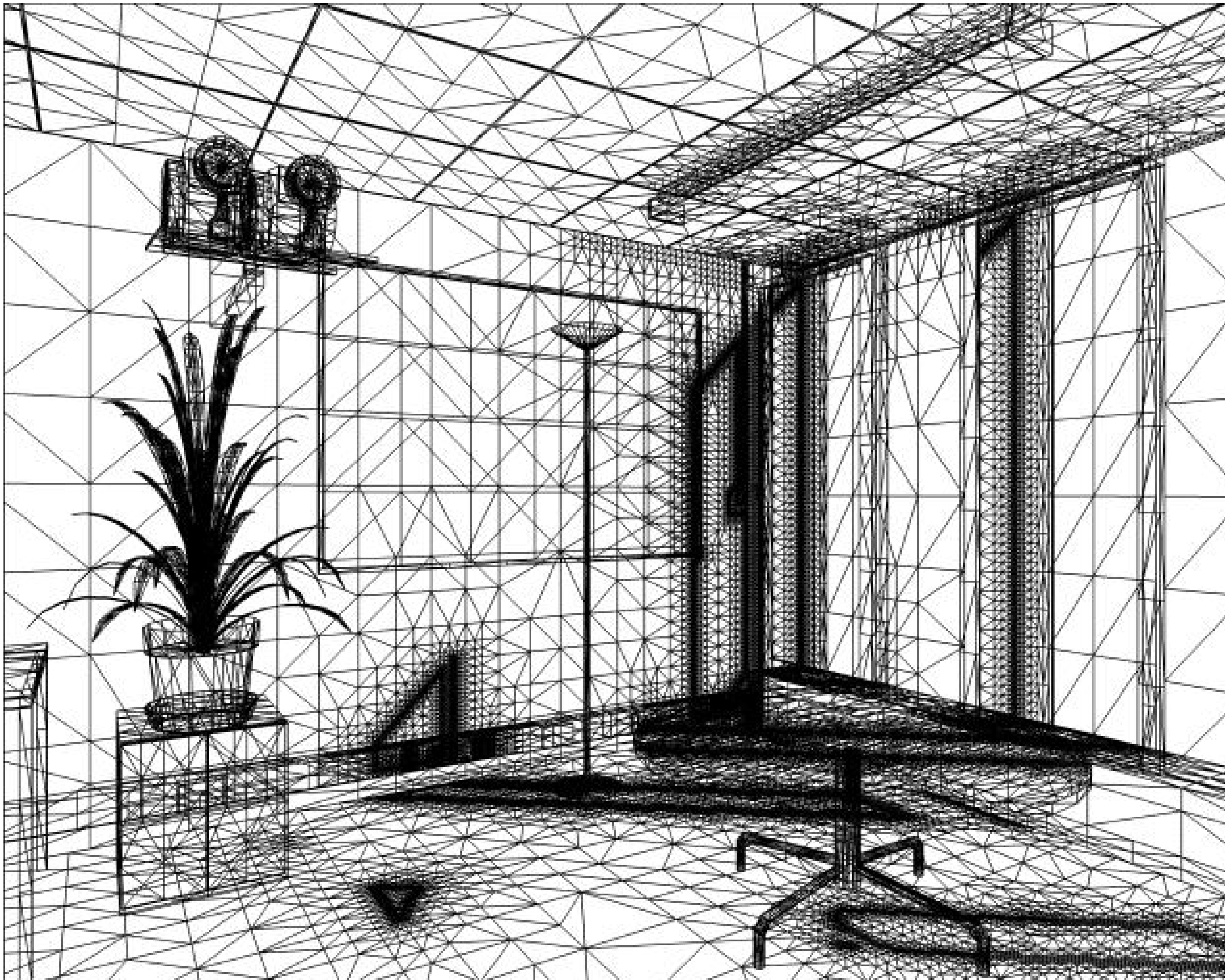
Shadows, areas near lights, and edges in general look much better

Not an idea specific to radiosity! Adaptive subdivision is a general tool used in many areas of graphics and other fields as well

Adaptive Subdivision Examples



http://www.acm.org/jgt/papers/TeleaVanOverveld97/




http://aig.cs.man.ac.uk/gallery/vrad.html

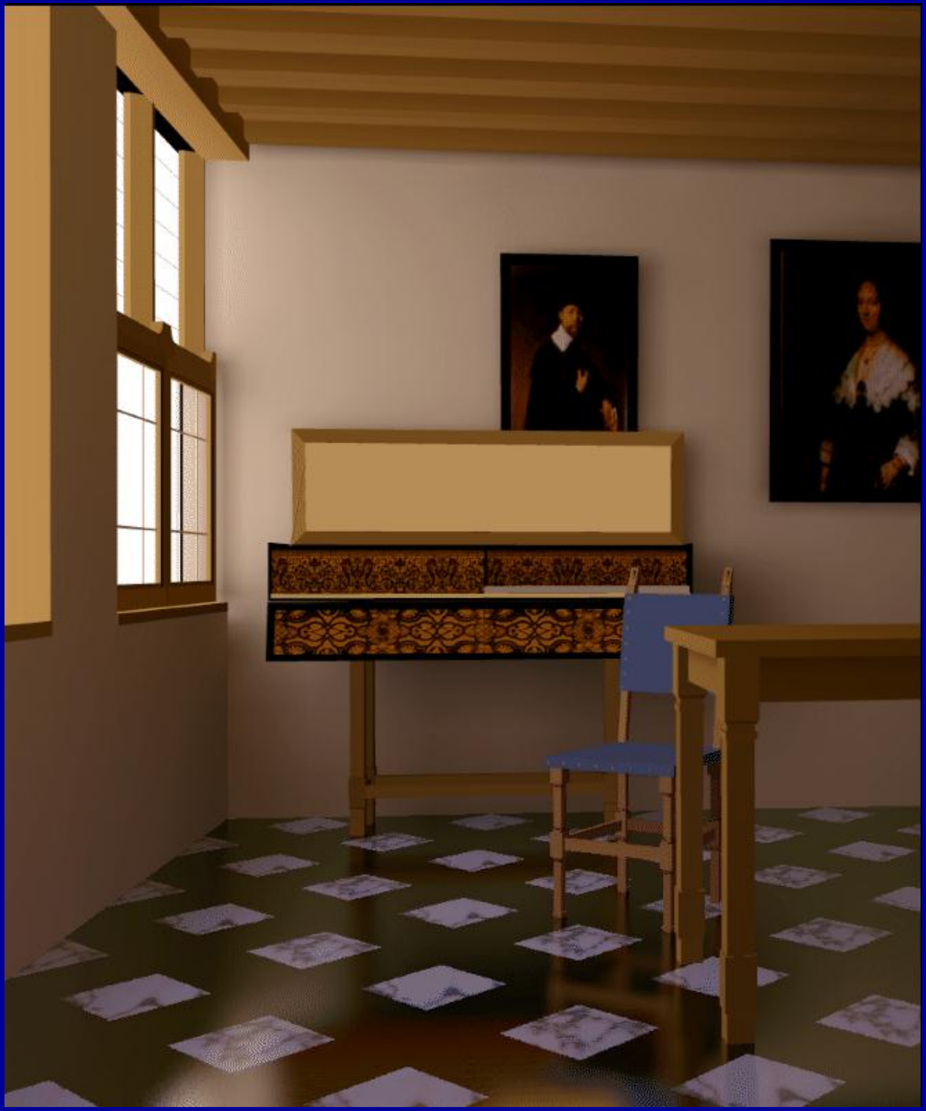
Another example

D. Lischinski

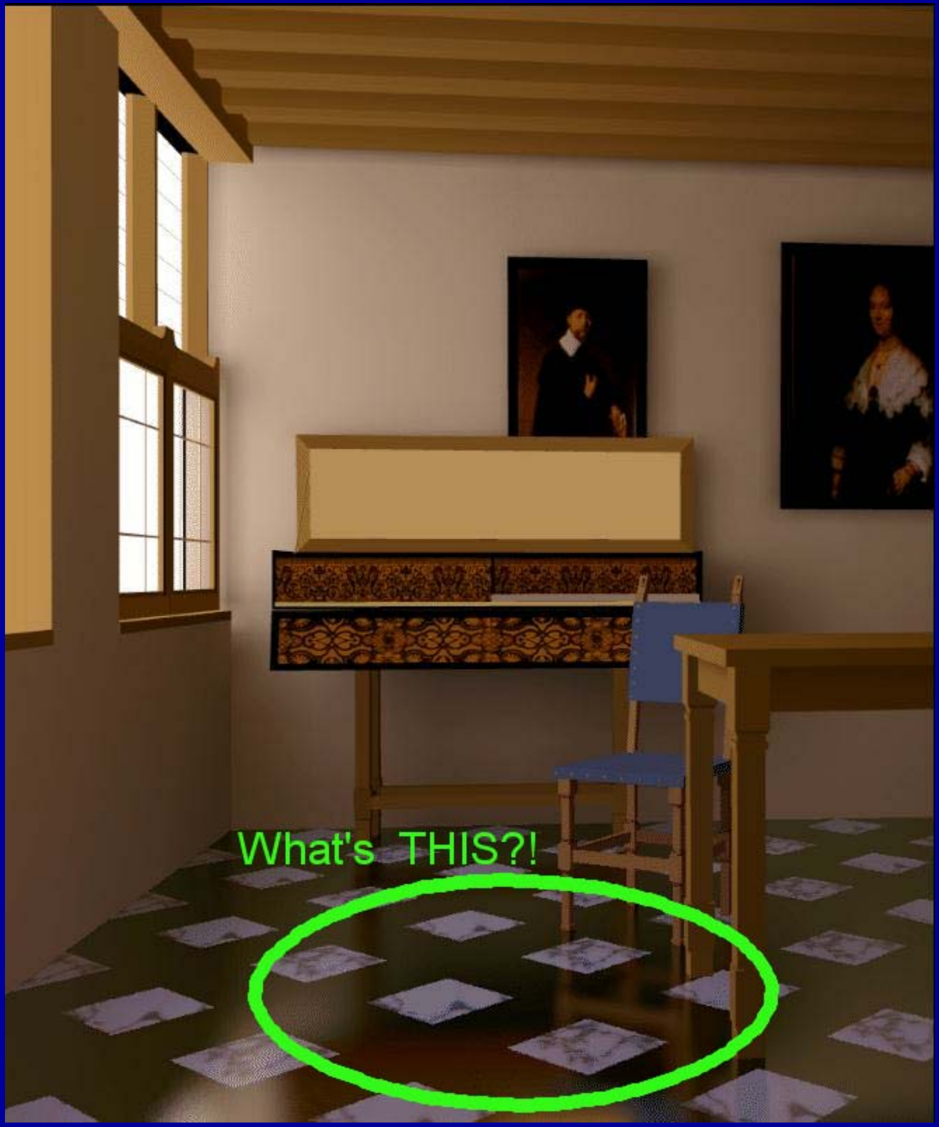
Outline

- A Brief Review/Introduction to Radiosity
- The Radiosity Equation, Form Factors
- Putting it all together, and Improving
- More Realism

Yet More Realism



Wait a minute…



D. Lischinski

Specular Effects in Radiosity?

Keep viewer independence

Light reflected differently in different directions

Calculations for each source and each direction

Impractical

A Better Idea: The Best of Both Worlds

Combine radiosity and raytracing

Goal: Represent four forms of light transport:

- Diffuse -> Diffuse
- Diffuse -> Specular
- Specular -> Diffuse
- Specular -> Specular

Two-pass approach, one for each method

First Pass: Enhanced Radiosity

- Diffuse -> Diffuse
  - Normal diffuse reflection model
  - Diffuse transmission (translucent objects) – requires modified form factor
- Specular -> Diffuse
  - Specular transmission (transparent objects, e.g. windows) – involves extended form factor
  - Specular reflection (reflective objects, e.g. mirrors) – create actual “mirror image” environment with copies of all patches. Expensive!

Enhanced Radiosity - Evaluation

- Only accounts for a single specular reflection (try creating "mirror image" environments for two mirrors facing each other)
- Accurate diffuse model
- Equations solved as in the classical method
- Still viewer-independent

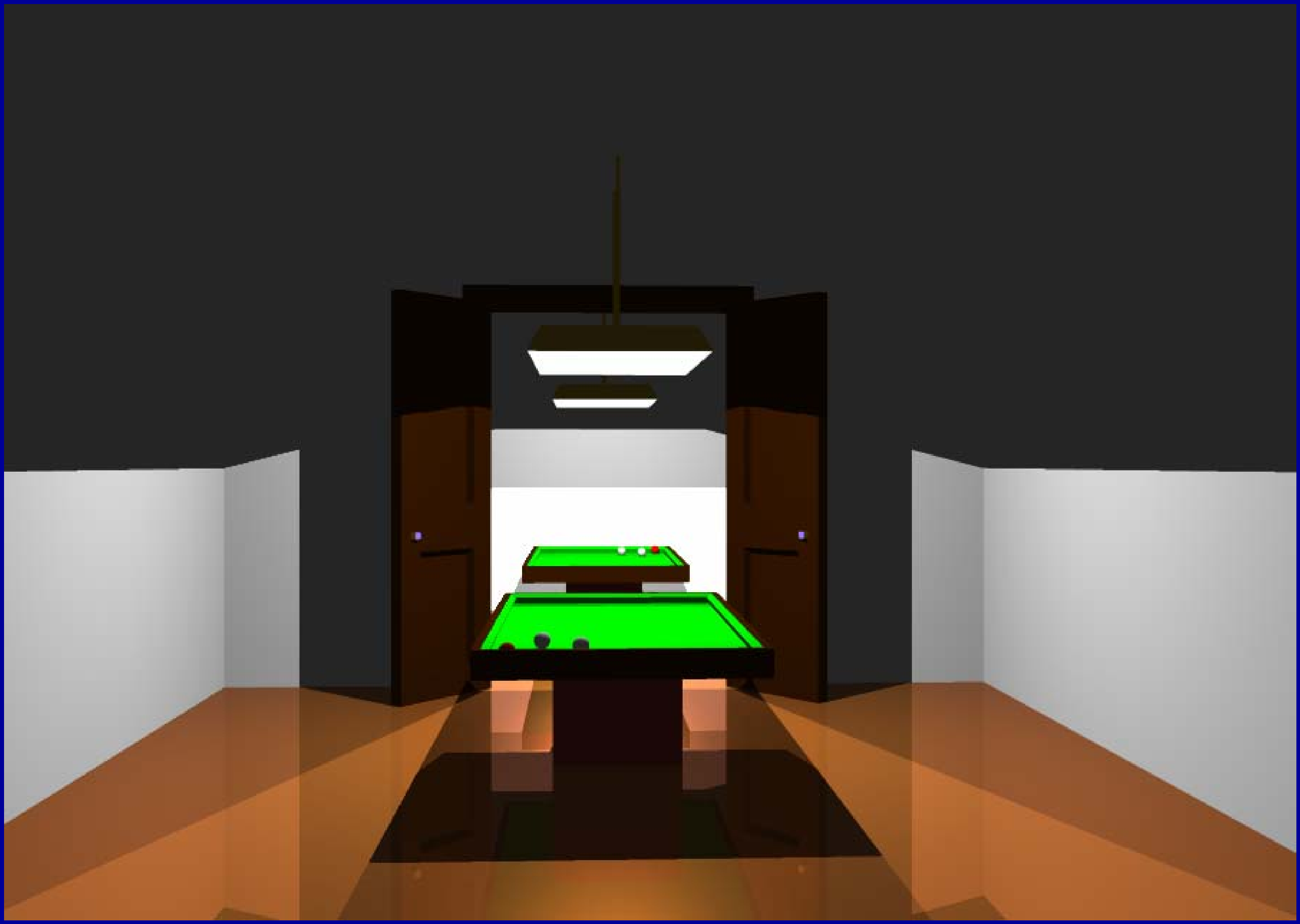
Second Pass: Enhanced Raytracing

- Specular -> Specular
  - Reflection and transmission as in classical method
- Diffuse -> Specular
  - Use the radiosity calculated in the first pass
  - Integrate incoming light over a hemisphere (or hemicube), or approximate with a tiny frustum in the direction of reflection
  - Recurse if visible surface is specular

First Pass Result

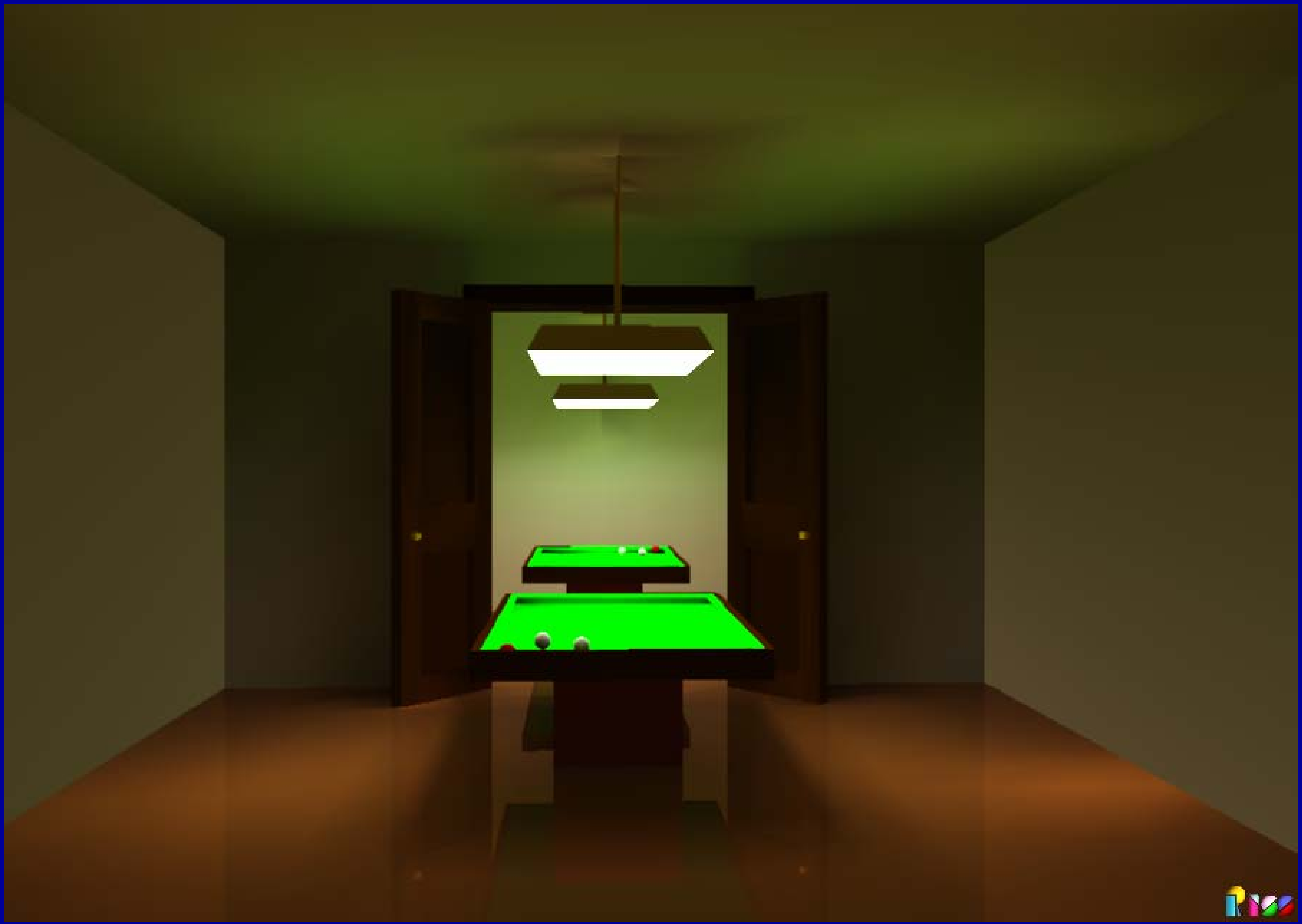
http://www.cg.tuwien.ac.at/research/rendering/rays-radio/

Second Pass Result
(radiosity info. not yet used, just raytracing)



http://www.cg.tuwien.ac.at/research/rendering/rays-radio/

Combined (Final) Result



http://www.cg.tuwien.ac.at/research/rendering/rays-radio/

Two-Pass Global Illumination: Evaluation

- Very expensive. Takes the cost of radiosity added to the cost of raytracing and then throws even more calculations into the mix
- Many approximations remain, particularly in specular -> diffuse and diffuse -> specular transport

Two-Pass Global Illumination: Evaluation

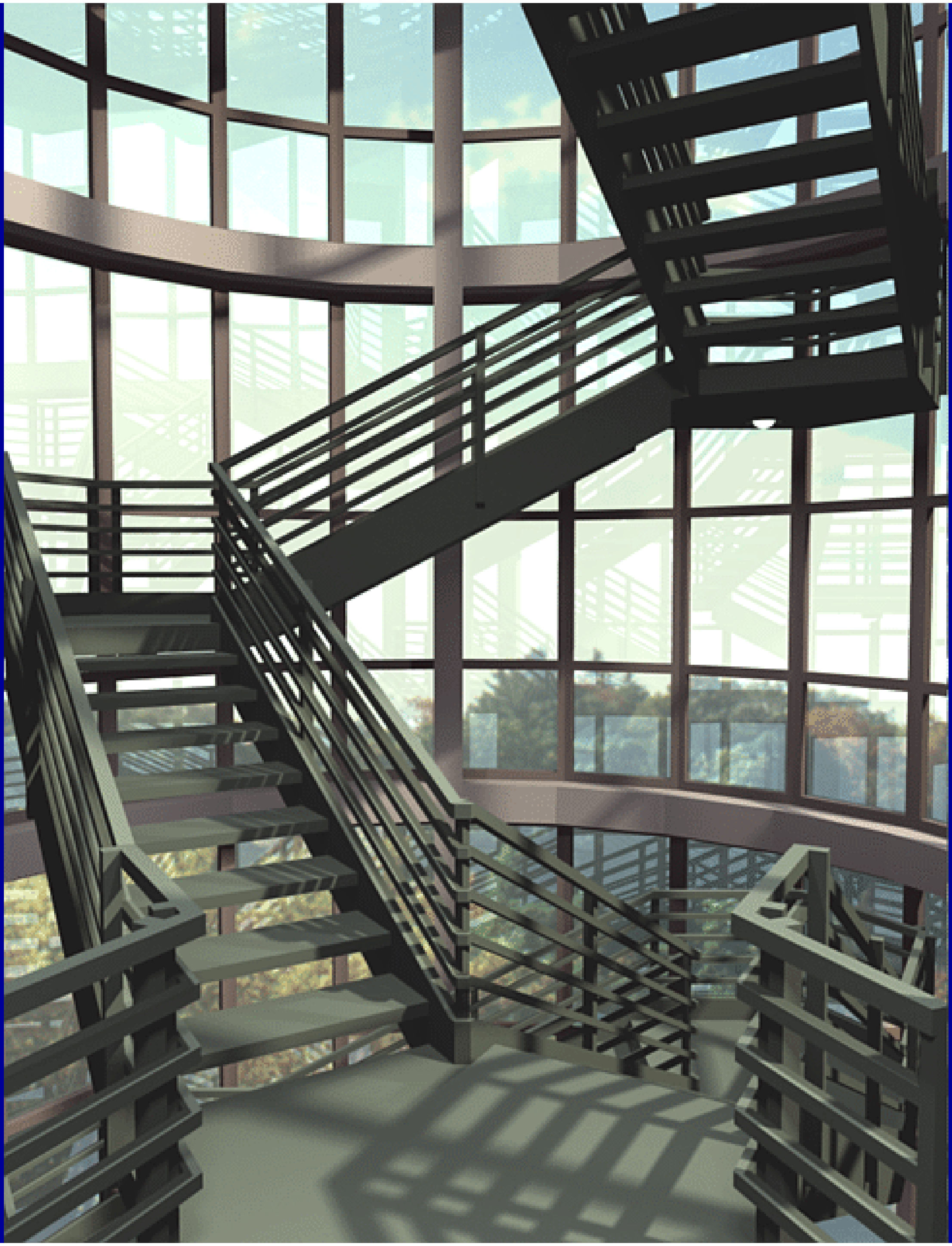
Produces very convincing effects and works very well for scenes with small numbers of reflecting/transmitting objects

Used in combination with other methods for extremely high-quality images

More Pretty Pictures

D. Lischinski

D. Lischinski



Summary: Classical Radiosity

- Divide all surfaces into patches.
- Calculate form factors between all patches.
  - Lighting and viewer independent
- Solve the radiosity equation
  - Viewer independent
- Render using standard 3D hardware.

Acknowledgements/Resources

- Demo that explores resolution and other parameters
  - http://www.mvpny.com/RadTutMV/RadiosityTut1MV.html
- T. Yeap (many great radiosity resources)
  - http://www.scs.leeds.ac.uk/cuddles/rover/main.htm
- Cornell graphics group (many pretty pictures)
  - http://www.graphics.cornell.edu/online/research/

Announcements

Written Assignment 2 due today

Questions?

Ray Tracing assignment out today, due April 10

- Start early—this is significantly more work than the previous two assignments
- The starter code is pretty simple—you're responsible for all program structure, so plan carefully before you start.
- Construct VERY simple test cases for debugging.
- Debug thoroughly before moving on to the next feature you want to add.