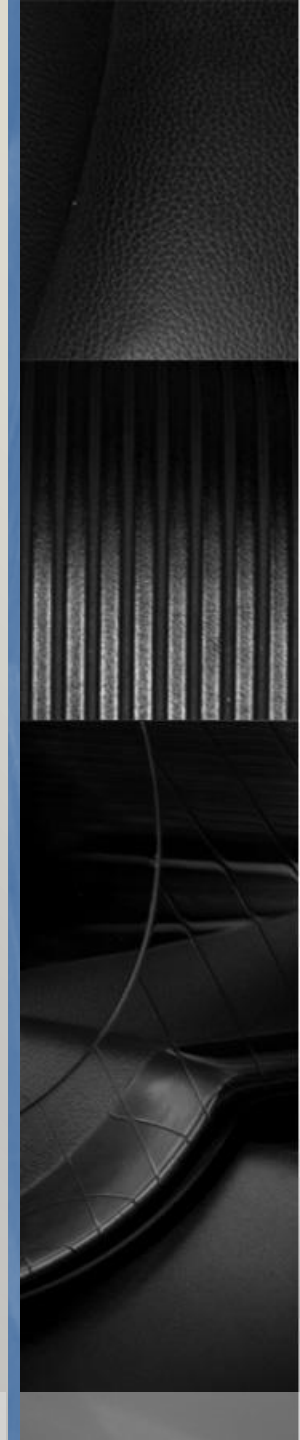
Web Based Graphics & Virtual Reality Systems

Ray tracing, Global Illumination and Radiosity



Recap

- So far we have discussed real-time rendering method using Z-buffer approach
- To render high quality image without the criteria of high framerate, there are a number of methods.
- In this lecture we will introduce a popular method called ray-tracing

Offline Rendering Methods

- Always involve complicated ray-surface interactions
  - Glossy(perfect) reflection
  - Multiple reflection
  - Refraction
  - Scattering

glossy / Multiple reflections

Global illumination?

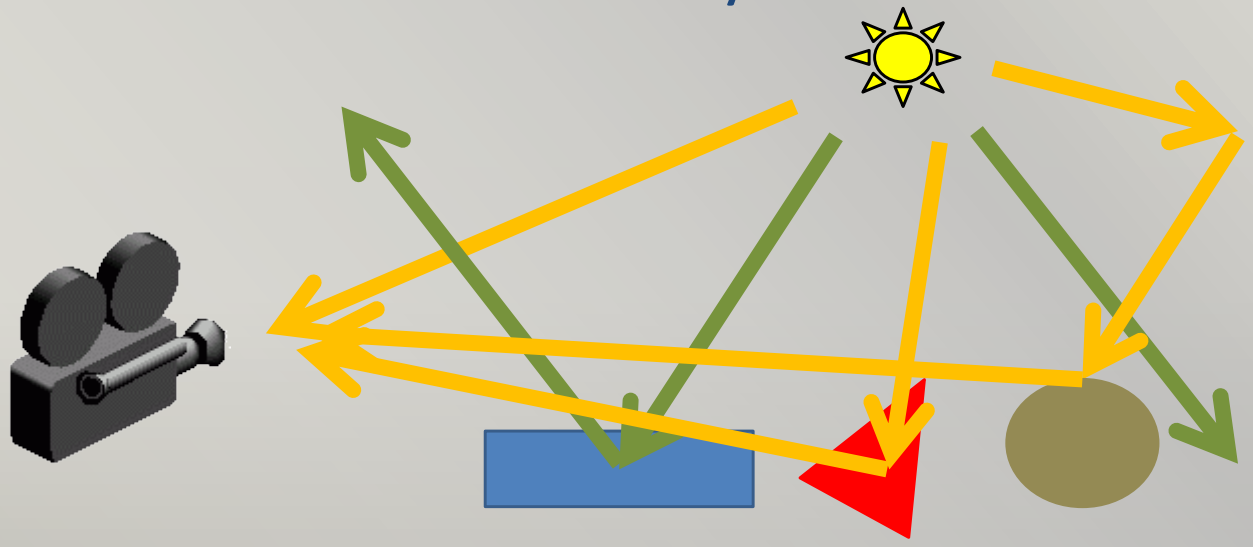
- Using Z-buffer approach is difficult
  - Though we can often use shading tricks to emulate the effect

Refraction

www.photos-of-the-year.com

Ray-tracing

- In real world, numerous of light rays bouncing around within the scene
- In rendering, we only care about those get into our camera/eye
- It is a wise choice to back-trace the rays from the camera



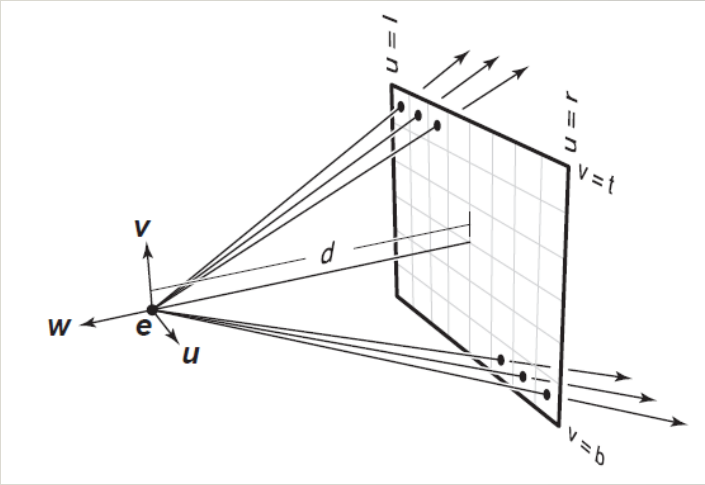
Ray-tracing

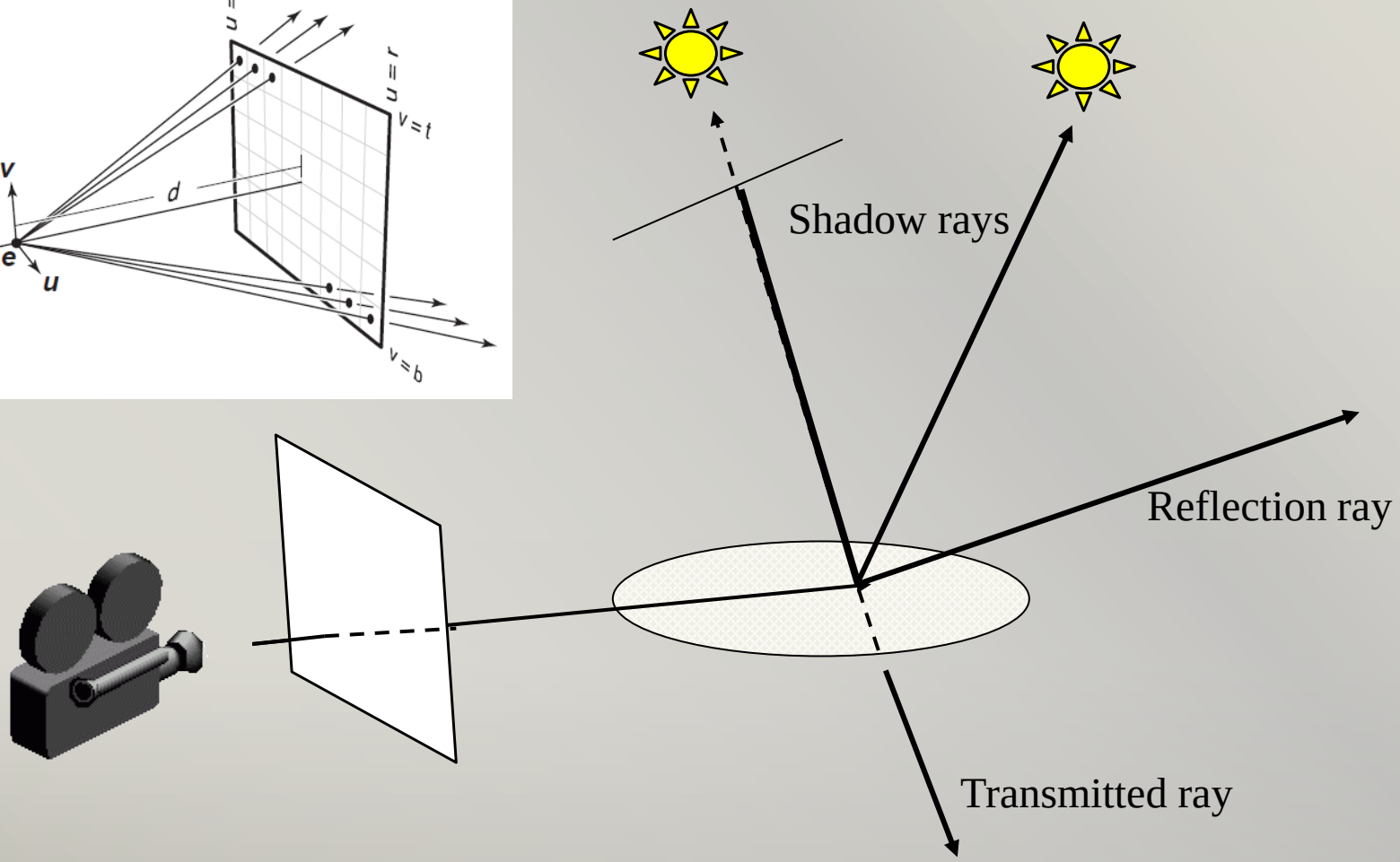
- Cast rays out from the eye
- Limited to only the viewport
- Each ray from a pixel, and determine what they hit first
  - Builds the image pixel by pixel, one at a time
  - Sum all the contributions to get the pixel color

Ray-tracing

- Cast additional rays from the hit point to determine the pixel color
  - Shadow rays toward each light. If they hit something, then the object is shadowed from that light, otherwise use “standard” model for the light
  - Reflection rays for mirror surfaces, to see what should be reflected in the mirror
  - Transmission rays to see what can be seen through transparent objects

Ray-tracing


u = l
u = r
v = t
v = b
v
d
w
e
u


Shadow rays
Reflection ray
Transmitted ray

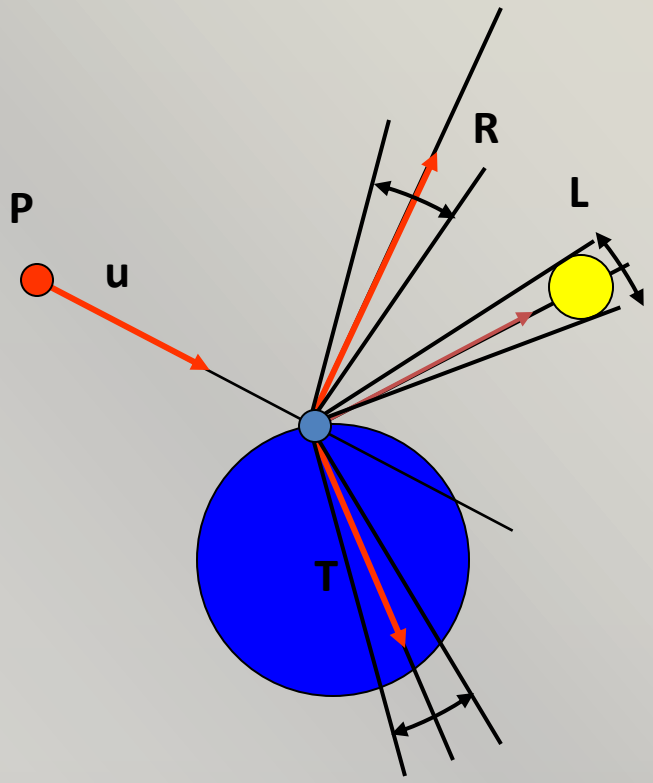
Pseudo-code of Ray-tracing

```
for each pixel do
    compute viewing ray
    for each object do
        compute ray triangle interception
        if intercept then
         compute light model with surface normal n and
          light direction
          set pixel color to value computed from light
          model
           skip to next pixel
```

Here we only consider single reflection

Reflection in ray tracing

- When multiple reflections is considered
- More rays will be created at the interception point
  - Reflection to light source
  - Reflection to other surfaces
  - Refraction
  - etc.



Difference to Z-Buffer Approach

Z-Buffer approach (Real-time rendering)

- Render in units of triangles
- Things are projected on the screen, then paint
- Less flexible

Raytracing approach

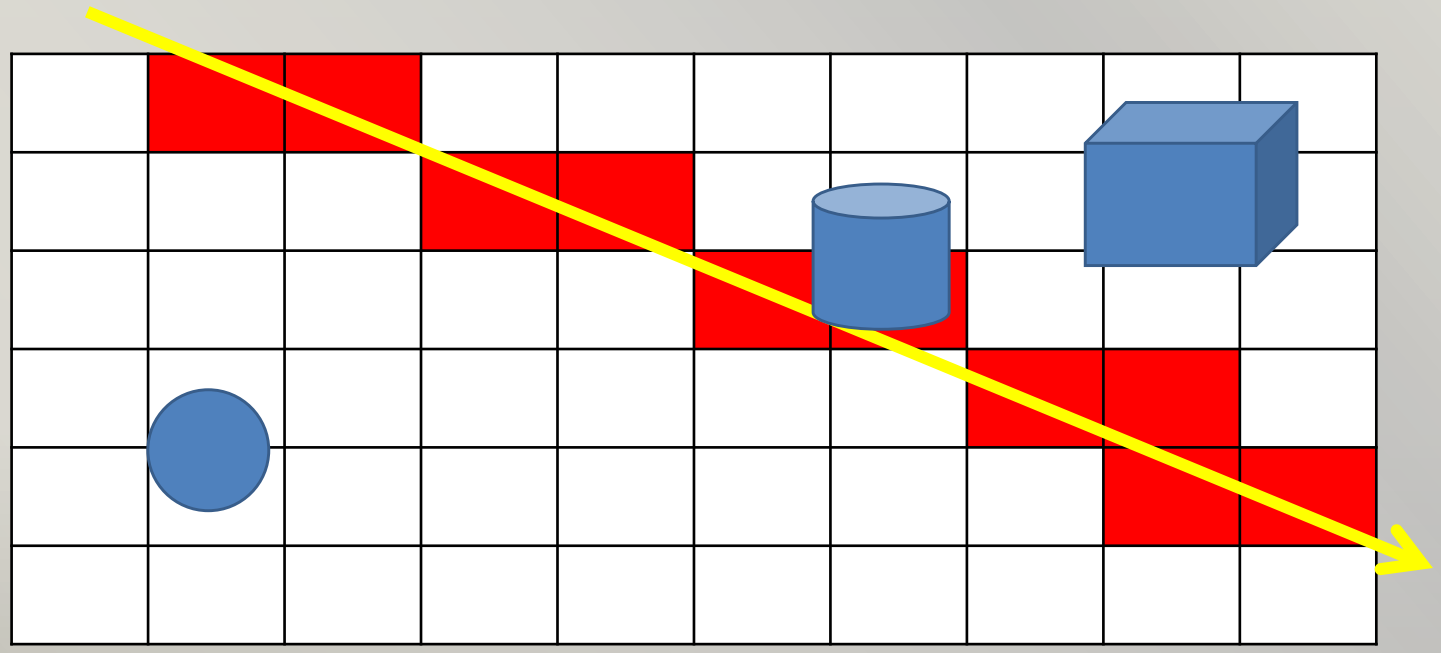
- Render in units of light rays/pixels
- Simulate light traversal in space
- More flexible to render complex effects

Problem with Ray-Tracing

- High complexity when the scene contains a large number of objects
- How can we find potential intercepting object more quickly ?

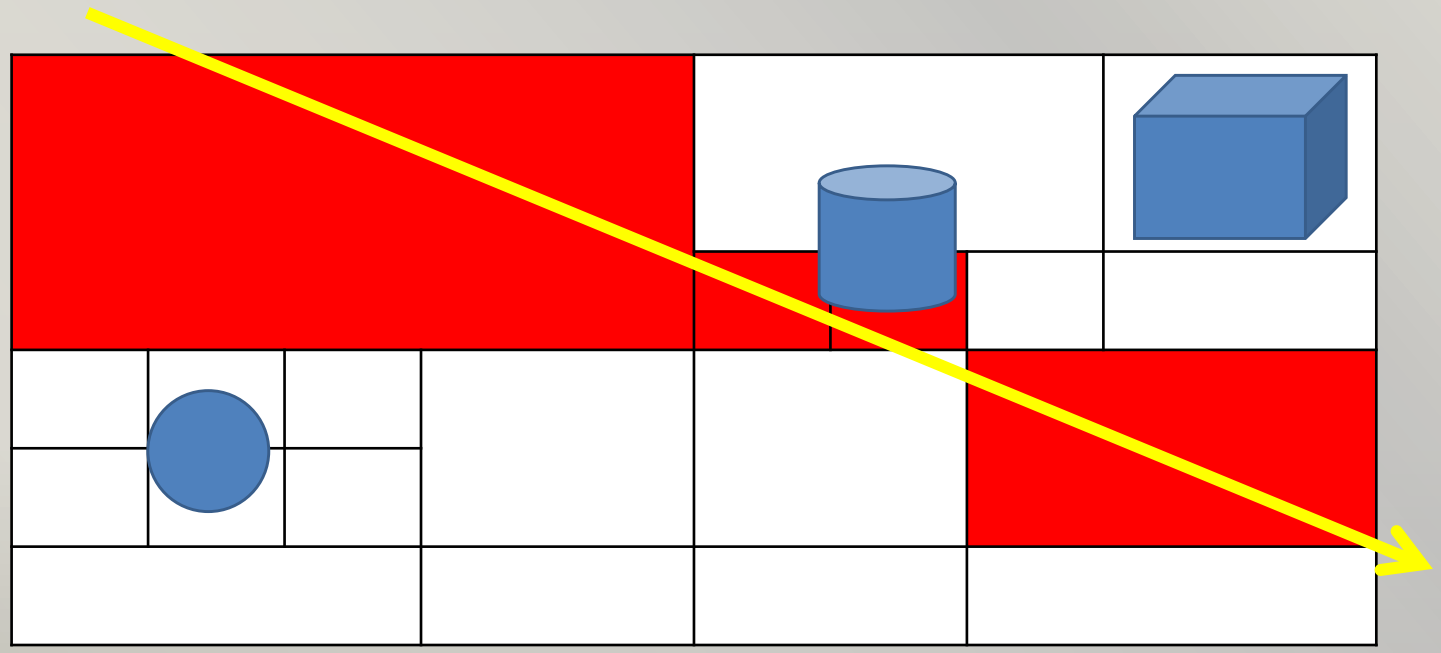
Spatial Partitioning

- A widely used method for fast searching in space
- Simplest approach is to divide space into uniform box regions



Spatial Partitioning

- For large vacant region, it is a waste to divide them into smaller regions
- Search become faster if we have less regions to check!!



Spatial Partitioning

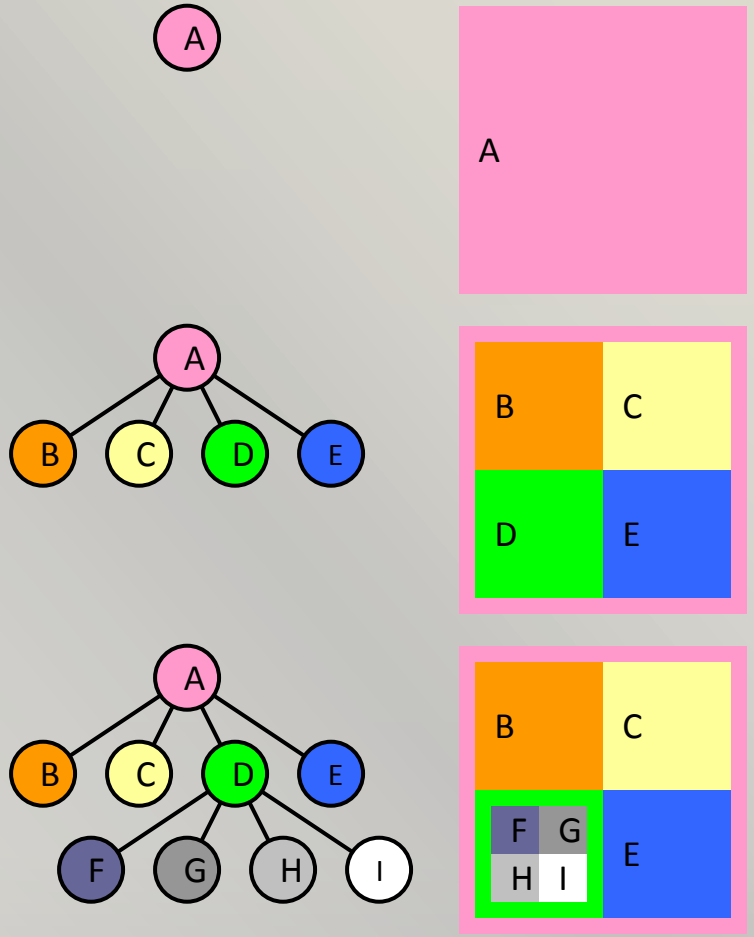
- Approach
  - Divide the space into many sub-regions
    - Usually in a sub-divided structure
  - If the ray across this sub-region, further search the smaller sub-regions inside
  - Until the smallest region is found, then do an interception detection to all objects inside

Spatial Partitioning

- Therefore, the sub-division of regions form a tree like structure
- Common methods used in Graphics
  - Quadtree / Octree
  - Binary Space Partitioning (BSP) tree
  - Kd tree
- Each object is therefore being arranged to a node of the tree

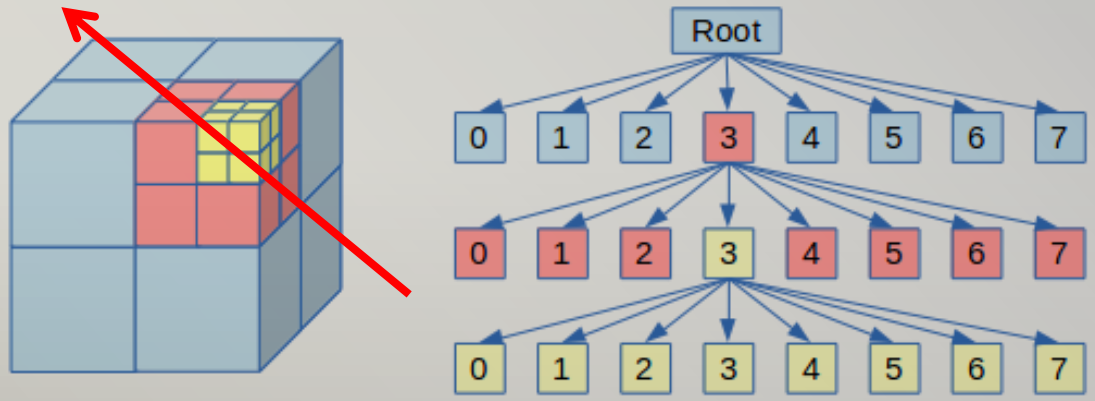
Quadtree / Octree

- Every time it divide a 2D space into 4 quadrants or a 3D space into 8 quadrants
- Subdivision occurs only at branch which still contain large number of object/facets
  - By a user defined threshold



Quadtree / Octree

- A recursive subdivision of space
- Each level, we have 8 smaller box regions
- If a ray is across a certain node (detect by ray-box interaction), we try to search its sub-node until leaf
- Only do detail interceptions with objects inside leaf node

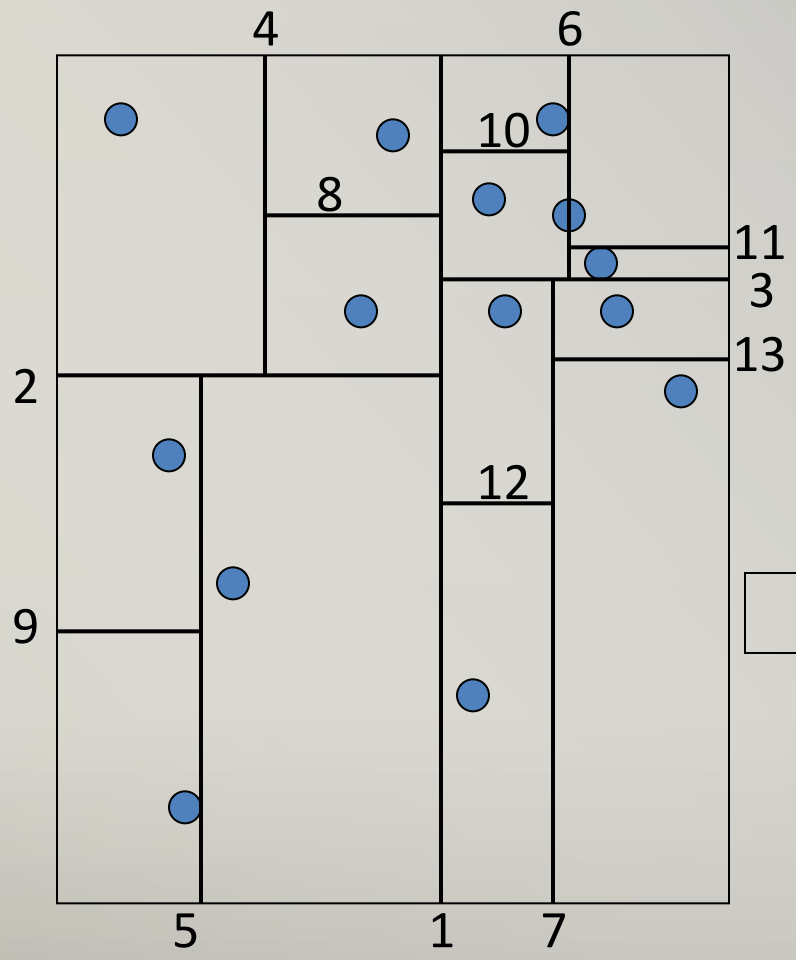


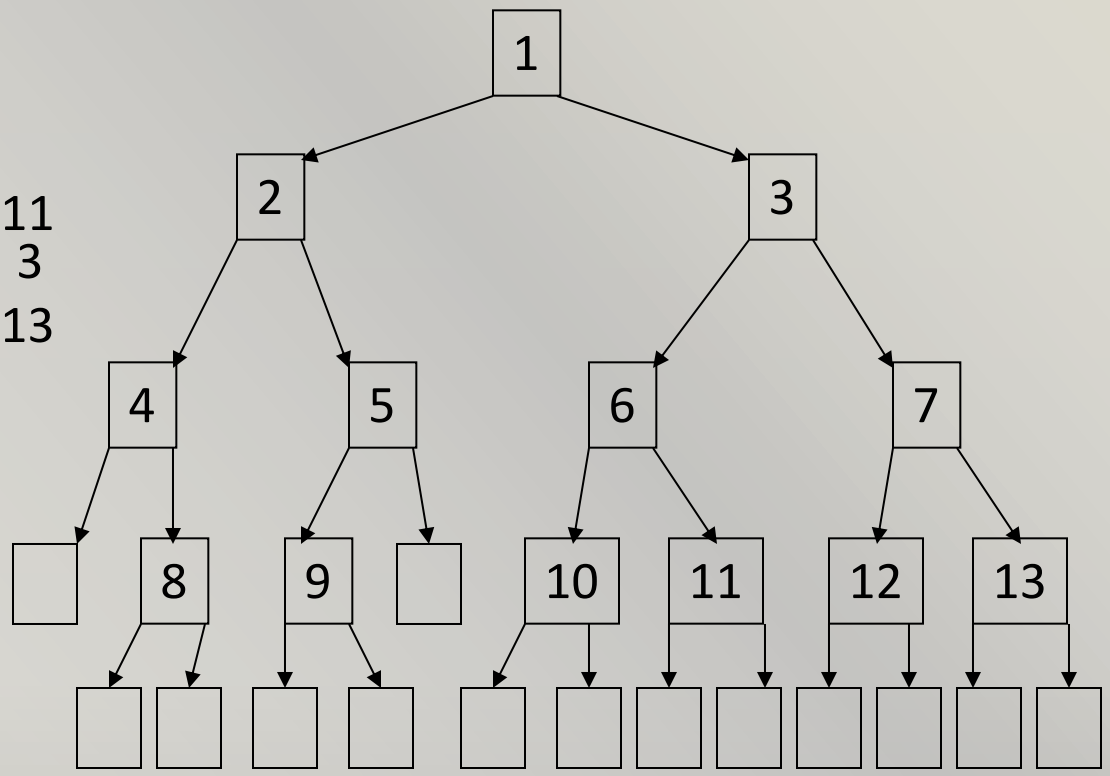
Kd-tree

- A kd-tree is a tree that divide space into two in each time (/level)
- It has the following properties:
  - Each node represents a rectilinear region (faces aligned with axes)
  - Each node is associated with an axis aligned plane that cuts its region into two, and it has a child for each sub-region
  - The directions of the cutting planes alternate with depth

    First cut perpendicular to x-axis, then perpendicular to y, then z and then back to x (some variations apparently ignore this rule)

Kd-tree Example





Kd-tree

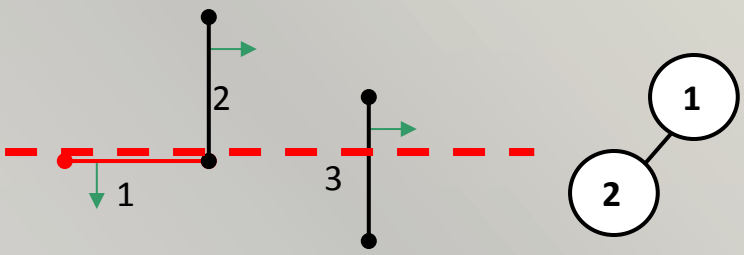
- Kd-trees generalize octrees by allowing splitting planes at variable positions and dimension
  - Note that cut planes in different sub-trees at the same level do not have to be the same
- Kd-trees work well when axis aligned planes cut things into meaningful cells
- Application not limited to graphics but also in multidimensional data query

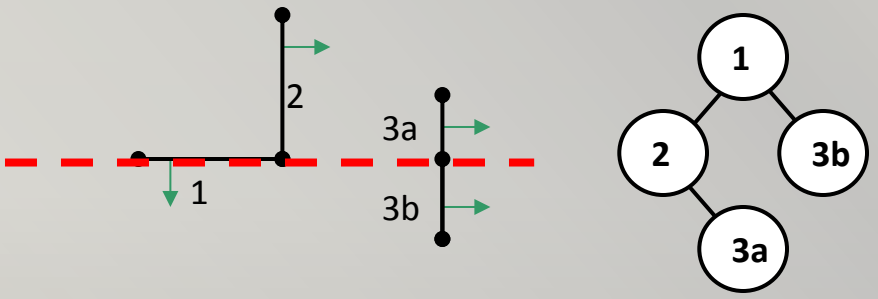
BSP-tree

- A binary tree is constructed
- Divide into two spaces each time
- Space partitioning uses available facets(triangles) instead of axis-aligned cut planes
  - Differ from Quadtree/Octtree which use oriented cut planes (i.e. XY, YZ or XZ planes)

Building of BSP-Tree

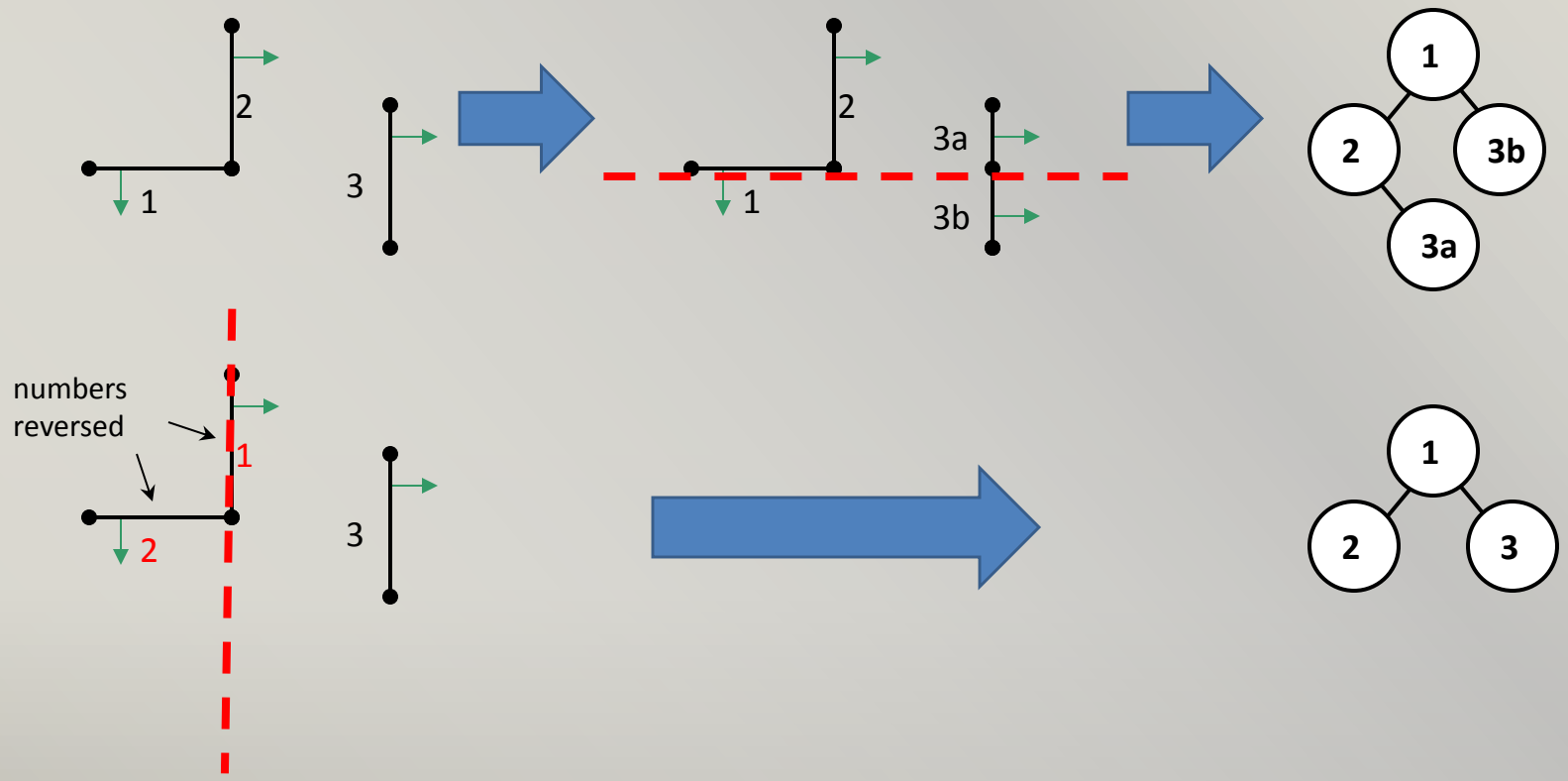
- Choose one triangle from any object in the scene
- Use it to form a plane
- Subdivide the space with in this plane





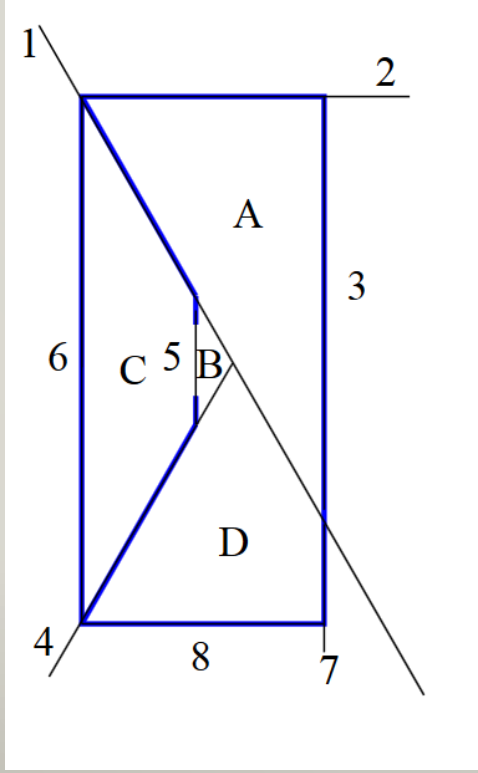
Building of BSP-Tree

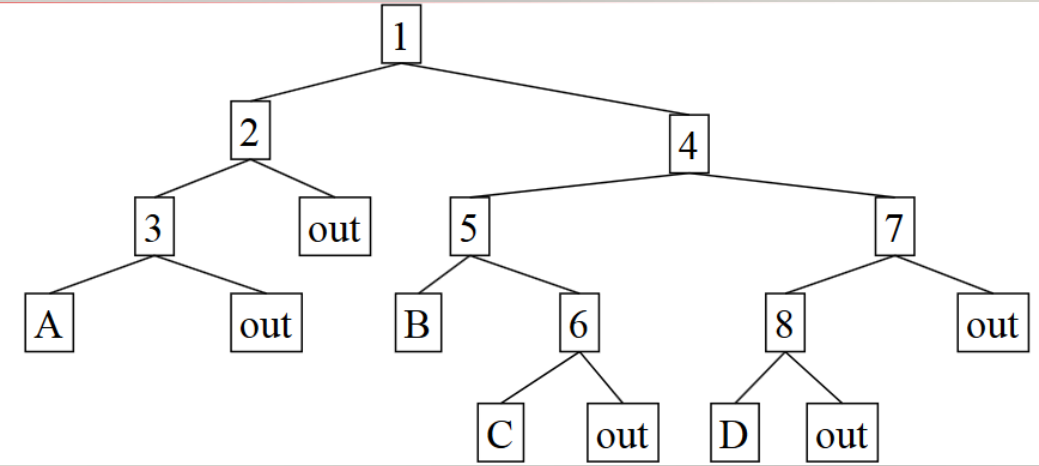
- The choice of subdivision order will affect the result



BSP-Tree

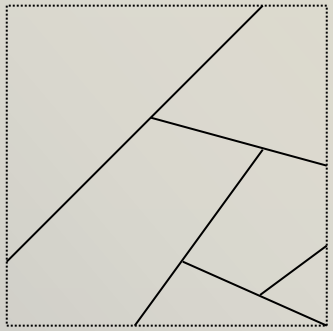
- An example of BSP-tree



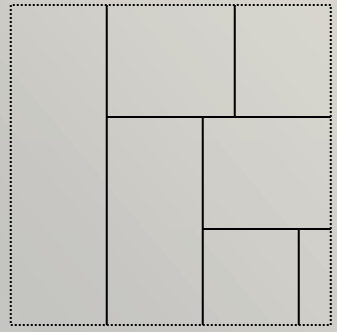


BSP Tree, kd Tree and Octree

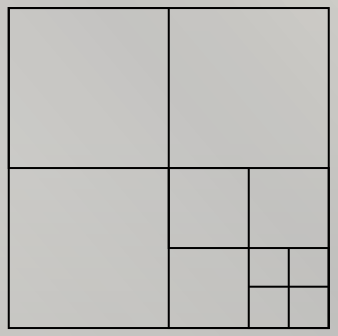
	BSP Tree	Kd Tree	Octree
Division of space in each level	2	2	8
Cut planes	Polygon aligned	Axis aligned	Axis aligned
Space represen ted by each leaf node	Half space to the cut plane	Half space to the cut plane	Cube space



BSP-tree



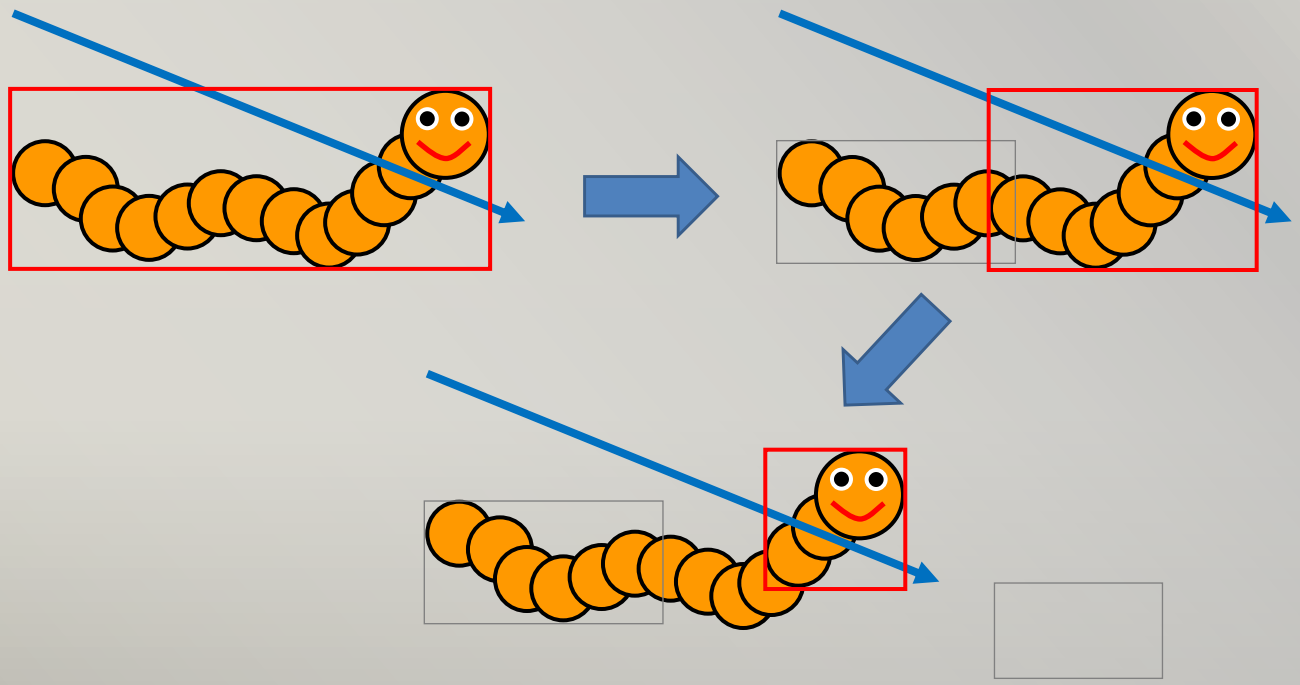
kD-tree



Octree

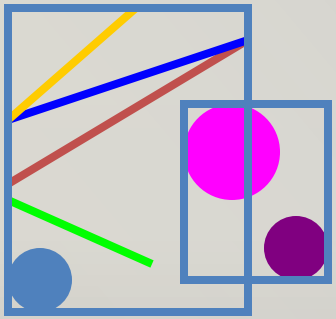
Bounding Volume Hierarchy

- BVH group objects within a bounding volume
- Use the bounding volume for ray-interaction checking first
- From large bounding box to smaller ones builds a hierarchy



Bounding Volume Hierarchies

- Tightly fits objects
- Redundant spatial representation

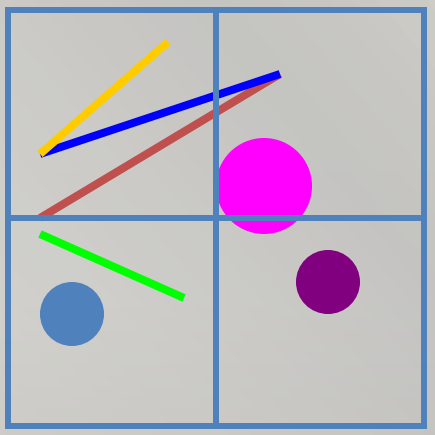


Volumes overlap multiple objects

Spatial Partitioning

Tightly fills space

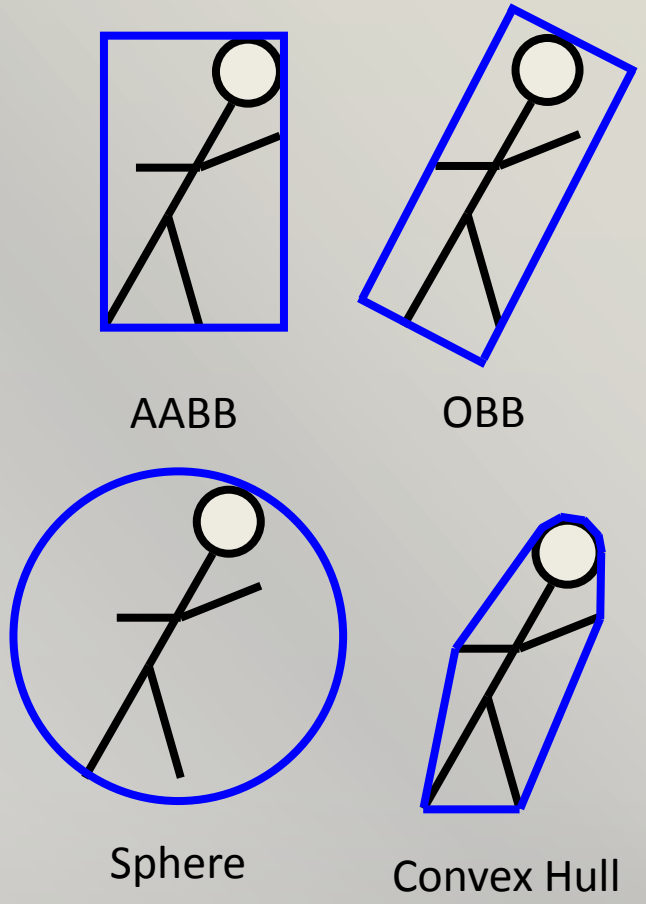
Redundant object representation



Objects overlap multiple volumes

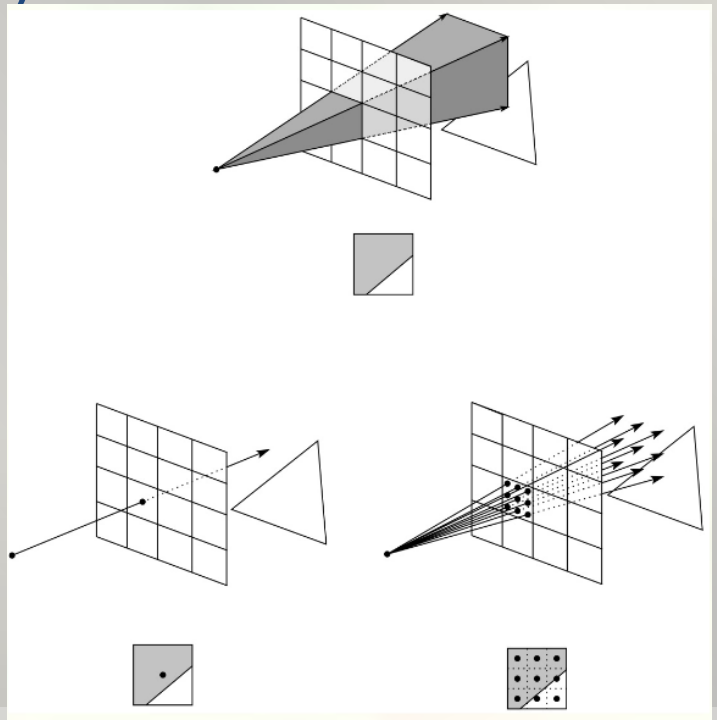
Building Bounding Volume Hierarchies

- Choose a bounding volume type
  - Axis-aligned bounding box (AABB)
  - Oriented bounding box (OBB)
  - Sphere
  - Convex Hull
- Choose a clustering strategy
  - Top-down:
    how do we partition objects among children?
  - Bottom-up:
    how do we find leaf clusters and merge into parents?

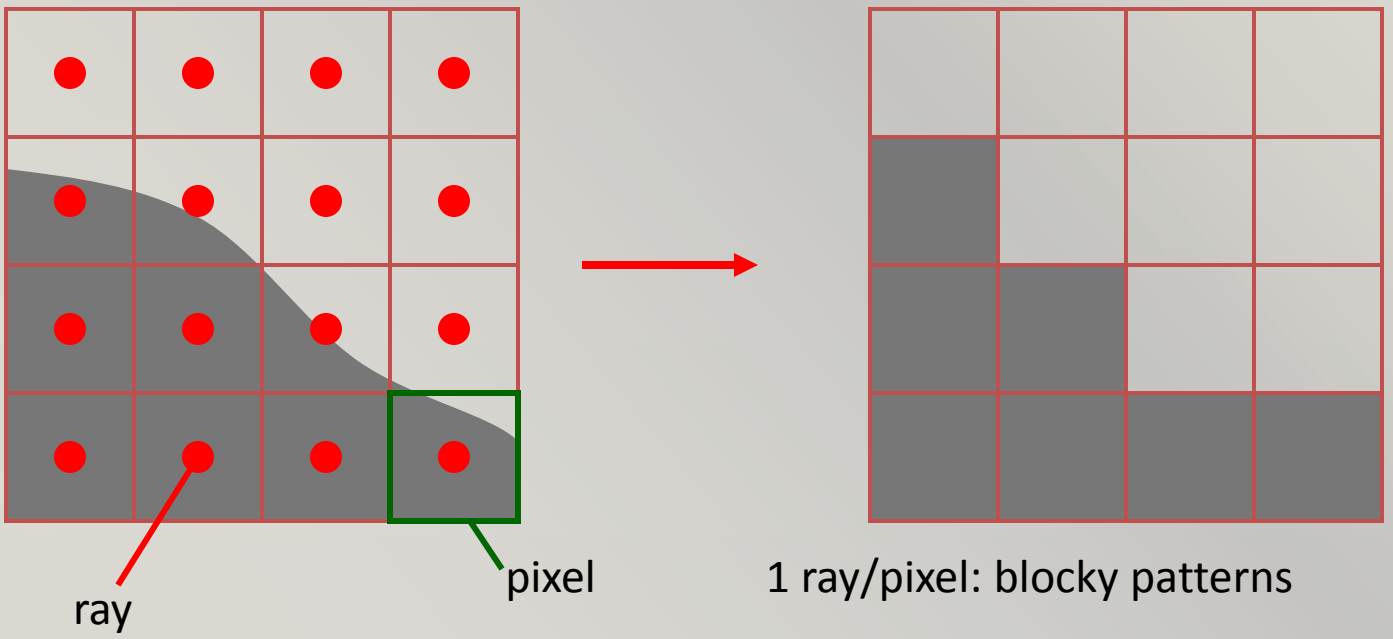


Problem with Ray-Tracing

- A naïve raytracing implementation may cause visual artifacts, e.g. aliasing
- Problem is we use single ray's result as a solid angle's result (see the pyramid volume)
- **Supersampling** can be a solution
  - More than 1 ray from each pixel
  - But more computation

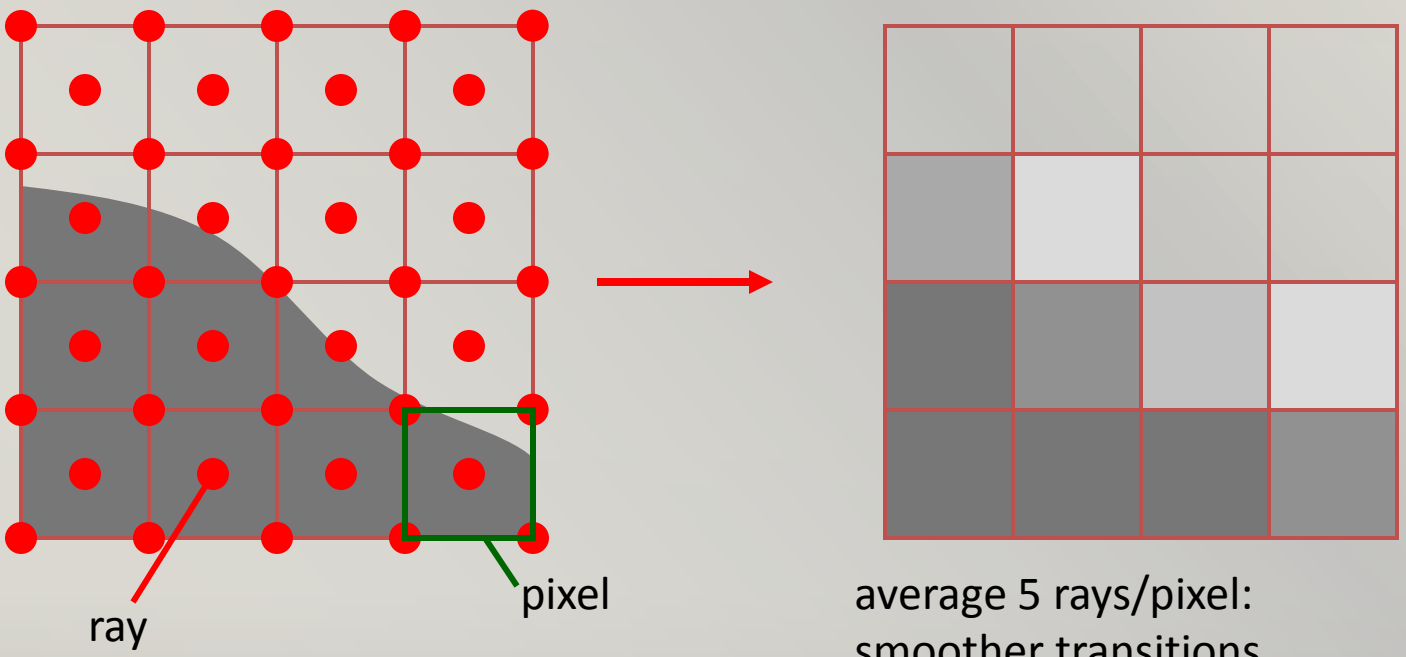


Aliasing Problem in Ray-tracing



Anti-Aliasing in Ray-tracing

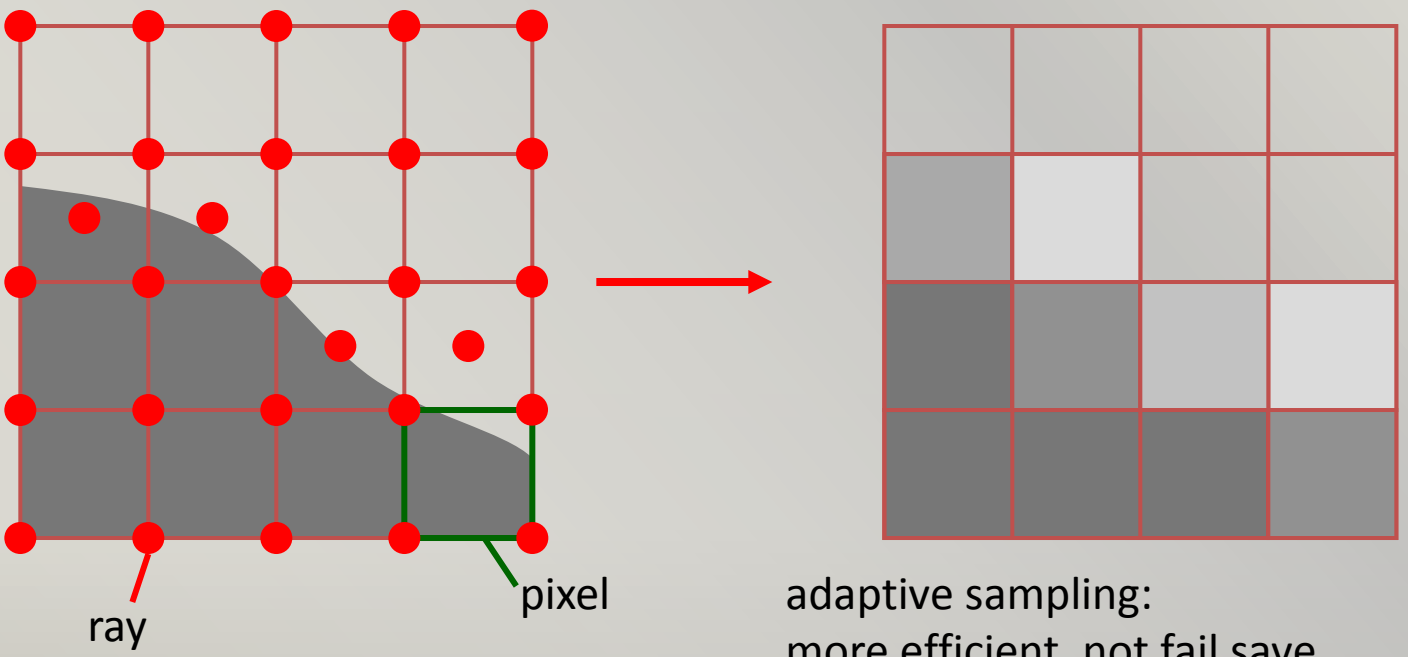
- Supersampling



average 5 rays/pixel:
smoother transitions

Anti-Aliasing in Ray-tracing

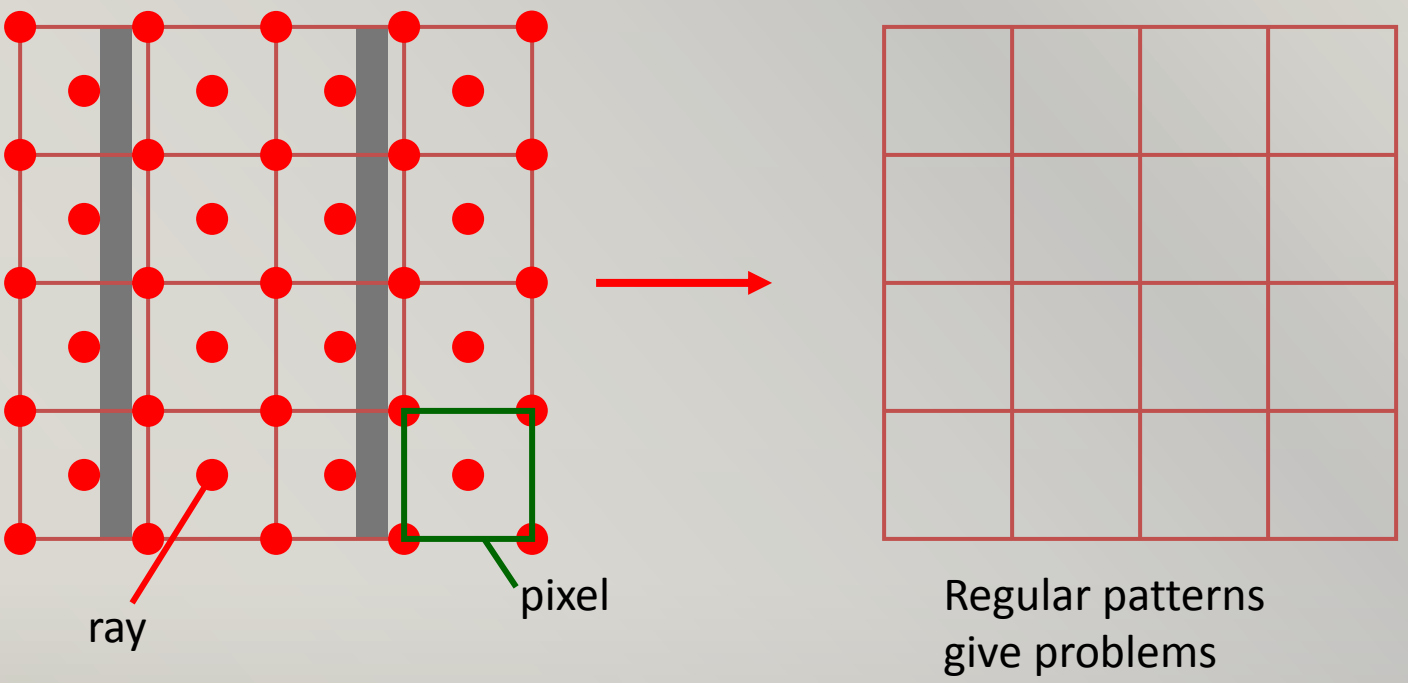
- Adaptive sampling
  - Only supersample at boundries



adaptive sampling:
more efficient, not fail save

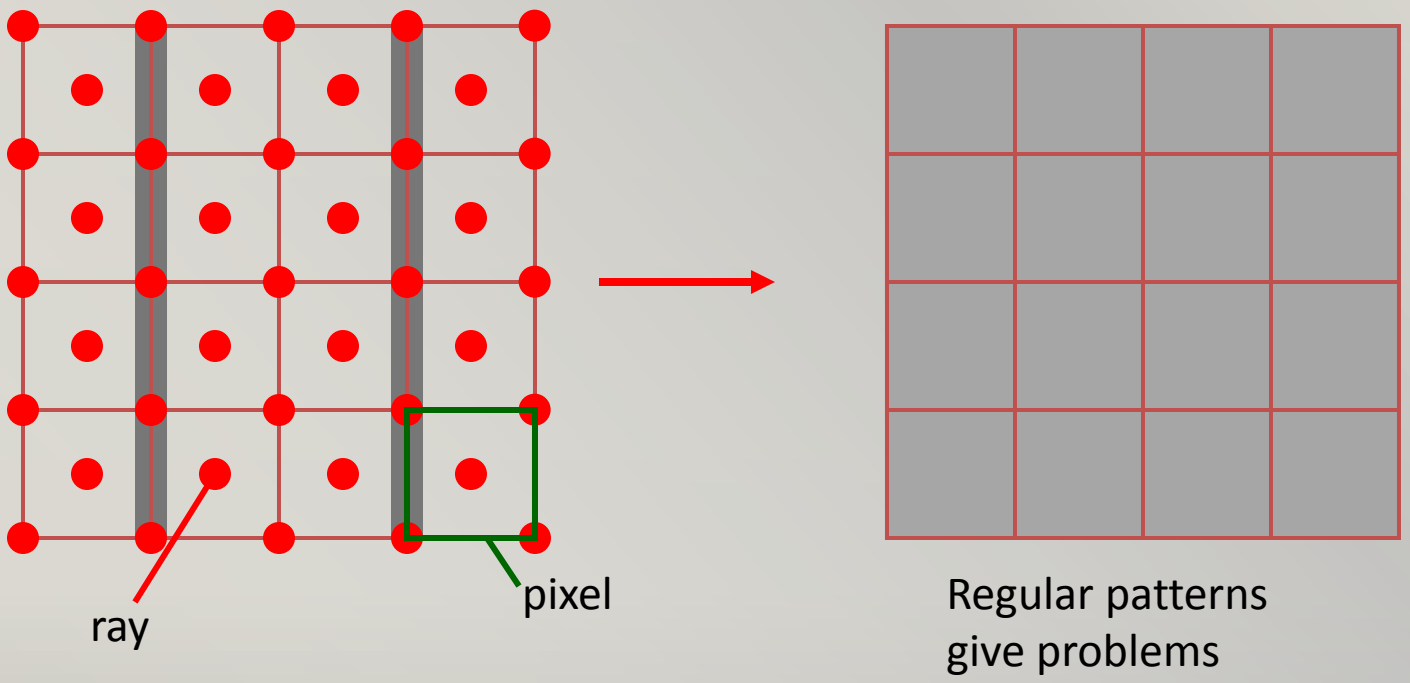
Problem of Supersampling

- Regular grid supersampling can cause potential problem



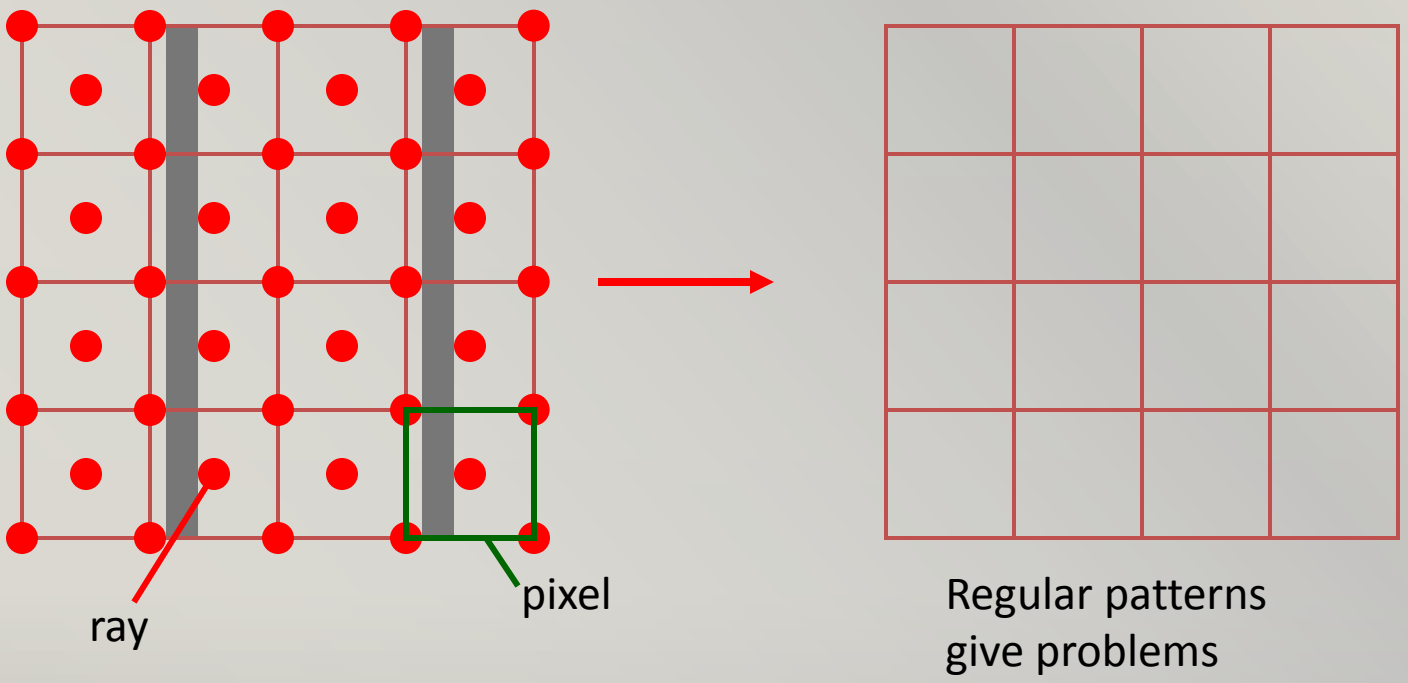
Problem of Supersampling

- Highly sensitive to minor difference



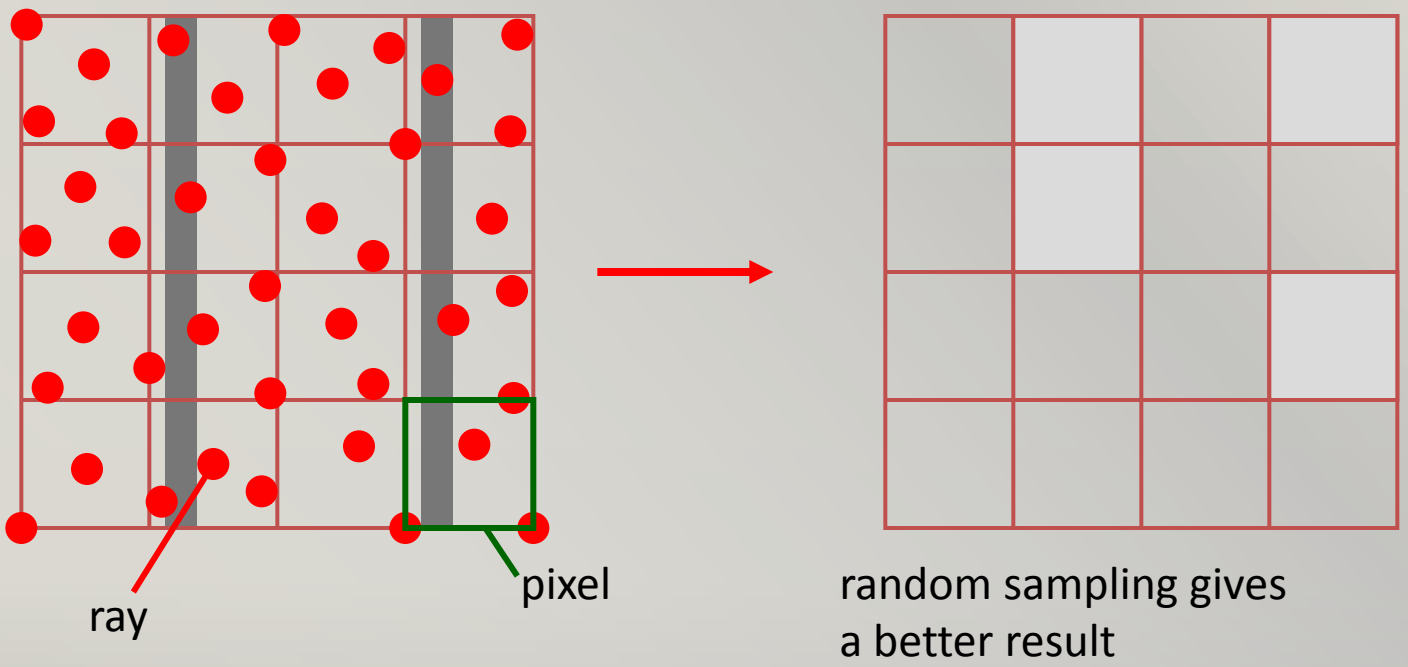
Problem of Supersampling

- Also, flickering can occur



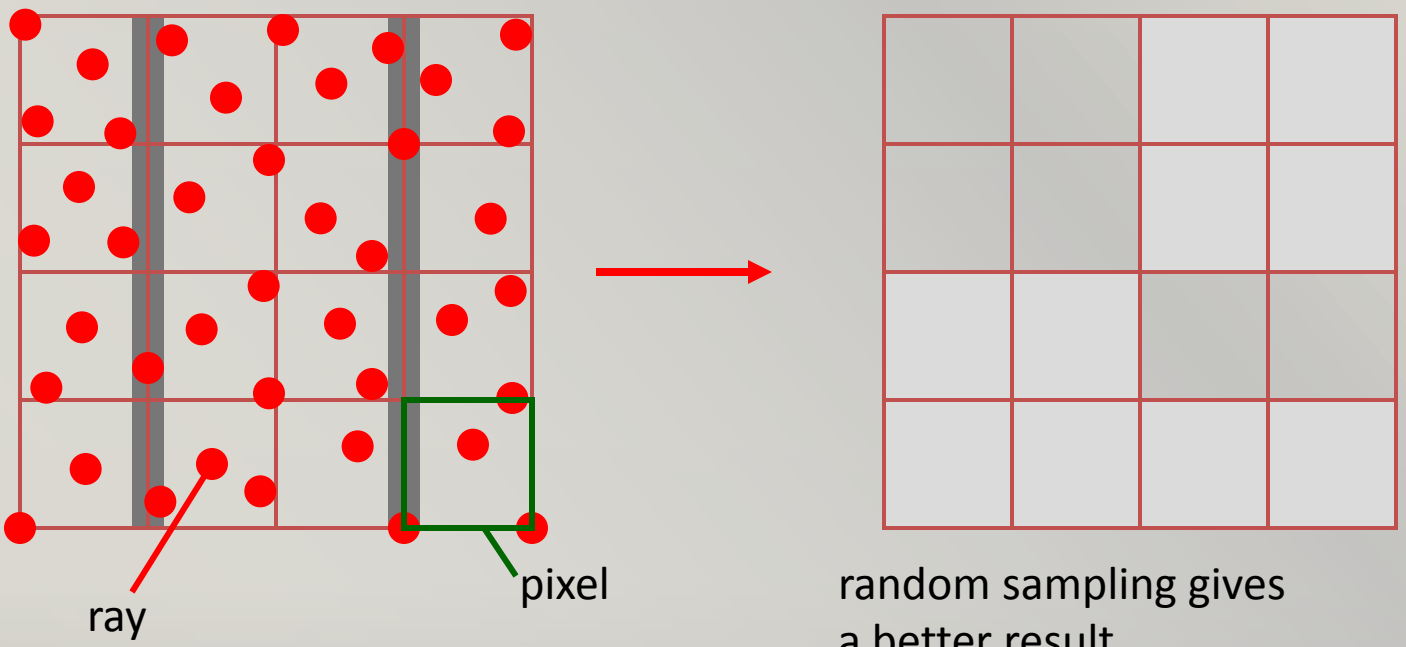
Distributed ray tracing

- By introducing randomness in the sampling



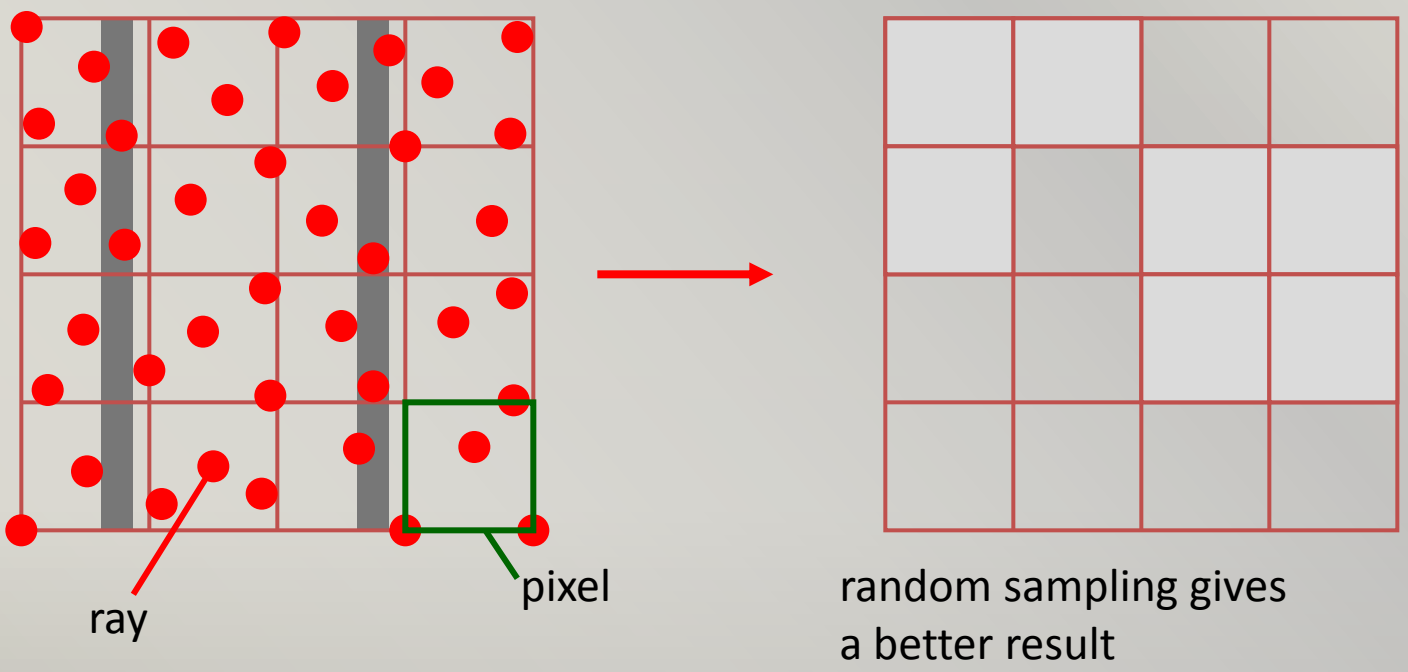
Distributed ray tracing

- The problem of flickering and aliasing is minimized



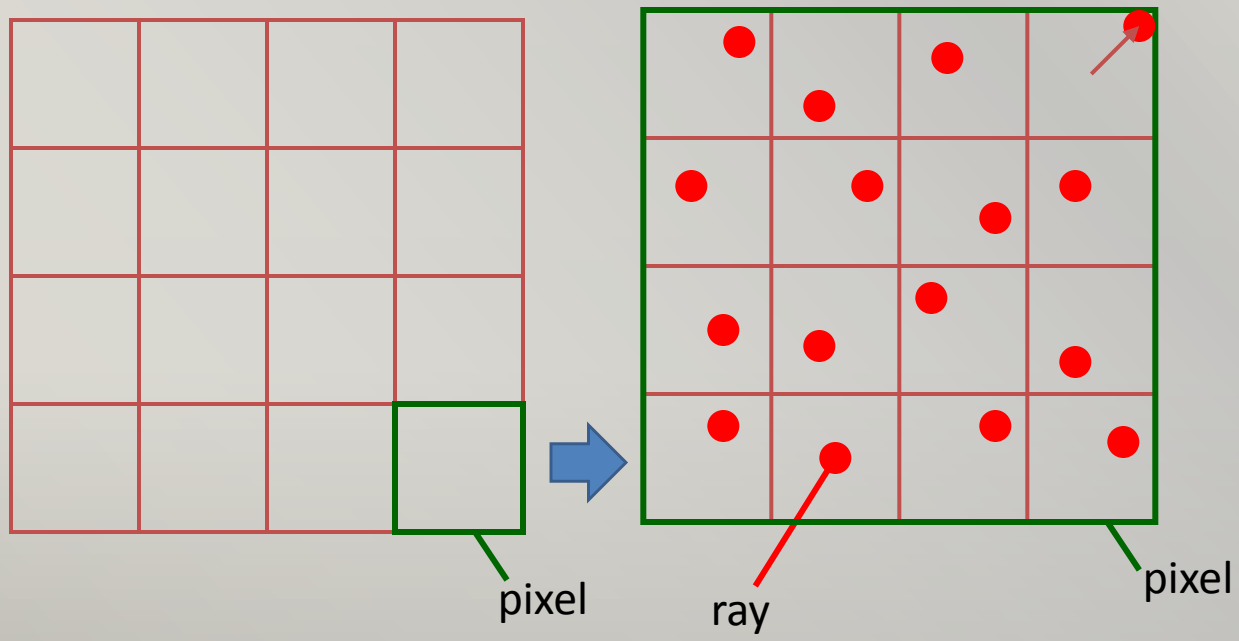
Distributed ray tracing

- With the same number of samples, distributed ray tracing do better in antialiasing



Distributed ray tracing

- A kind of Monte Carlo sampling (from statistics)
- Typically 16 rays/pixel are used
- Each ray is shifted along a small random vector called **jittering**

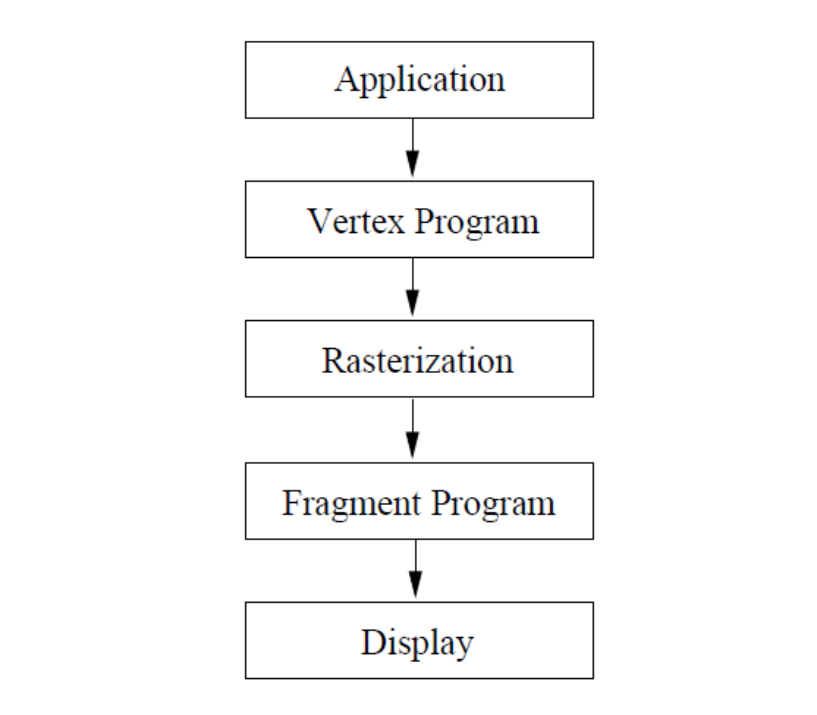


Real-time Raytracing

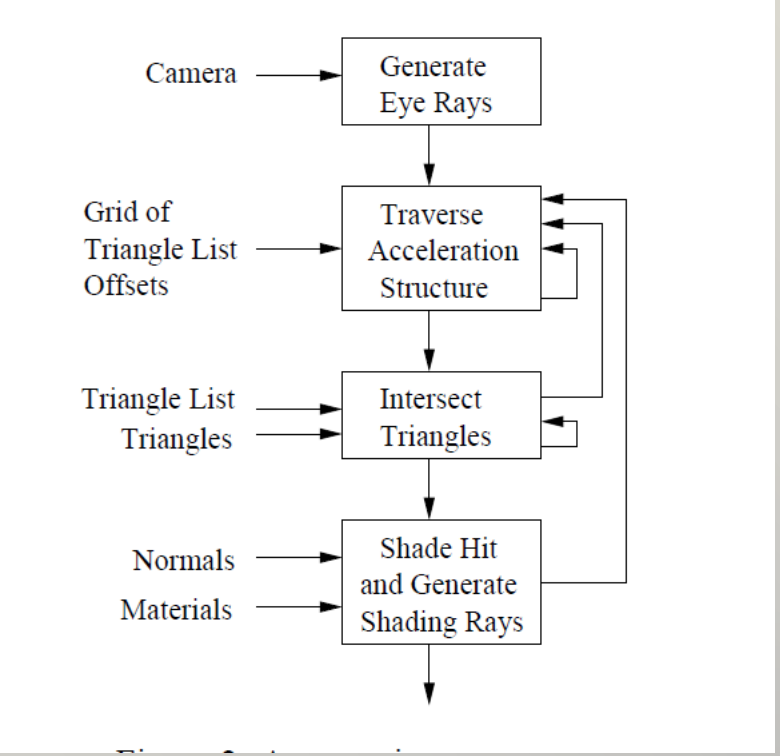
- Raytracing is usually slow, it consider multiple reflection
  - Generate large number of rays
- Fortunately, each ray works quite similarly
- Accelerate with GPU is therefore a trivial choice
- However, reflected rays depends on incident rays
- Multiple passes are thus required

Ray Tracing on Programmable Graphics Hardware

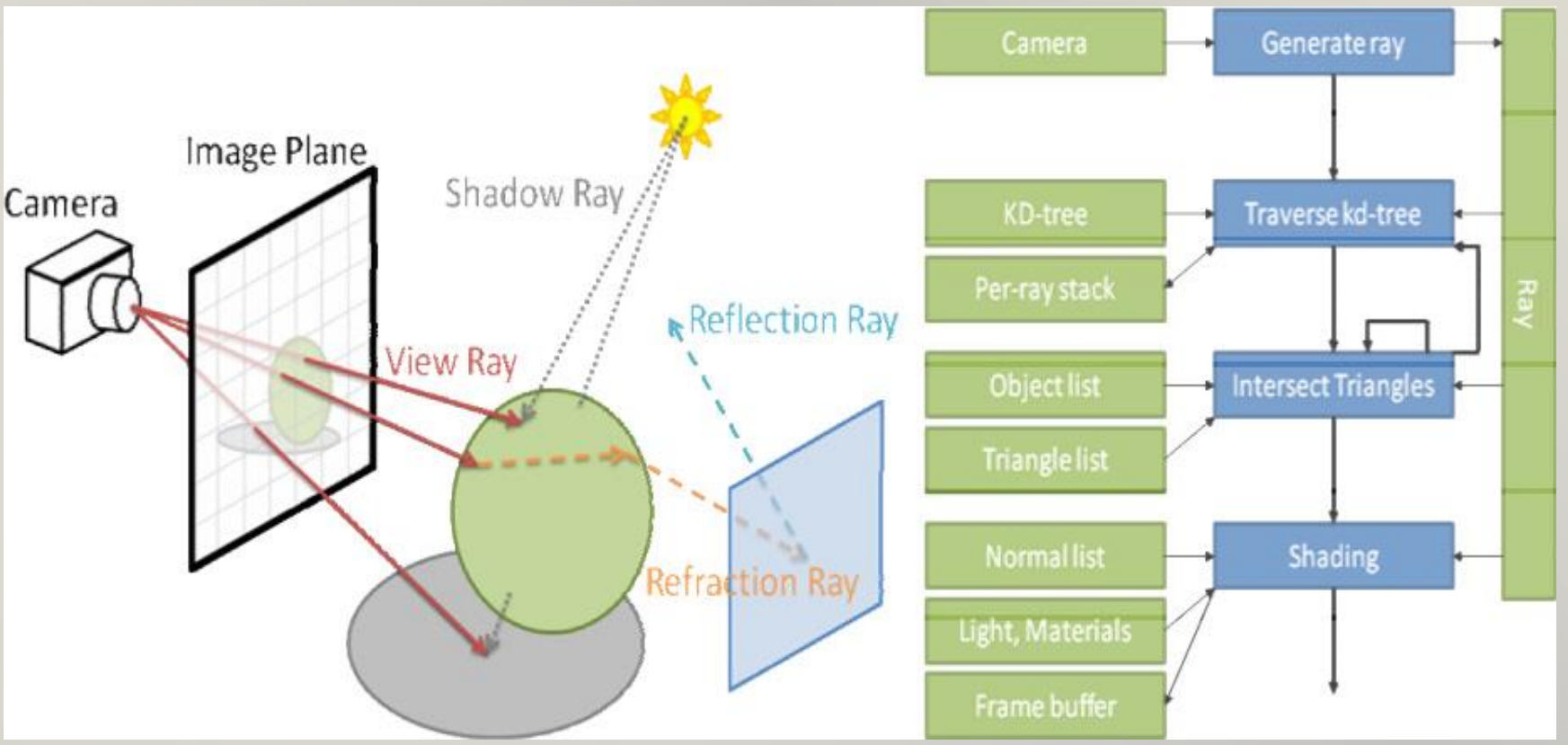
- GPU Pipeline



- Streaming Ray Tracing

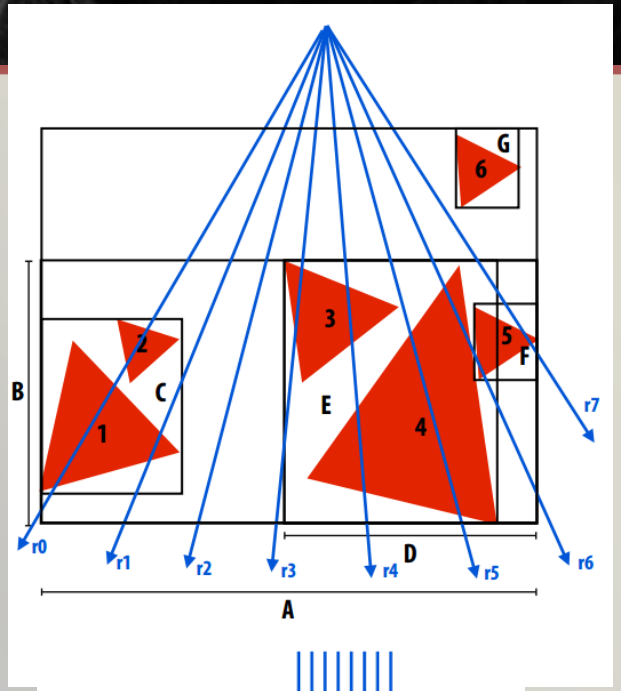


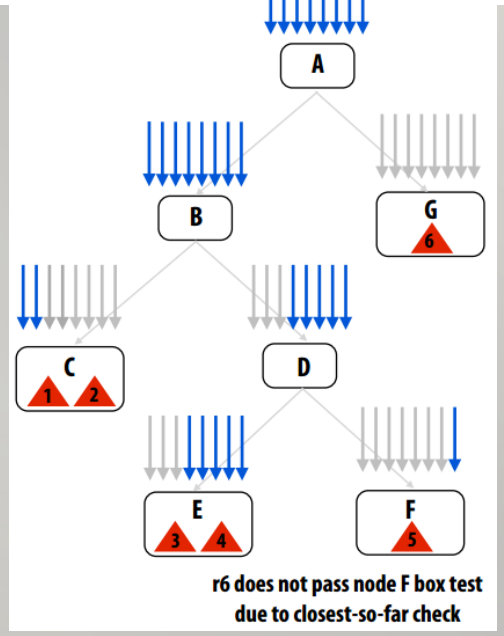
Ray Tracing on Programmable Graphics Hardware



Kd Tree Traversal

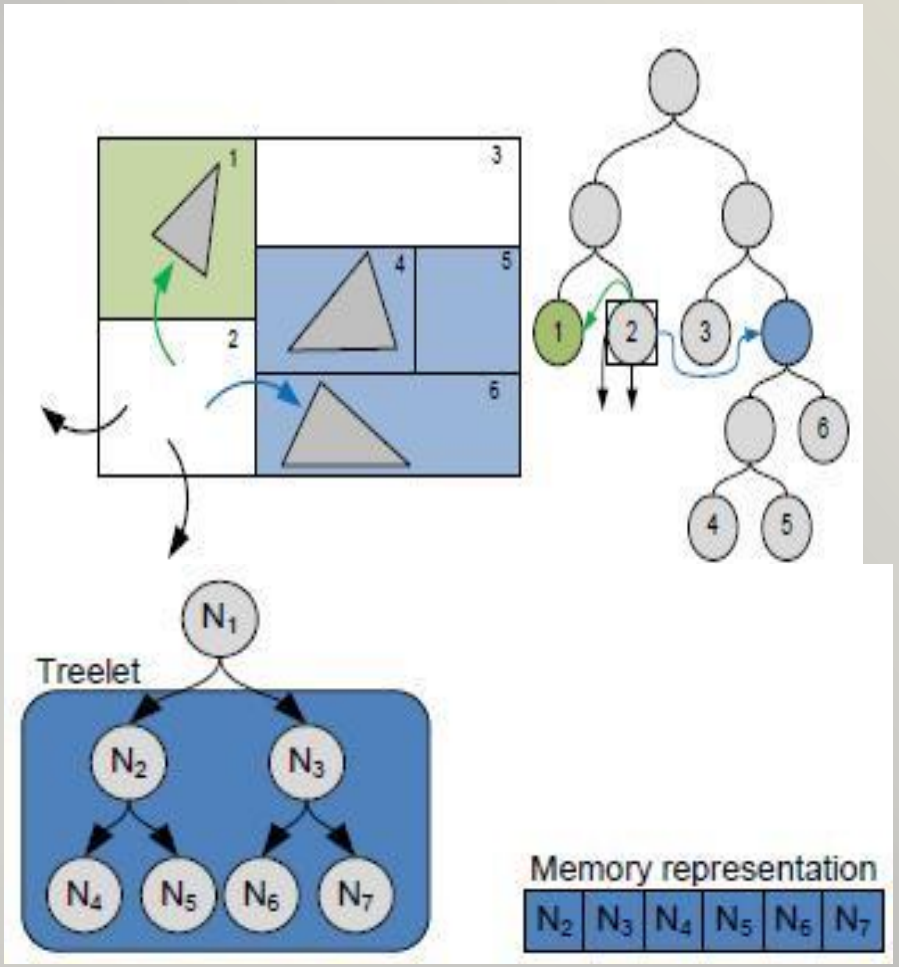
- Most time consuming part, thus, potential for optimization
- Kd Tree Traversal Issues
  - Single Ray vs Packet
    Run many rays in parallel
  - Stack vs Stackless
    - ✓ Avoid traversal of kd-tree through a stack
    - ✓ Stackless was good since implementing per ray stack was prohibitive on GPUs
    - ✓ CUDA solves this by general DRAM addressing





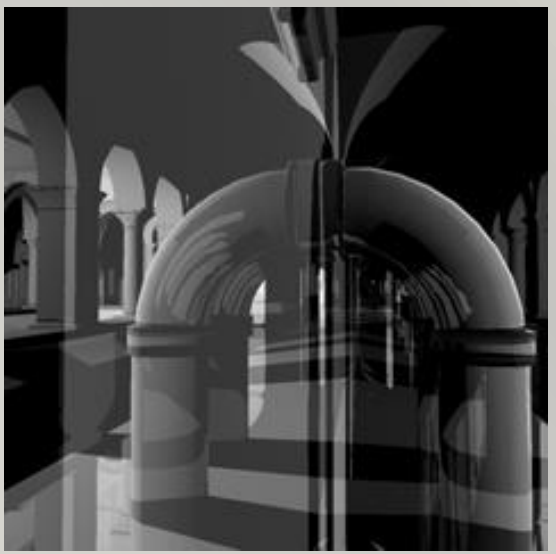
Stackless Kd-Tree Traversal

- Eliminate the need of maintaining a stack while traversal by making use of rope links for neighboring cells
- Optimized tree storage:
  - Geometry data in leaf with its AABB and its ropes to increase the chance of having the data in shared memory
  - Non leaf nodes stored as tree-lets, allows for memory coherence

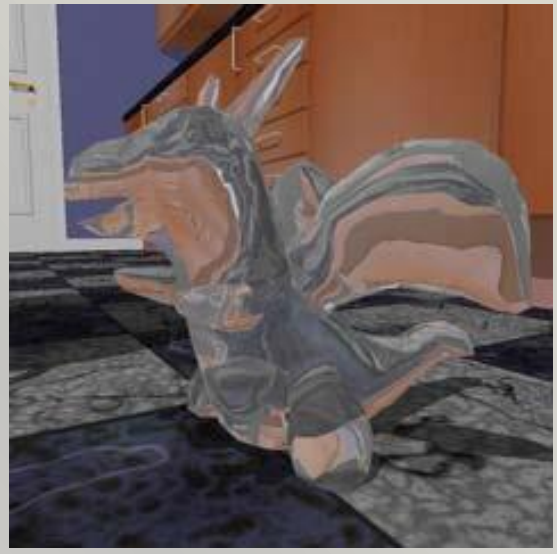


Results

One Pass Shadow: 18.1 fps
Two Pass Shadow: 20.1 fps
1-bounce reflection: 9.1 fps
2-bounce reflection: 5.9 fps
3-bounce reflection: 3.9 fps



One Pass Shadow: 21.0 fps
Two Pass Shadow: 23.9 fps
1-bounce reflection: 11.3 fps
2-bounce reflection: 7.2 fps
3-bounce reflection: 5.0 fps



Global Illumination & Radiosity

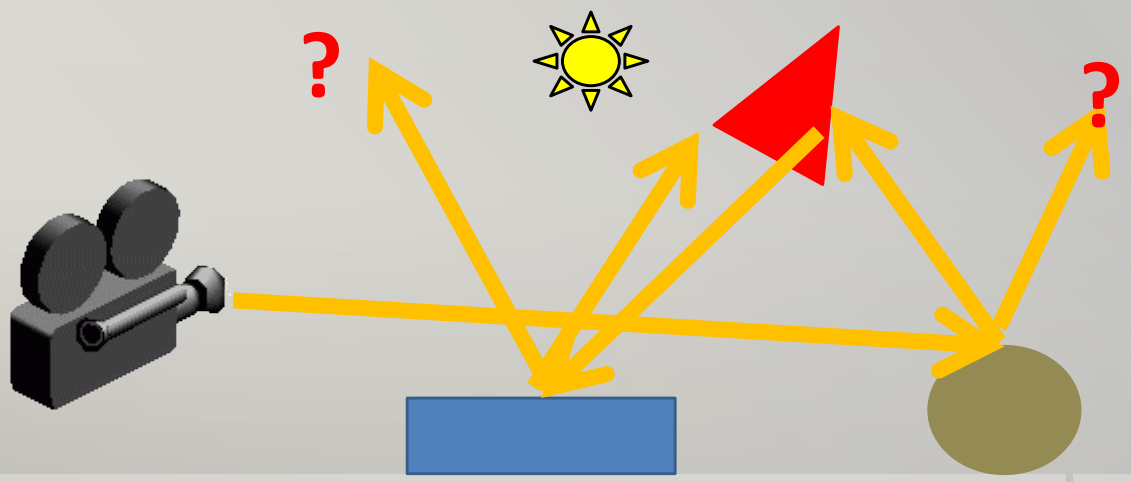
Multiple Reflections

When considering multiple reflections in ray-tracing, we will ask :

- When should we end?
- What are the amounts of light being absorbed and being reflected?

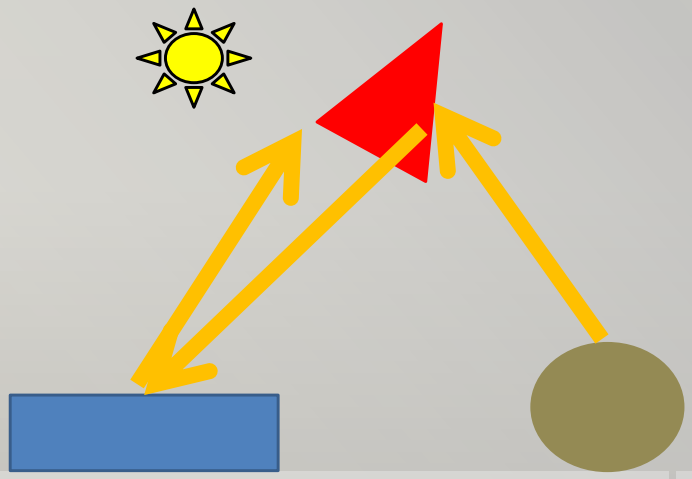
Multiple Reflections

- It can be a problem on when to terminate the reflection
- Also, for each reflection, many reflection rays can be generated (consider case of diffusion)
- Obviously, some rays can be ignored, that is those point at nothing/vacuum



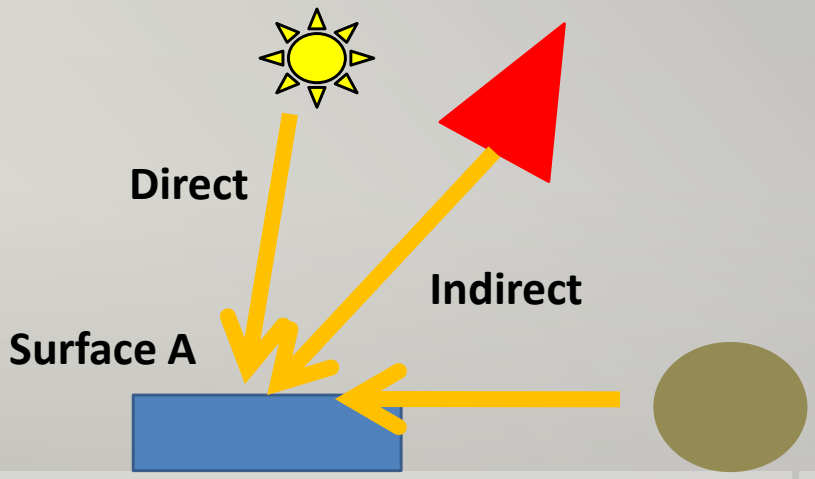
Multiple Reflections

- Actually, the problem of multiple reflections can be simplified or viewed in another aspect
- That is the light interaction among surfaces
- Our goal is to know the ultimate interaction result between each pair of surfaces

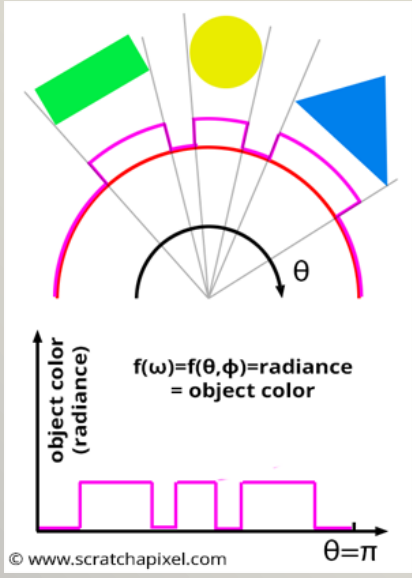


Multiple Reflections

- Consider we are trying to render a particular surface A, we want to know
  - Direct illumination (from light sources)
  - Indirect illumination (from other surfaces)
- When render the whole scene, we repeat the process for all surfaces



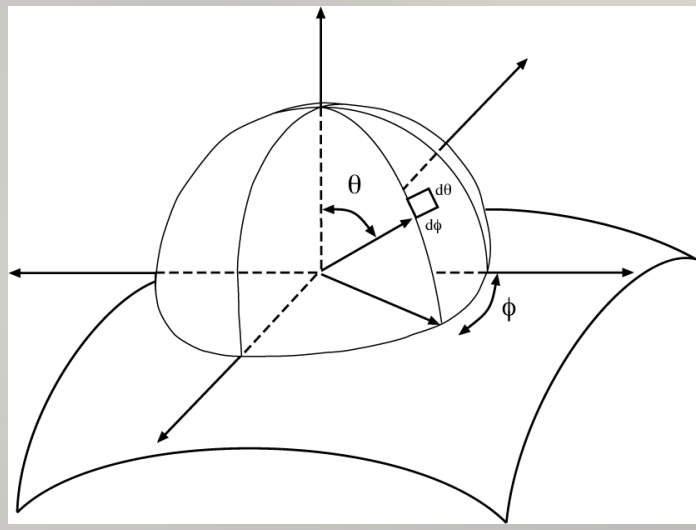
Radiosity (Thermal Heat Transfer)



- The indirect illumination involve the study of **radiometry** from physics
- The study of light distribution:
  - how "bright" will surfaces be?
  - what is "brightness"?

    measuring light

    interactions between light and surfaces
- Consider around any point is a hemisphere of directions



Radiosity (Thermal Heat Transfer)

- The "radiosity" method has its basis in the field of thermal heat transfer.

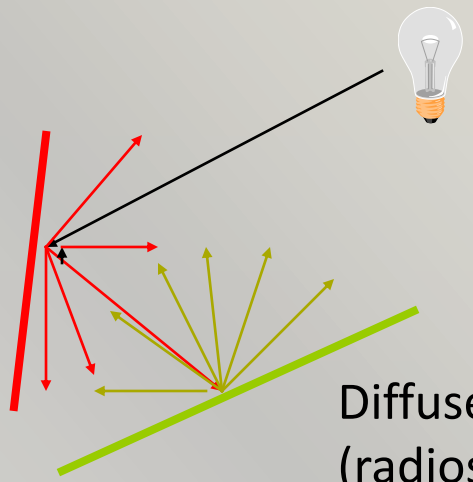
- Heat transfer theory describes radiation as the transfer of energy from a surface when that surface has been thermally excited.

- This encompasses both surfaces which are basic emitters of energy, as with light sources, and surfaces which receive energy from other surfaces and thus have energy to transfer.

- This "thermal radiation" theory can be used to describe the transfer of many kinds of energy between surfaces, including light energy.

Radiosity (Computer Graphics)

- Factors will consider in indirect illumination (i.e. diffuse-diffuse reflection / radiosity)
  - Visibility
  - Relative orientation
  - Distance
- Finally, we estimate their equilibrium state of light amount(a balance in input and output)



Diffuse-diffuse (radiosity)

Radiosity (Computer Graphics)

- Assumption #1: surfaces are diffuse emitters and reflectors of energy, emitting and reflecting energy uniformly over their entire area.

- Assumption #2: an equilibrium solution can be reached; that all of the energy in an environment is accounted for, through absorption and reflection.

- Also viewpoint independent: the solution will be the same regardless of the viewpoint of the image.

The Radiosity Equation

- The "radiosity equation" describes the **amount of energy** which can be emitted from a surface, as the sum of the energy inherent in the surface (a light source, for example) and the energy which strikes the surface, being emitted from some other surface.

- The energy which leaves a surface (surface "j") and strikes another surface (surface "i") is attenuated by two factors:
  - the **"form factor"** between surfaces "i" and "j", which accounts for the physical relationship between the two surfaces
  - the **reflectivity of surface "i"**, which will absorb a certain percentage of light energy which strikes the surface.

The Radiosity Equation

$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

Radiosity of surface i

Emissivity of surface i

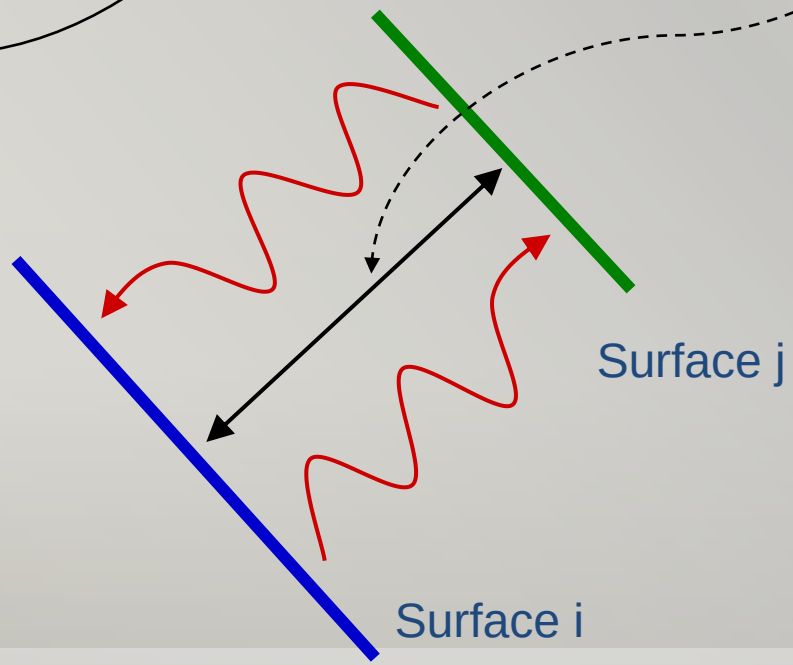
Reflectivity of surface i

will <u>absorb</u> a certain percentage of light energy which strikes the surface

Radiosity of surface j

Form Factor of surface j relative to surface i

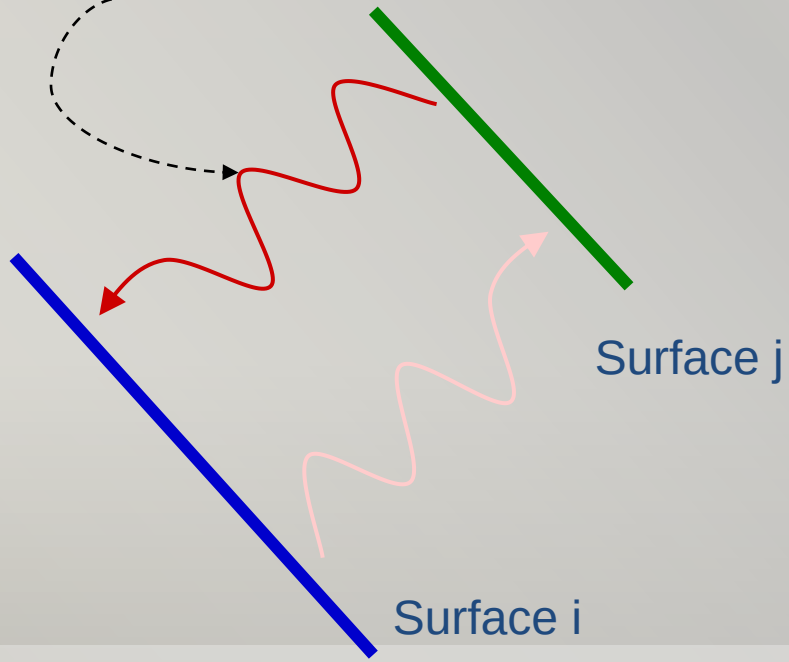
accounts for the <u>physical relationship</u> between the two surfaces



The Radiosity Equation

$$B_i = E_i + \rho_i \boxed{\sum B_j F_{ij}}$$

Energy reaching surface i from other surfaces



The Radiosity Equation

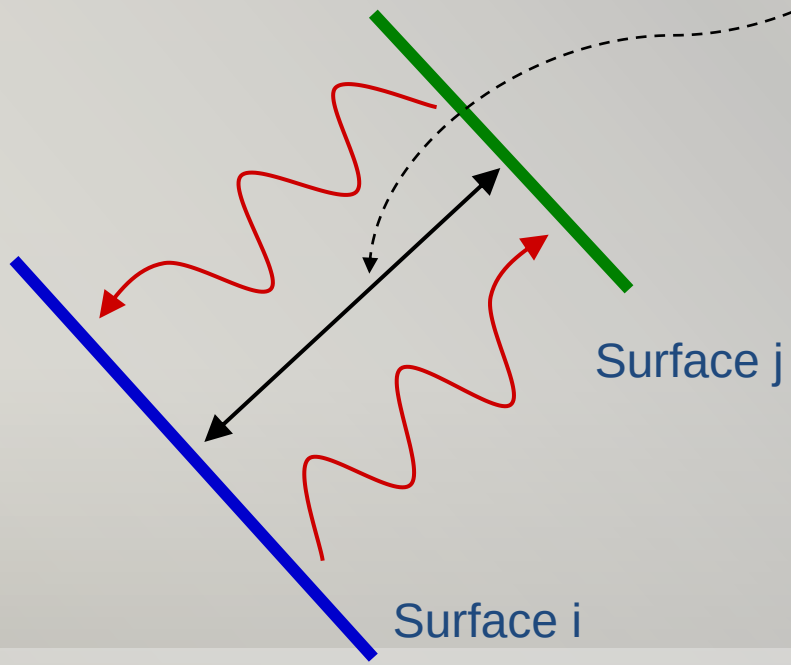
$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

Energy reaching surface i from other surfaces

Radiosity of surface j

Form Factor of surface j relative to surface i

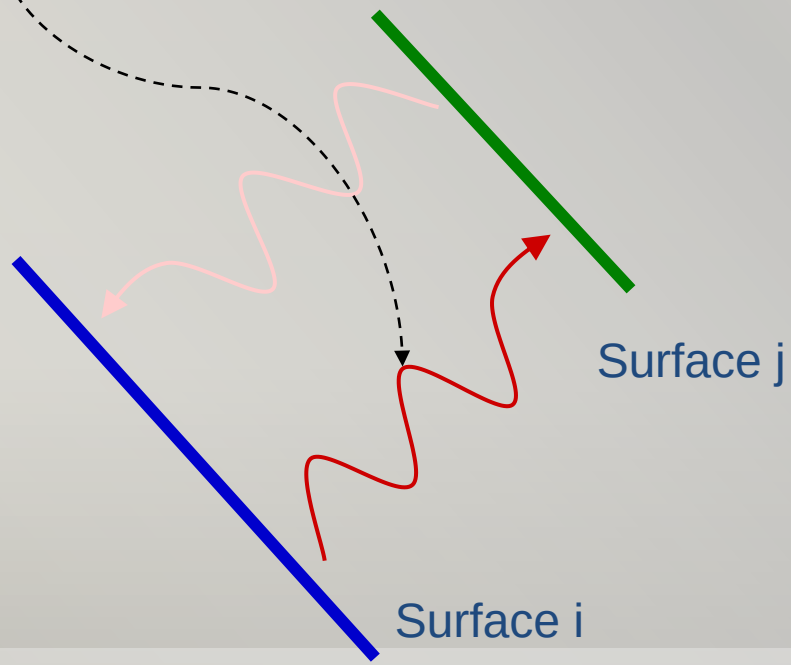
accounts for the physical relationship between the two surfaces



The Radiosity Equation

$$B_i = \boxed{E_i} + \rho_i \sum B_j F_{ij}$$

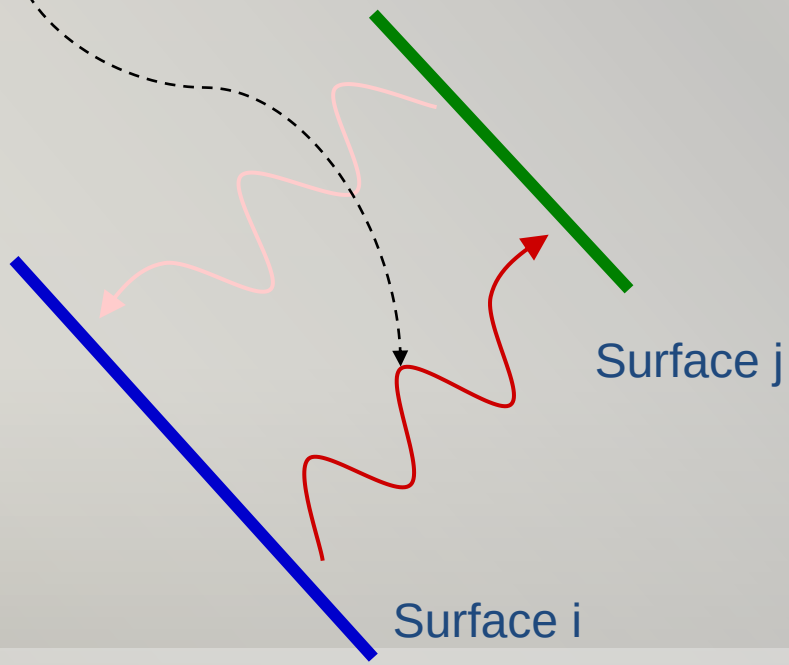
Energy emitted by surface i



The Radiosity Equation

$$B_i = E_i + \boxed{\rho_i \sum B_j F_{ij}}$$

Energy reflected by surface i



The Radiosity Equation

Energy reflected by surface i

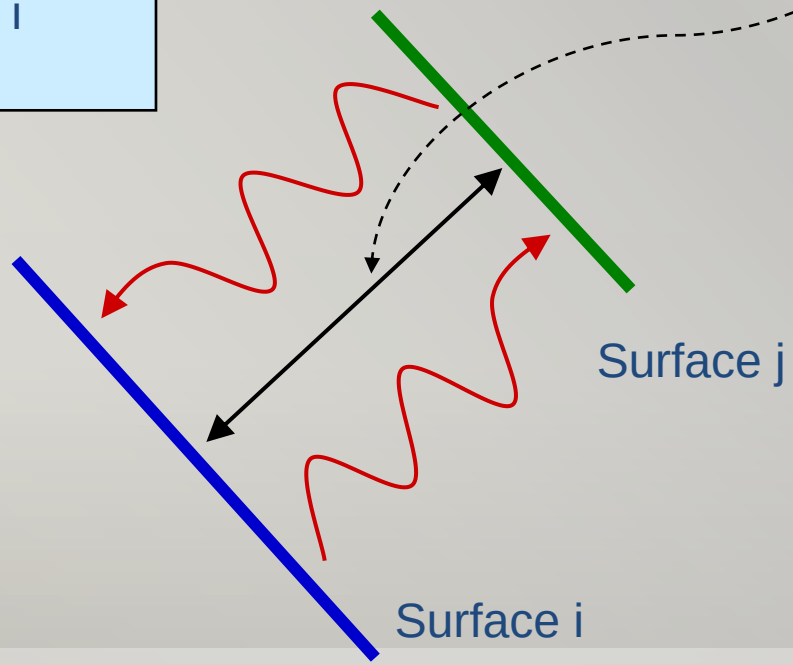
$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

Reflectivity of surface i

Radiosity of surface j

Form Factor of surface j relative to surface i

Energy reflected by surface i = **Reflectivity** of surface i * **Energy reaching** surface i from other surfaces



Form Factor accounts for the physical relationship between the two surfaces

Reflectivity will absorb a certain percentage of light energy which strikes the surface

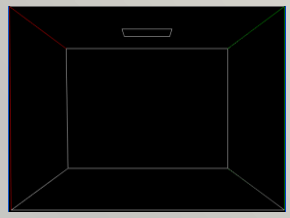
Radiosity

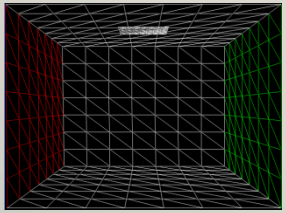
- **Classic radiosity = finite element method**
  - **Each surface patch is an element**
- **Assumptions**
  - **Diffuse reflectance only**
  - **Usually polygonal surfaces**

- **Advantages**
  - **Soft shadows and indirect lighting**
  - **View independent solution**
  - **Precompute for a set of light sources**

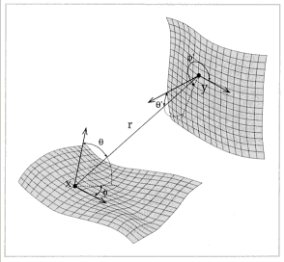
Classic Radiosity Algorithm

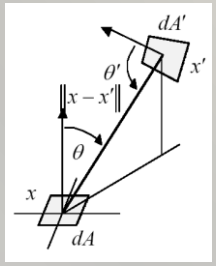
Mesh Surfaces into Elements



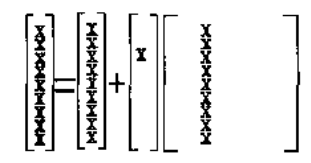


Compute Form Factors Between Elements





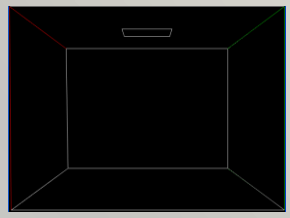
Solve Linear System for Radiosities

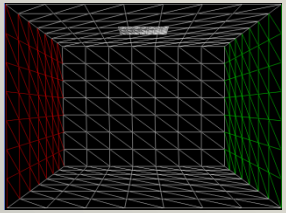


Reconstruct and Display Solution

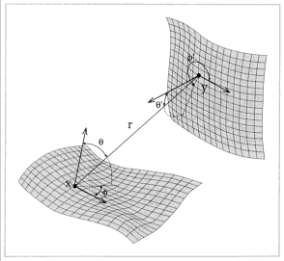
Classic Radiosity Algorithm

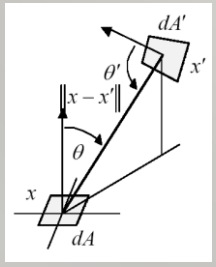
Mesh Surfaces into Elements



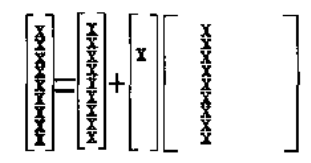


Compute Form Factors Between Elements





Solve Linear System for Radiosities

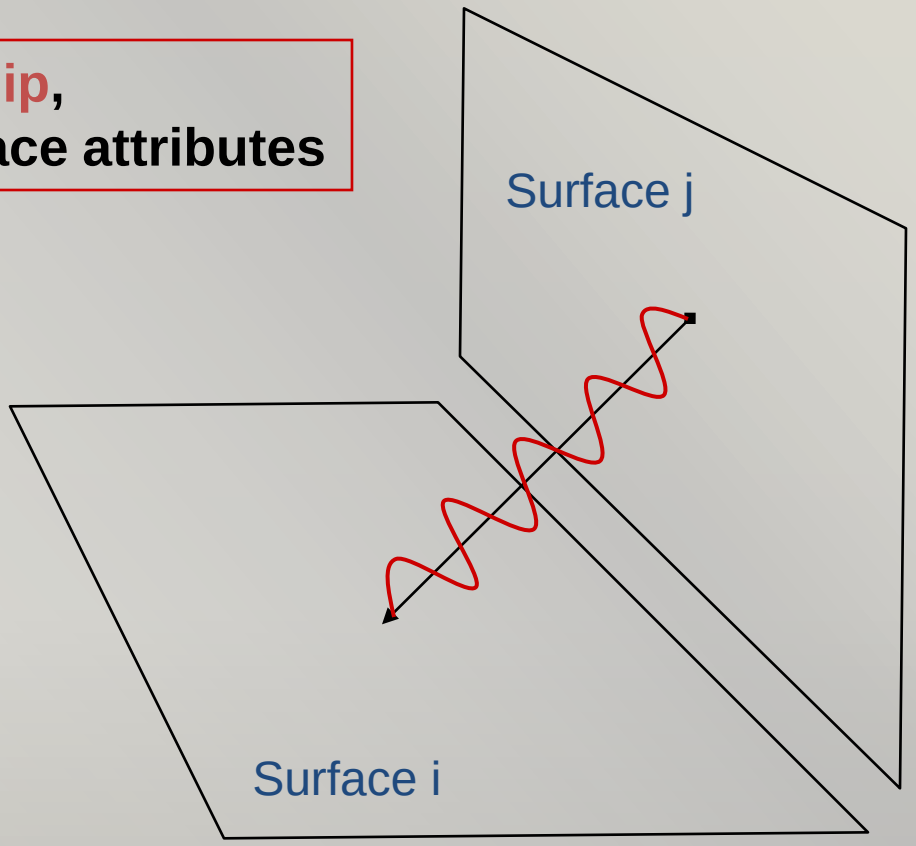


Reconstruct and Display Solution

The Form Factor:

the fraction of energy leaving one surface that reaches another surface

It is a purely geometric relationship, independent of viewpoint or surface attributes



Between <u>differential areas</u>, the form factor equals:

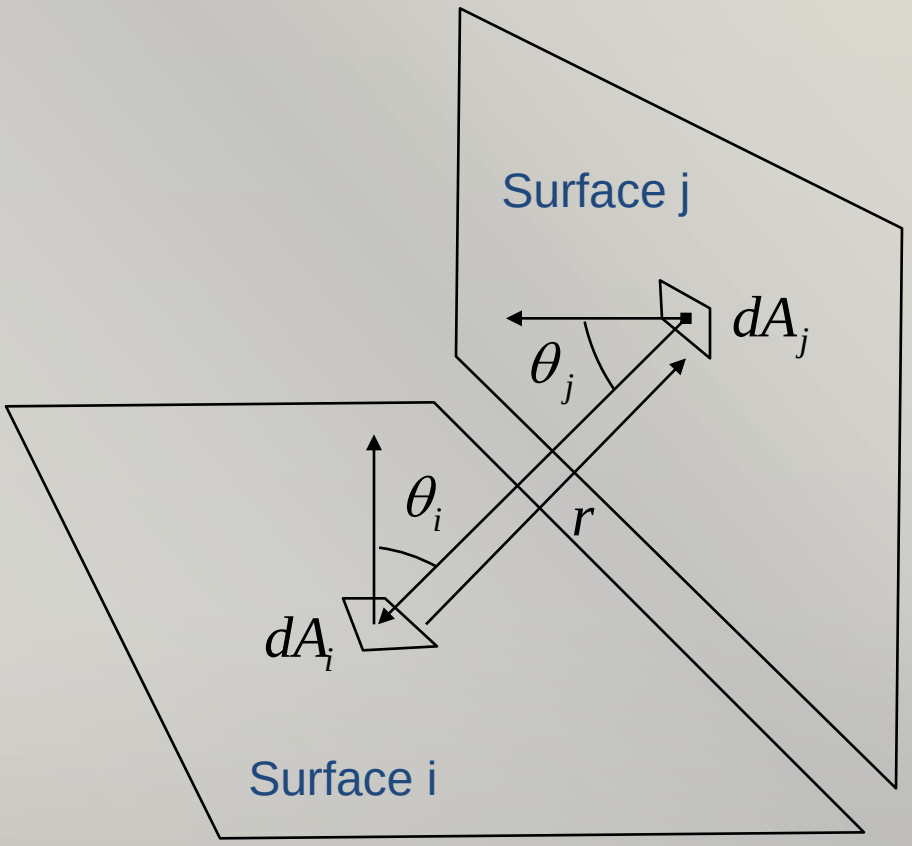
differential area of surface I, j

angle between Normal$_i$ and r

angle between Normal$_j$ and r

$$F\,dA_j\,dA_j = \frac{\cos\theta_i \cos\theta_j}{\pi |r|^2}$$

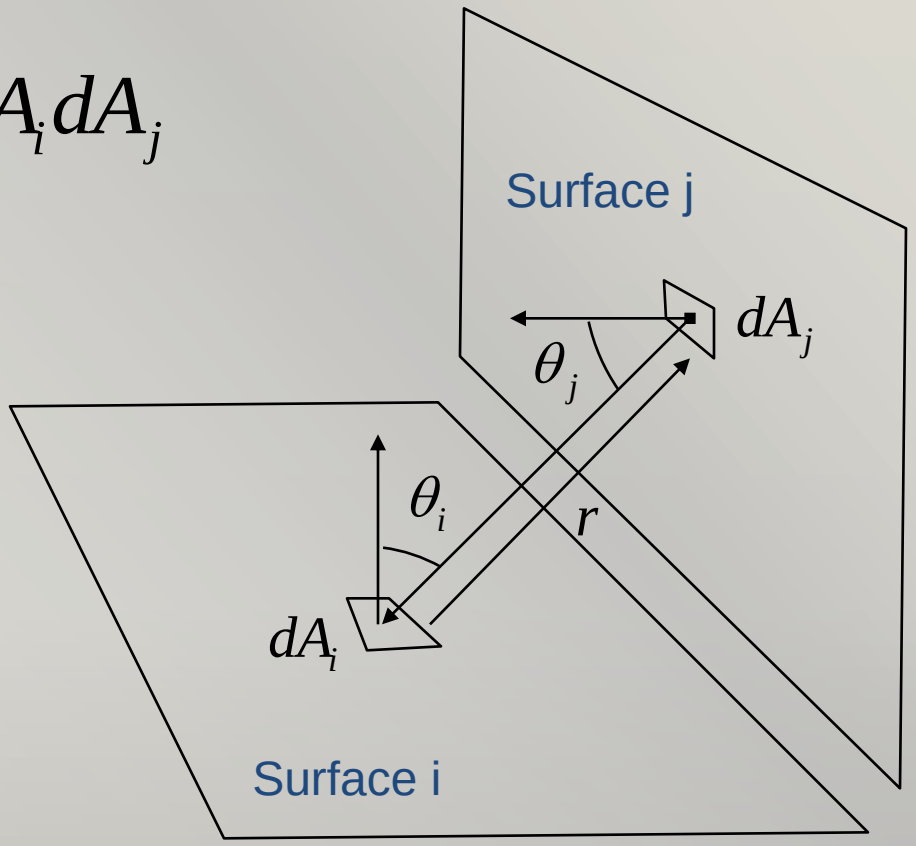
vector from dA$_i$ to dA$_j$



Between <u>differential areas</u>, the form factor equals: $$F dA_j dA_j = \frac{\cos\theta_i \cos\theta_j}{\pi |r|^2}$$

The overall form factor between i and j is found by integrating

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi |r|^2} dA_i dA_j$$



Next Step:
Learn ways of computing **form factors**

- Recall the Radiosity Equation:

$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

- The F_{ij} are the form factors

- Form factors independent of radiosities (depend only on scene geometry)

Form Factors in (More) Detail

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi |r|^2} dA_i dA_j$$

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi |r|^2} V_{ij} dA_i dA_j$$

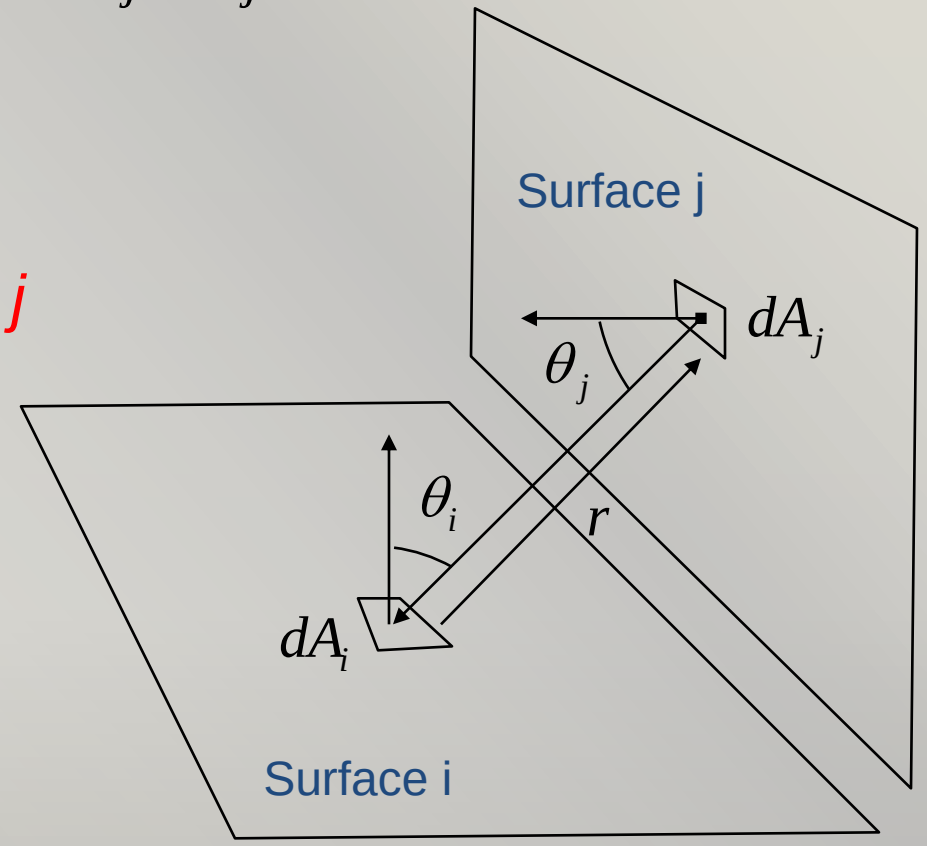
where V_{ij} is the visibility (0 or 1)

We have <u>two integrals</u> to compute:

$$F_{ij} = \frac{1}{A_i} \boxed{\int_{A_i}} \boxed{\int_{A_j}} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} V_{ij}\, dA_j\, dA_i$$

Area integral over surface *i*

Area integral over surface *j*



Area Sampling

1. Subdivide A_j into small pieces dA_j
2. For all dA_j

   cast ray *dAj-dAj* to determine V_{ij}

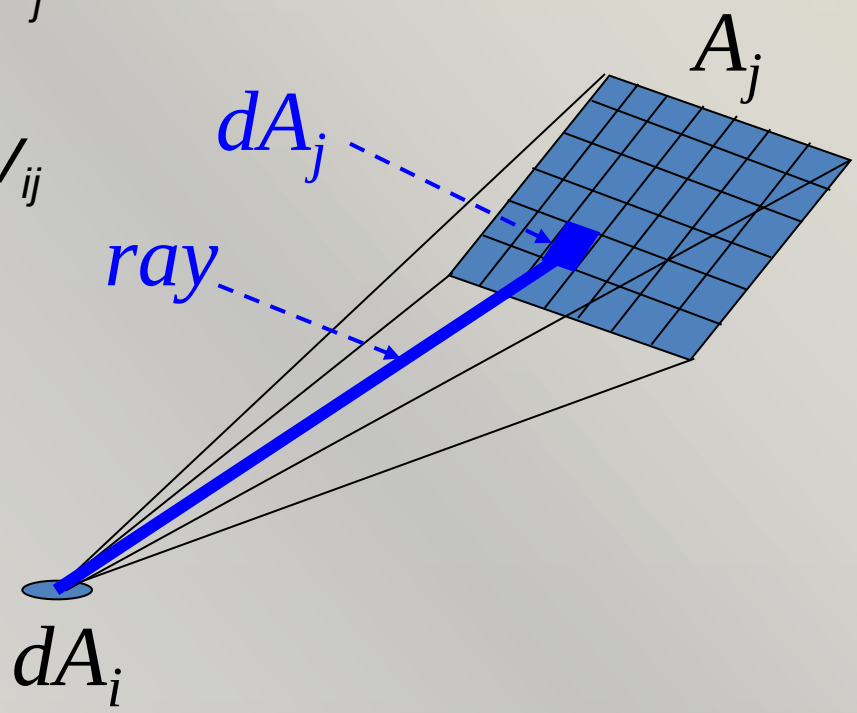
   if visible

   compute F_{dAidAj}

$$F_{dA_i dA_j} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} V_{ij} dA_j$$

   sum up

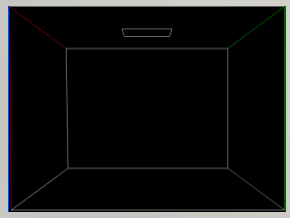
   F_{dAiAj} += F_{dAidAj}

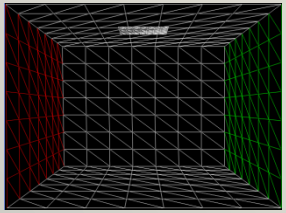


3. We have now F_{dAiAj}

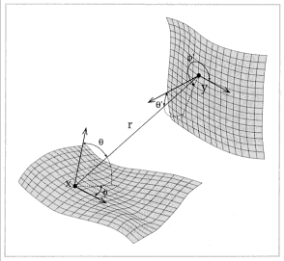
Classic Radiosity Algorithm

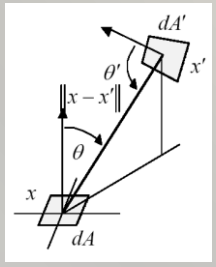
Mesh Surfaces into Elements



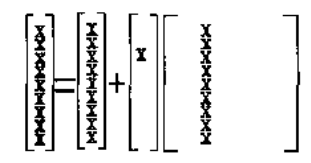


Compute Form Factors Between Elements





Solve Linear System for Radiosities



Reconstruct and Display Solution

Recall →The Radiosity Equation

$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

Radiosity of surface i

Emissivity of surface i

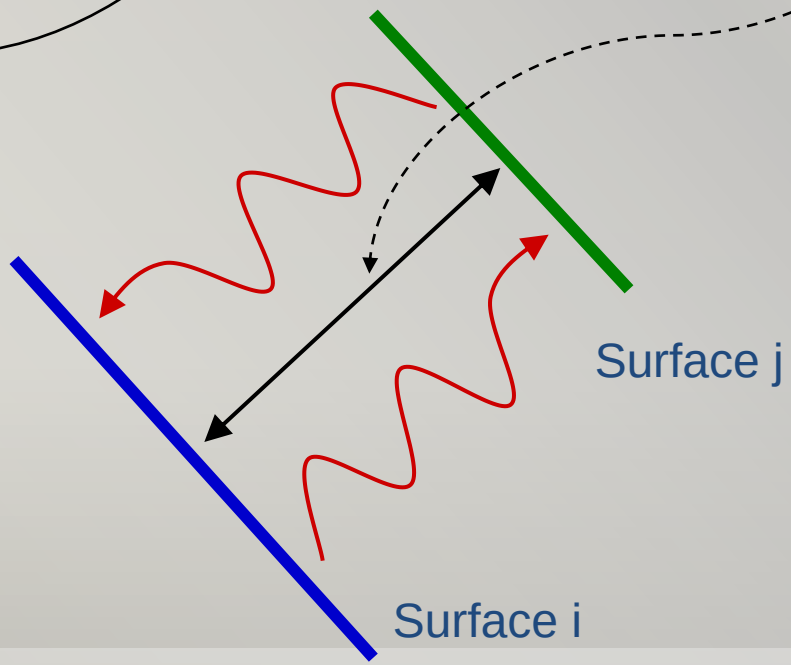
Reflectivity of surface i

will absorb a certain percentage of light energy which strikes the surface

Radiosity of surface j

Form Factor of surface j relative to surface i

accounts for the physical relationship between the two surfaces



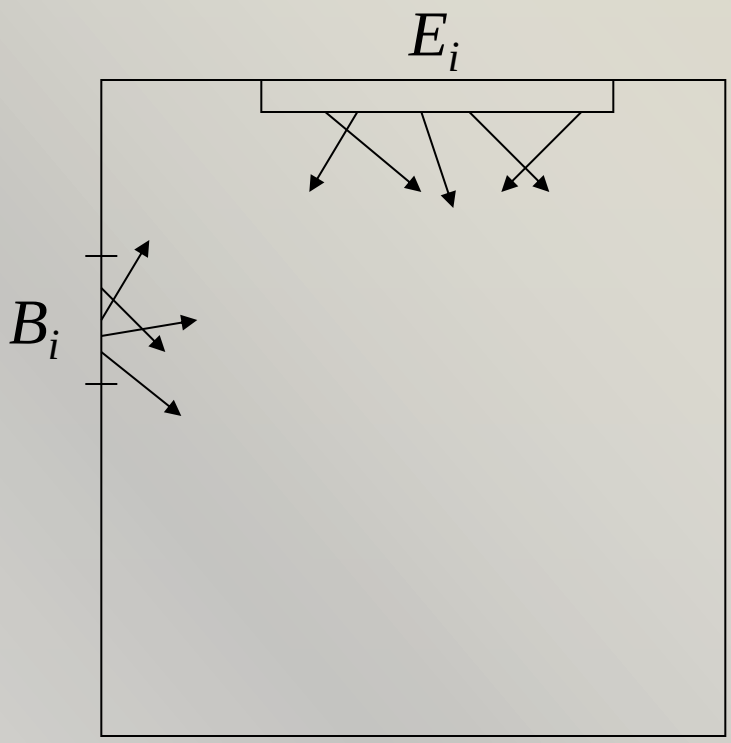
Radiosity Matrix

$$B_i A_i = E_i A_i + \rho_i \sum_{j=1}^{n} F_{ji} B_j A_j$$

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

$$\boxed{B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j}$$

$$B_i - \rho_i \sum_{j=1}^{n} F_{ij} B_j = E_i$$



$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

Radiosity Matrix

- The "full matrix" radiosity solution calculates the form factors between each pair of surfaces in the environment, then forms a series of simultaneous linear equations.

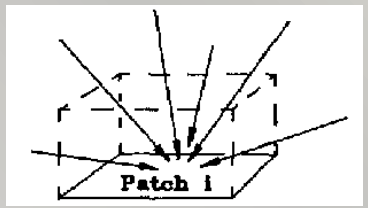
$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

- This matrix equation is solved for the "B" values, which can be used as the final intensity (or color) value of each surface.

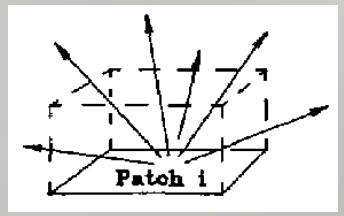
Radiosity Matrix

- This method produces a complete solution, at the substantial cost of
  - first calculating form factors between each pair of surfaces
  - and then the solution of the matrix equation.
- Solve [F][B] = [E]
- **Direct methods: *$O(n^3)$***
  - **Gaussian elimination**

    Goral, Torrance, Greenberg, Battaile, 1984



- **Iterative methods: *$O(n^2)$***

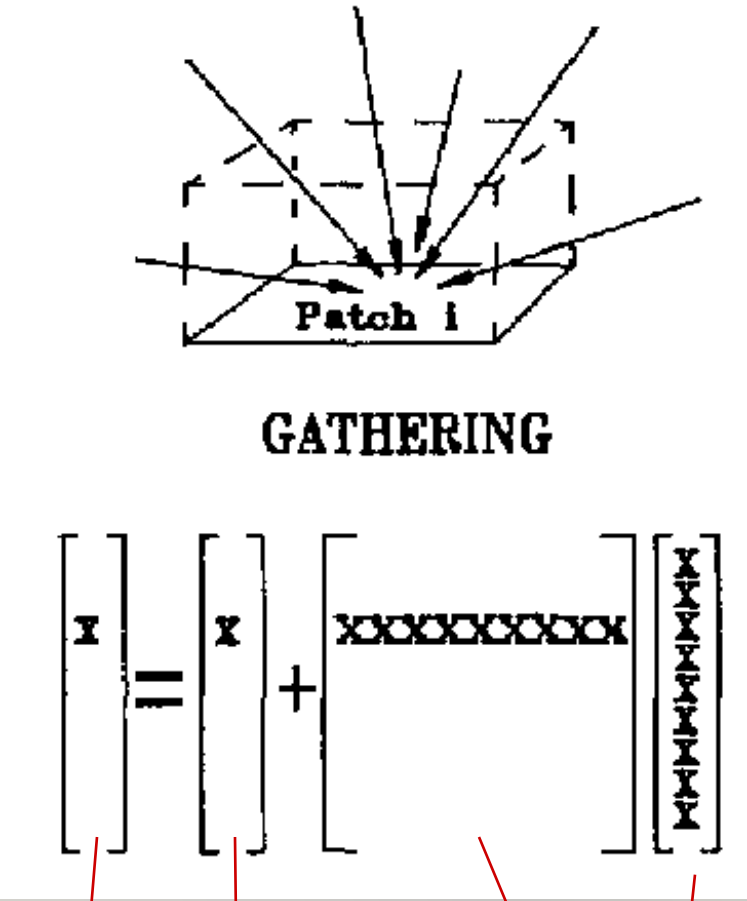


Energy conservation

→¨diagonally dominant ¨ → iteration converges

Gathering

- In a sense, the light leaving patch i is determined by *gathering* in the light from the rest of the environment



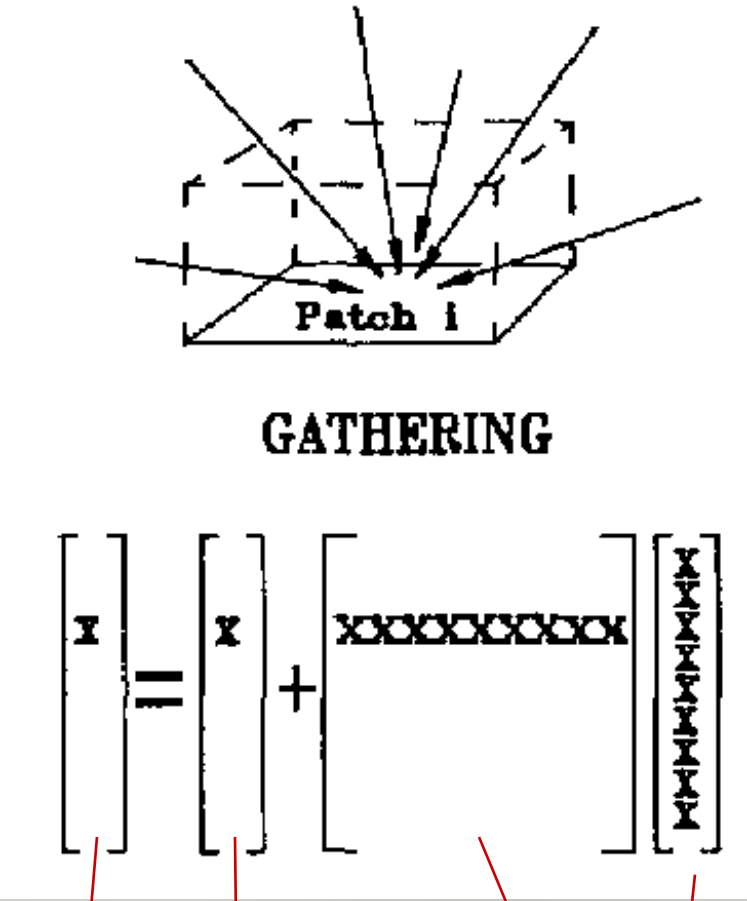
$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij}$$

$$B_i = E_i + \sum_{j=1}^{n} \left(\rho_i F_{ij}\right) B_j$$

$$B_i \text{ due to } B_j = \rho_i B_j F_{ij}$$

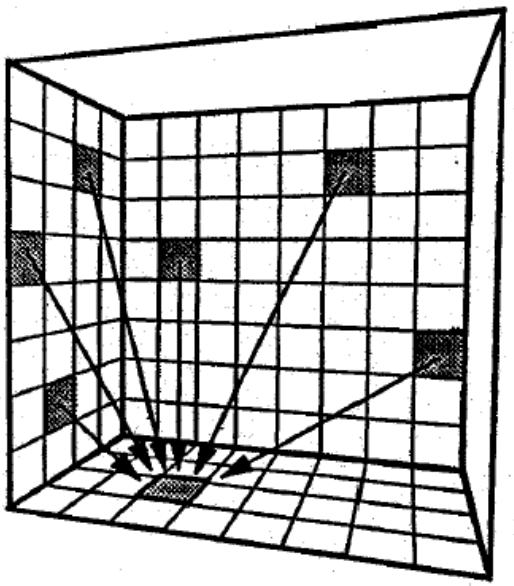
Gathering

- Gathering light through a hemi-cube allows one patch radiosity to be updated.



$$B_i = E_i + \sum_{j=1}^{n} \left(\rho_i F_{ij}\right) B_j$$

Gathering



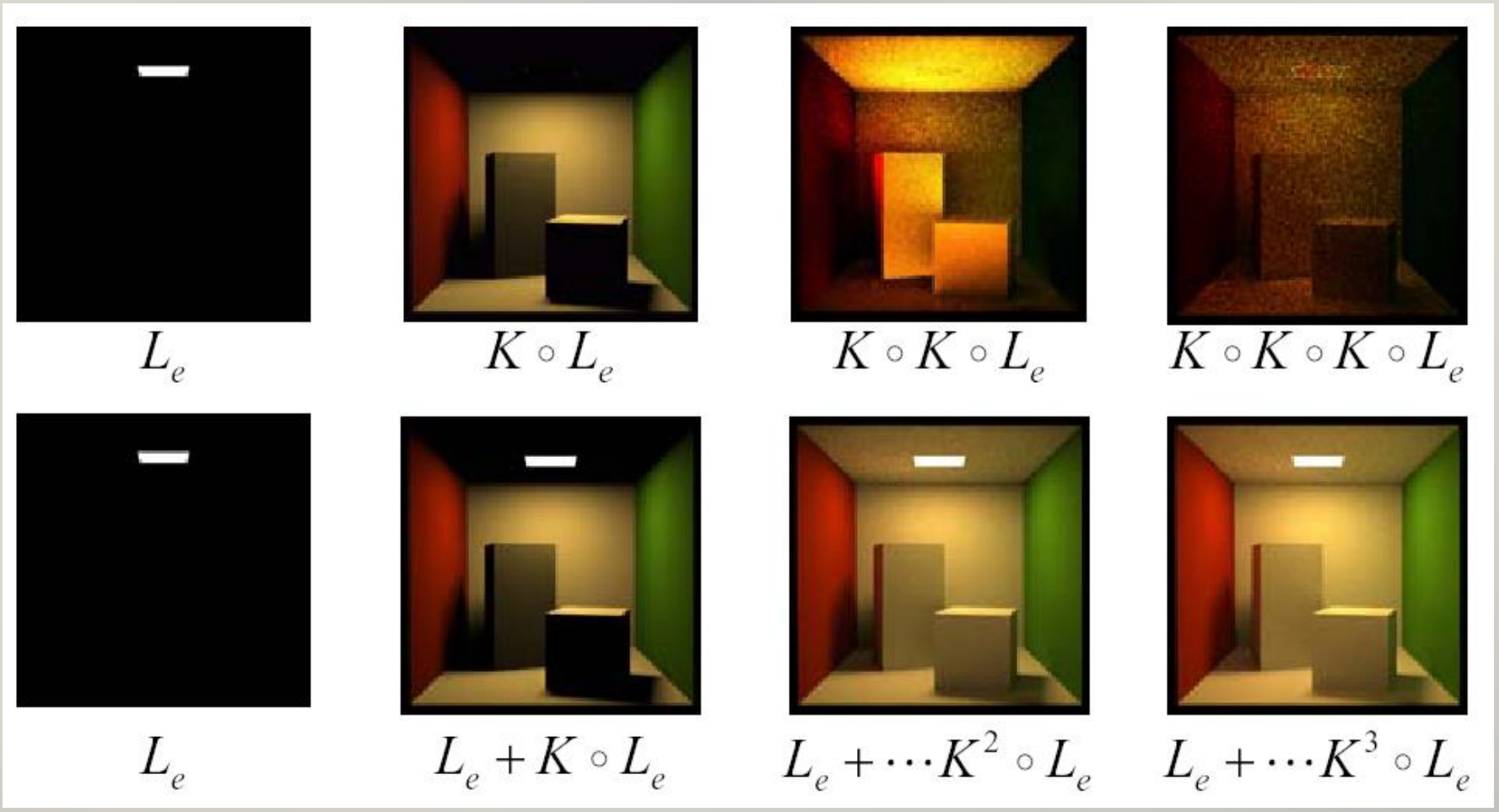
```
for(i=0; i<n; i++)
  B[i] = Be[i];

while( !converged ) {
  for(i=0; i<n; i++) {
    E[i] = 0;
    for(j=0; j<n; j++)
      E[i] += F[i][j]*B[j];
    B[i] = Be[i]+rho[i]*E[i];
  }
}
```

Row of F times B

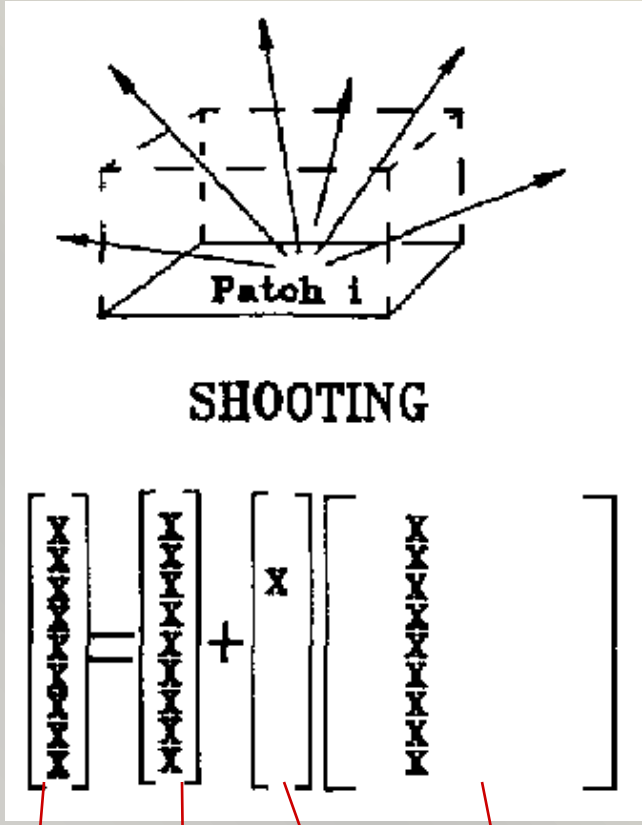
Calculate one row of F and discard

Successive Approximation



Shooting

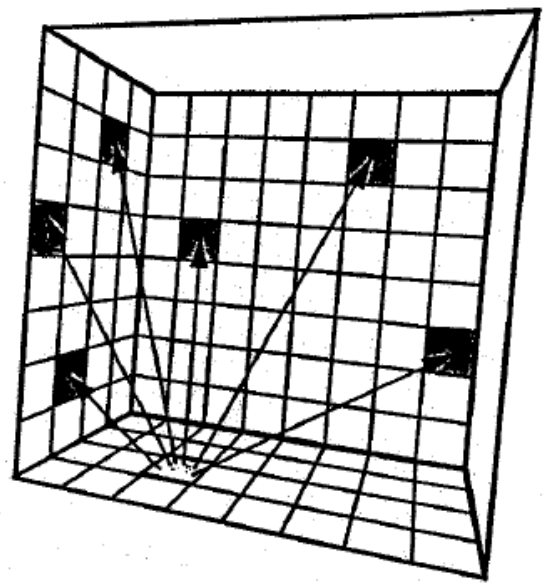
- Shooting light through a single hemi-cube allows the whole environment's radiosity values to be updated simultaneously.



For all j $\Rightarrow$ $B_j = B_j + B_i\left(\rho_j E_{ji}\right)$

where $F_{ji} = \dfrac{F_{ij} A_i}{A_j}$

Shooting

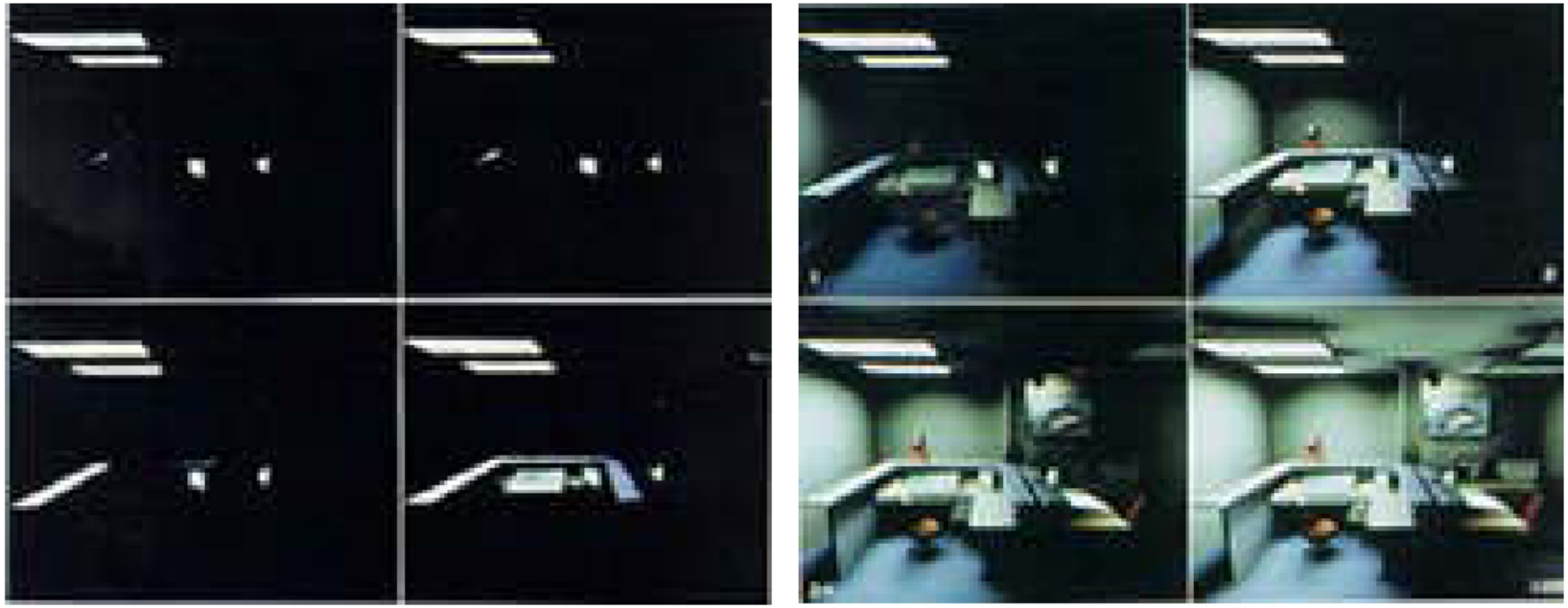


```
for(i=0; i<n; i++) {
  B[i] = dB[i] = Be[i];
  while( !converged ) {
    set i st dB[i] is the largest;
    for(j=0;j<n;j++)
      if(i!=j) {
        db =rho[j]*F[j][i]*dB[i];
        dB[j] += db;
        B[j] += db;
      }
    dB[i]=0;
  }
}
```

Brightness order

Column of F times B

Progressive Radiosity

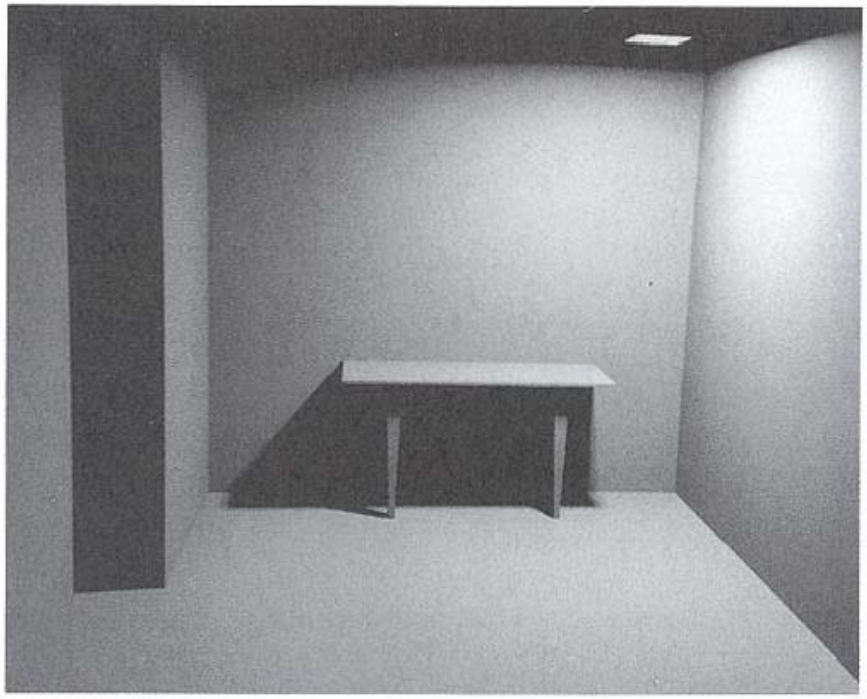


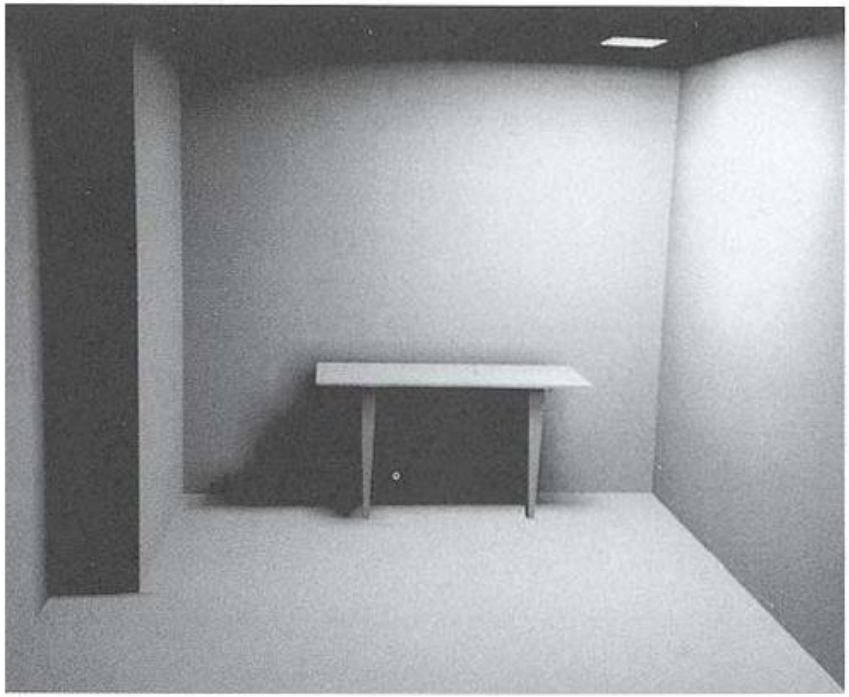
(a) (b)

(a) Traditional Gauss-Seidel iteration of 1, 2, 24 and 100.
(b) Progressive Refinement (PR) iteration of 1, 2, 24 and 100.

From Cohen, Chen, Wallace, Greenberg 1988

Accuracy

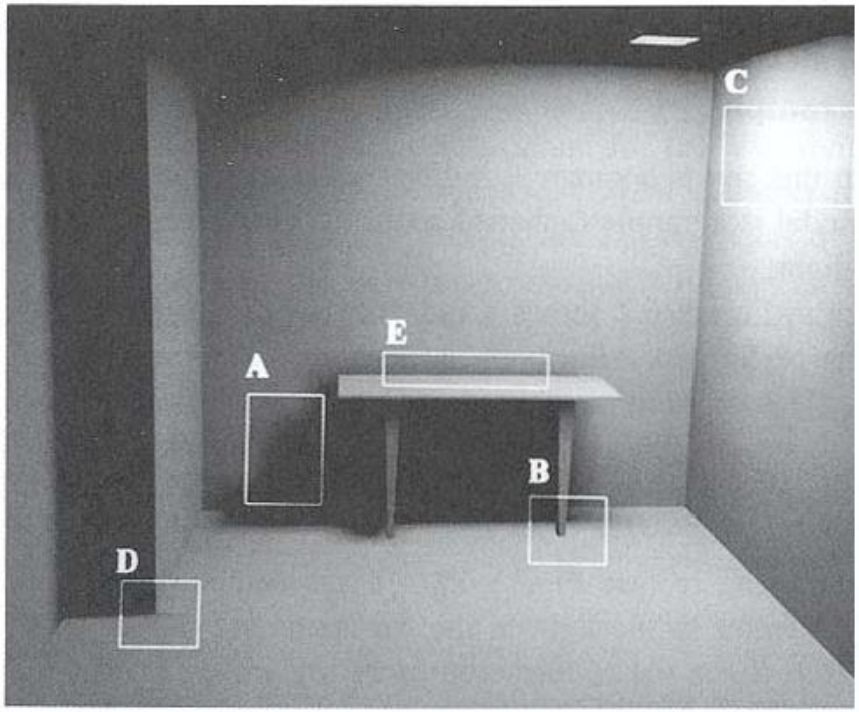


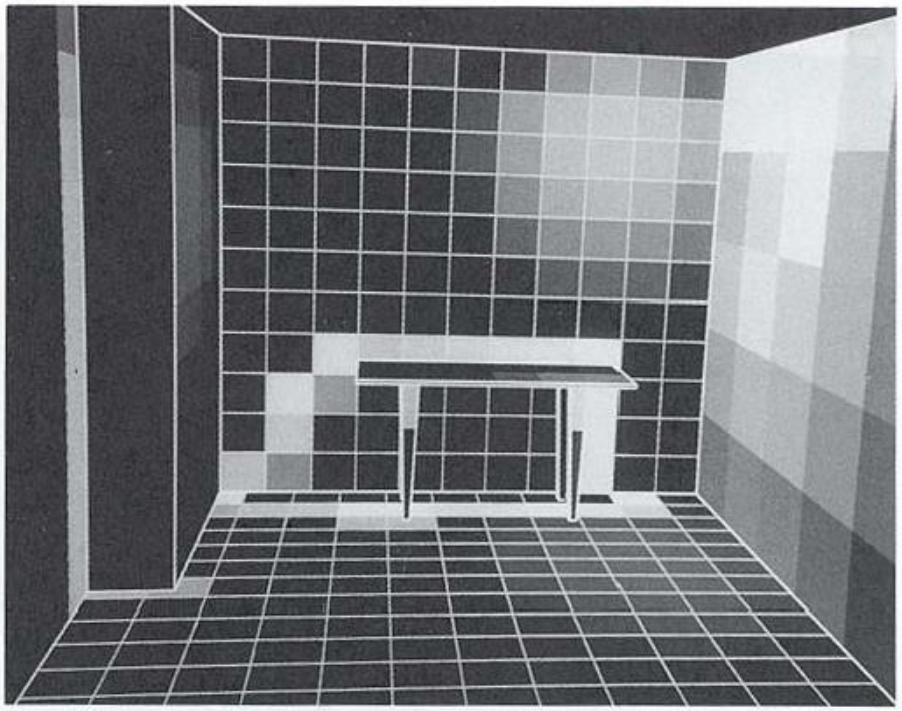


Reference Solution **Uniform Mesh**

Table in room sequence from Cohen and Wallace

Artifacts

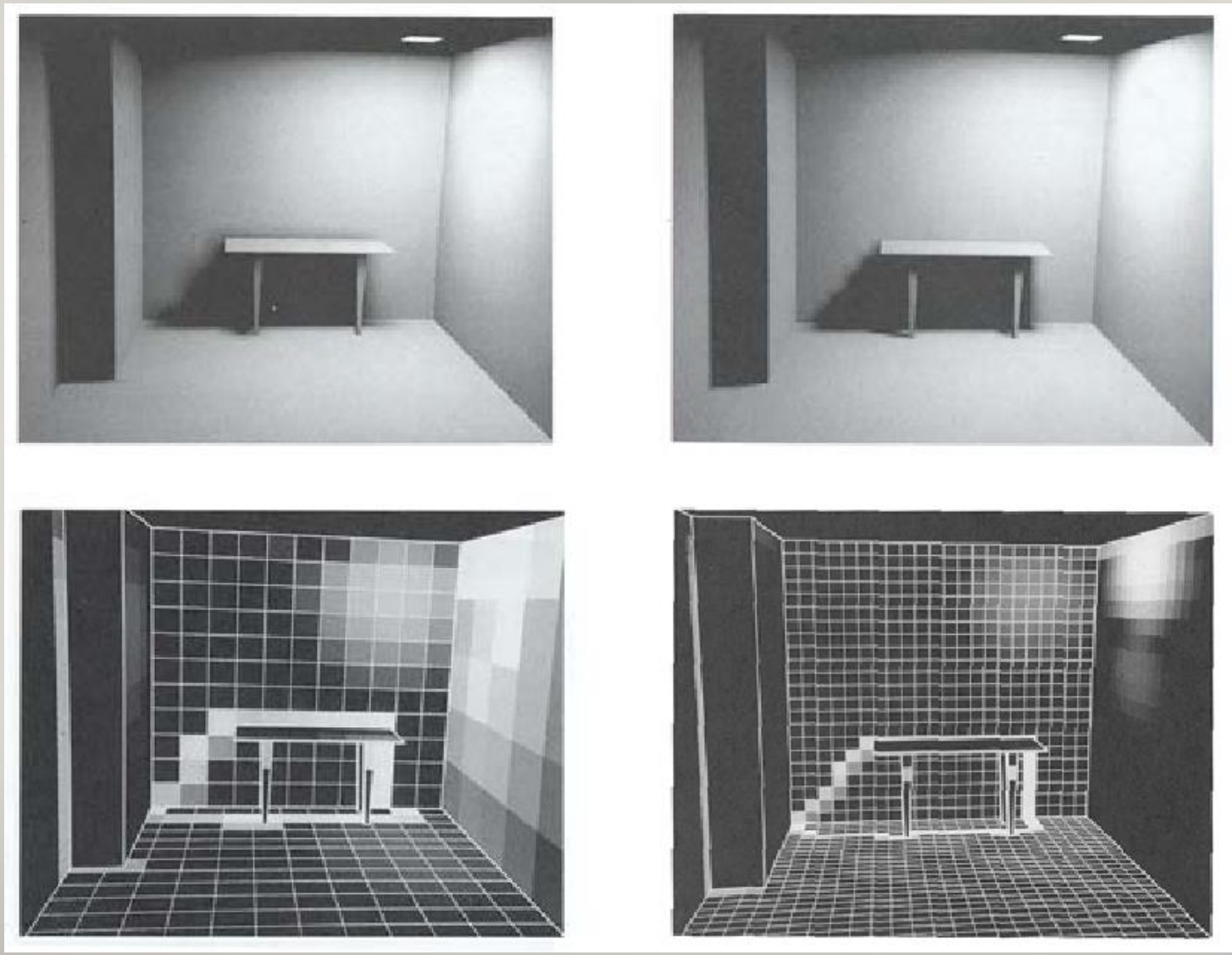




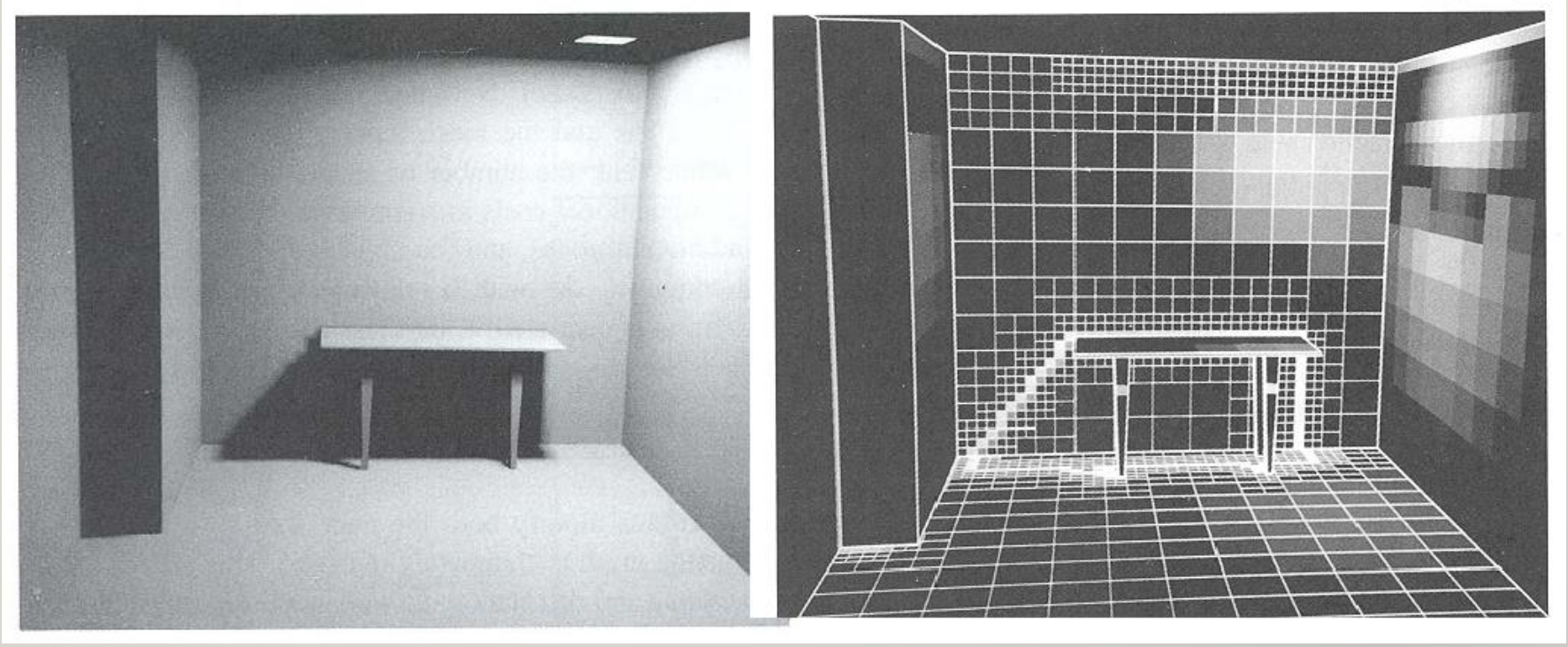
Error Image

A. Blocky shadows
B. Missing features
C. Mach bands
D. Inappropriate shading discontinuities
E. Unresolved discontinuities

Increasing Resolution

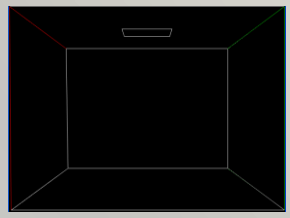


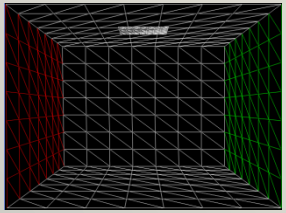
Adaptive Meshing



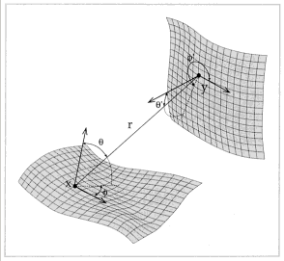
Classic Radiosity Algorithm

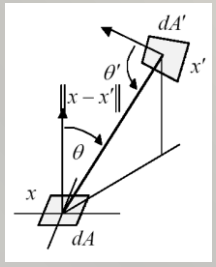
Mesh Surfaces into Elements



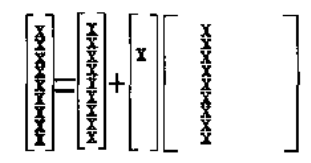


Compute Form Factors Between Elements





Solve Linear System for Radiosities

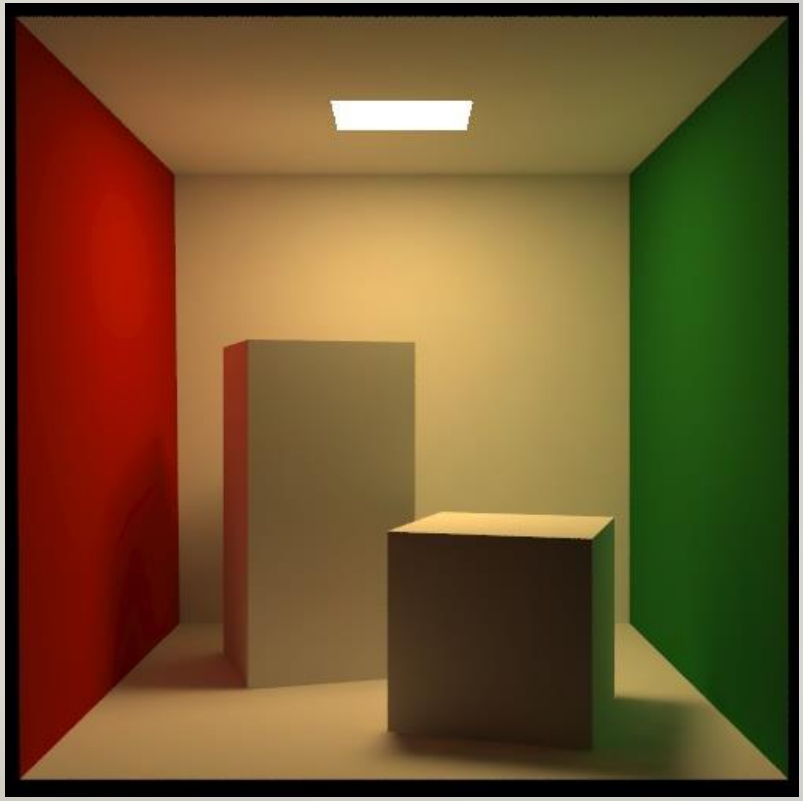


Reconstruct and Display Solution

Steps in Radiosity(a summary)

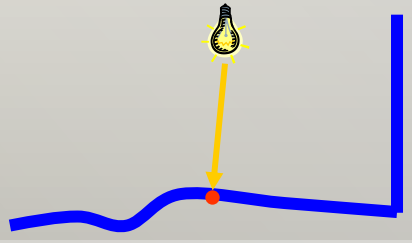
- We have the form factors
- How do we find the radiosity solution for the scene?
  - The "Full Matrix" Radiosity Algorithm
  - Gathering & Shooting
  - Progressive Radiosity
- Meshing

Some Radiosity Results

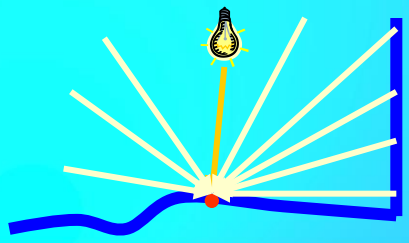


Radiosity

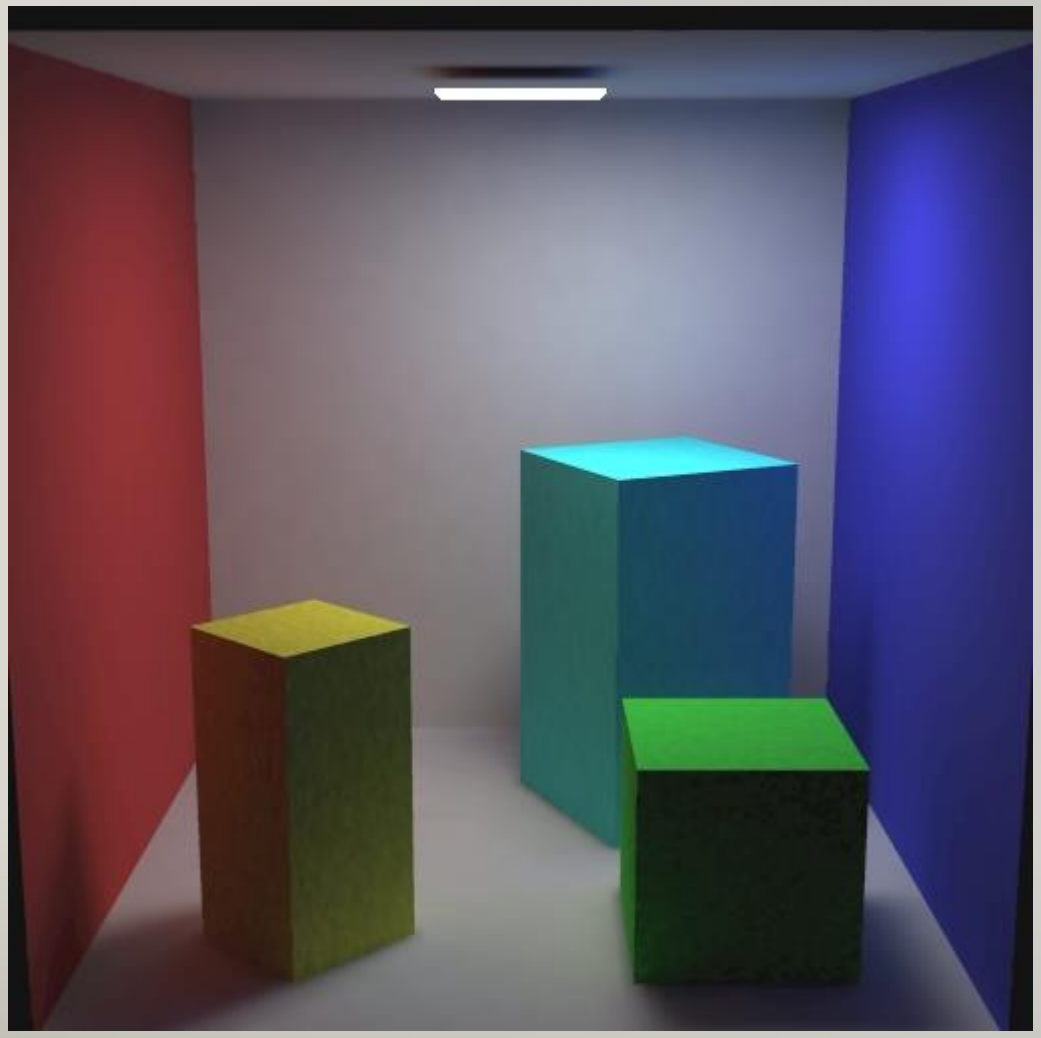
Direct-only

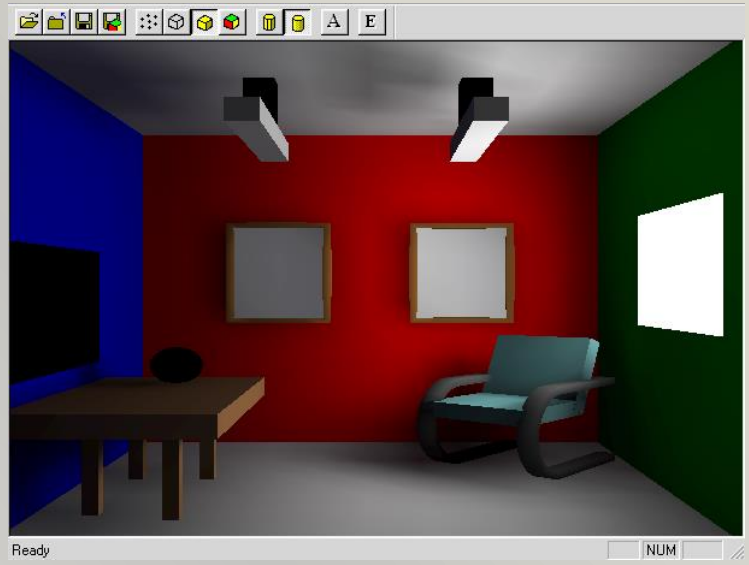


global =
direct +
indirect



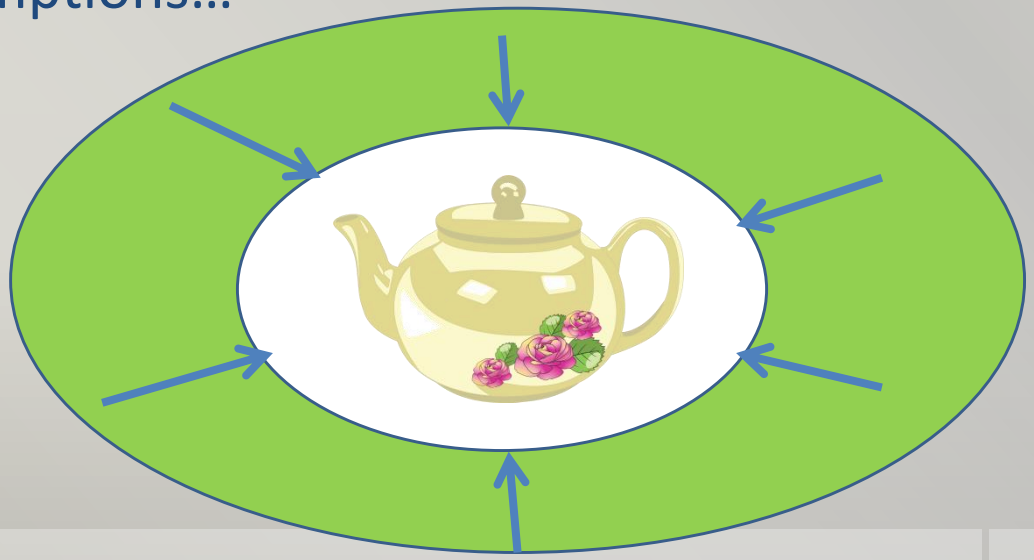
Radiosity





Precomputed Radiance Transfer (PRT)

- Precomputation of radiosity save time, but still complicated for a large scene
  - Remember it is a surface-to-surface relationship
- To make the idea real-time, simplification is needed
- How about just consider one object and its surrounding?
- Several assumptions…



Assumptions

- Static geometry (i.e. no deformation)
  - To allow precomputation
- Lighting is assumed to be distant relative to the object
- Only consider irradiance (or exit radiance) from the object
  - No need to consider effect to the outside environment
- Sometimes it is called “relighting”

PRT Terminology

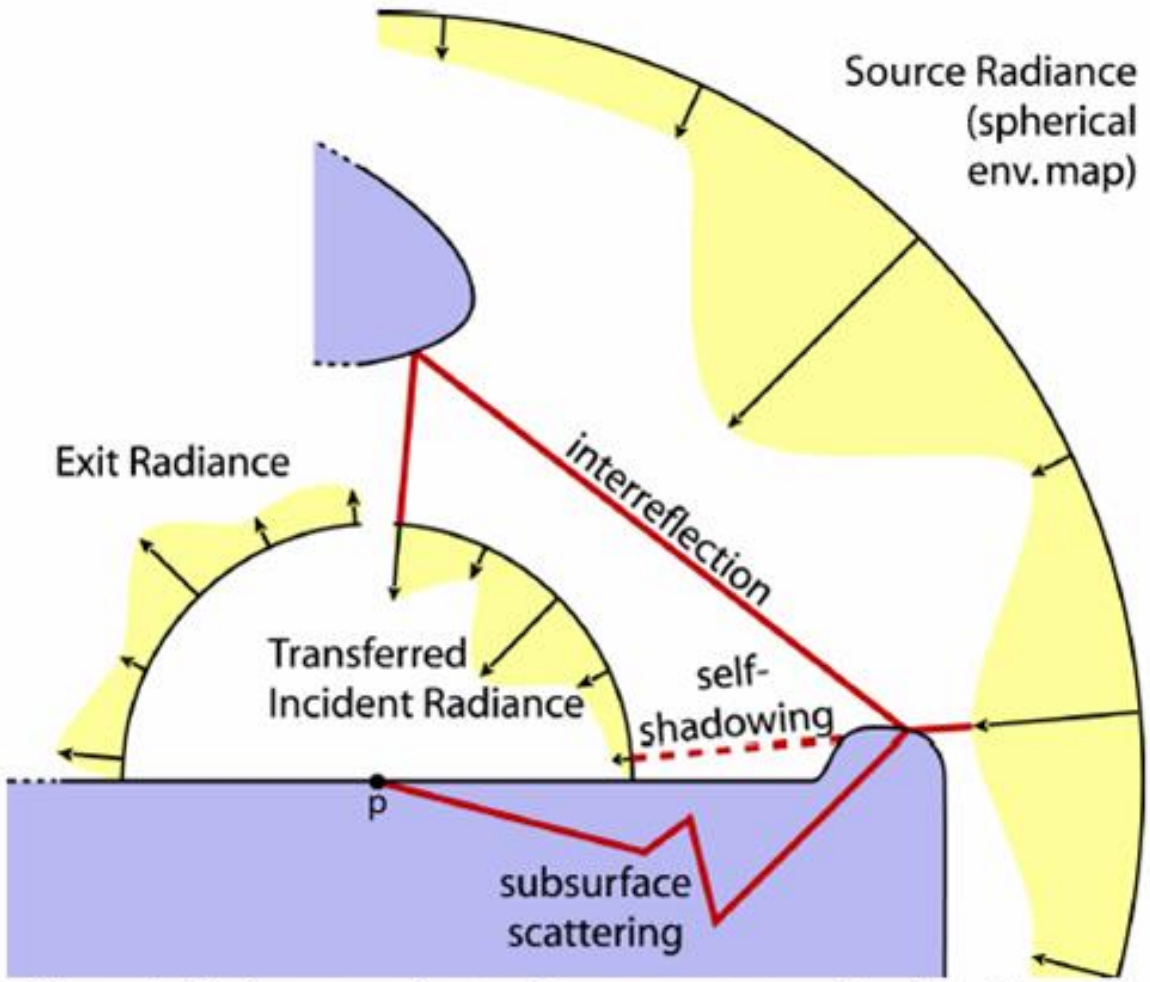
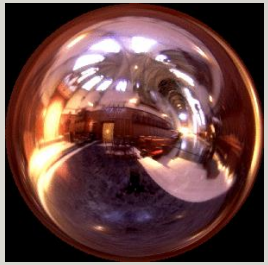
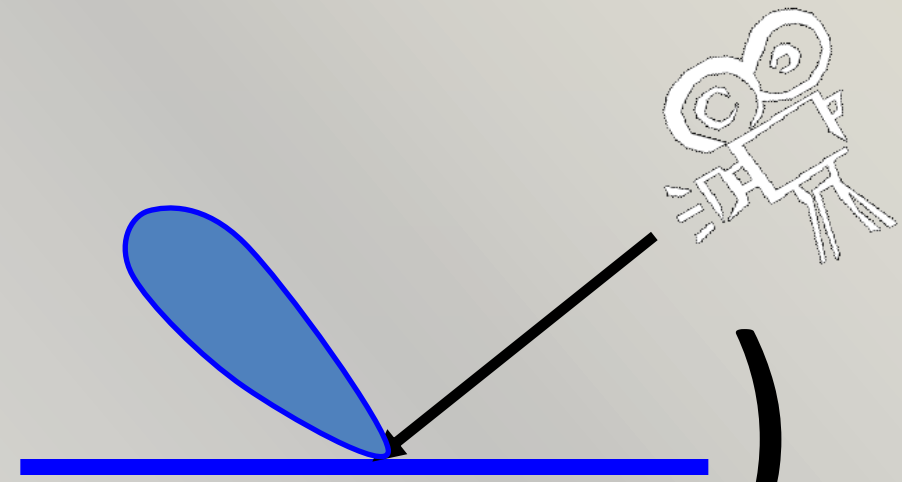


Figure 1: Radiance transfer at p from source to transferred incident to exit.

PRT

- Rendering: for each vertex / pixel, do

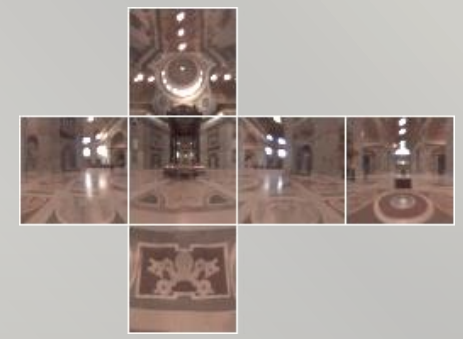
$\int ($  Environment map $\bullet$  BRDF $)$

Relighting / PRT as a Matrix-Vector Multiplication

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_N \end{bmatrix}$$

N are the number of pixels in output
M is the number of lighting directions

$$= \begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1M} \\ T_{21} & T_{22} & \cdots & T_{2M} \\ T_{31} & T_{32} & \cdots & T_{3M} \\ \vdots & \vdots & \ddots & \vdots \\ T_{N1} & T_{N2} & \cdots & T_{NM} \end{bmatrix} \begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_M \end{bmatrix}$$



Relighting as a Matrix-Vector Multiply

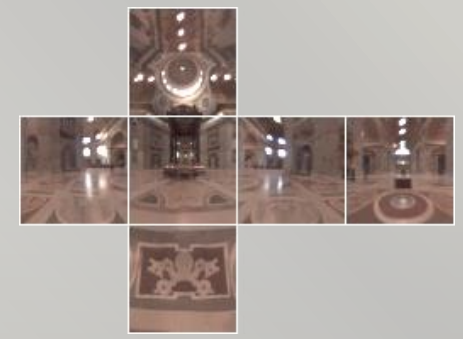
$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_N \end{bmatrix}$$

Output Image
(Pixel Vector)

Input Lighting
(Cubemap Vector)

Precomputed
Transport
Matrix

$$= \begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1M} \\ T_{21} & T_{22} & \cdots & T_{2M} \\ T_{31} & T_{32} & \cdots & T_{3M} \\ \vdots & \vdots & \ddots & \vdots \\ T_{N1} & T_{N2} & \cdots & T_{NM} \end{bmatrix} \begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_M \end{bmatrix}$$

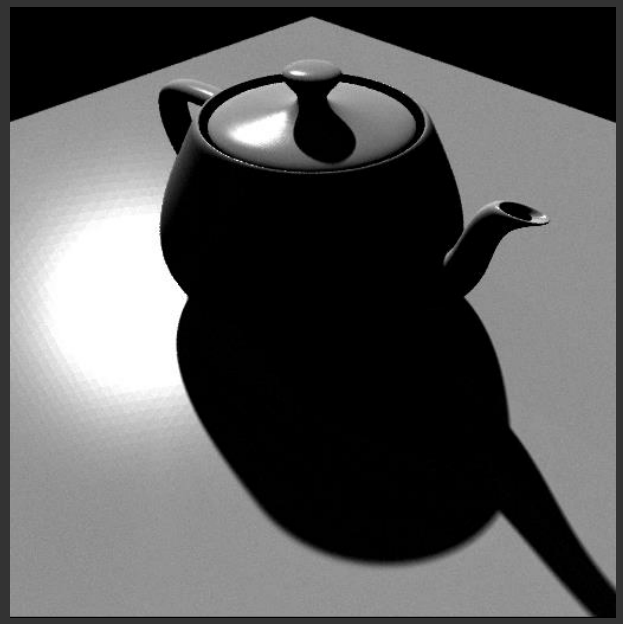


Matrix Columns (Images)

$$\begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1M} \\ T_{21} & T_{22} & \cdots & T_{2M} \\ T_{31} & T_{32} & \cdots & T_{3M} \\ \vdots & \vdots & \ddots & \vdots \\ T_{N1} & T_{N2} & \cdots & T_{NM} \end{bmatrix}$$

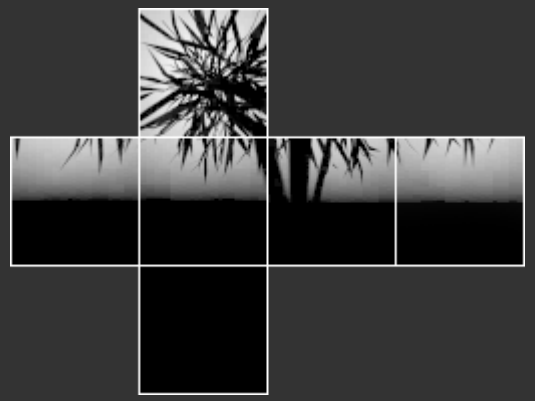
Precompute: Ray-Trace Image Cols

$$\begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1M} \\ T_{21} & T_{22} & \cdots & T_{2M} \\ T_{31} & T_{32} & \cdots & T_{3M} \\ \vdots & \vdots & \ddots & \vdots \\ T_{N1} & T_{N2} & \cdots & T_{NM} \end{bmatrix}$$



Precompute 2: Rasterize Matrix Rows

$$\begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1M} \\ T_{21} & T_{22} & \cdots & T_{2M} \\ T_{31} & T_{32} & \cdots & T_{3M} \\ \vdots & \vdots & \ddots & \vdots \\ T_{N1} & T_{N2} & \cdots & T_{NM} \end{bmatrix}$$



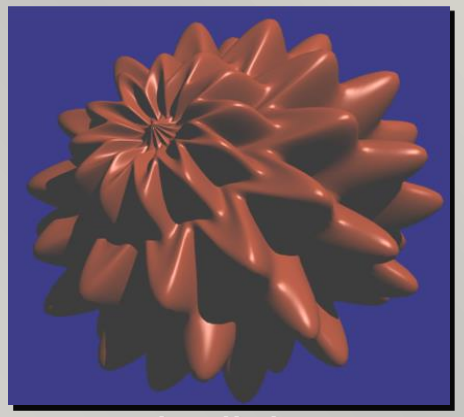
Solution

- Matrix is Enormous
  - NxM is large !!!
- Full matrix-vector multiplication is intractable
- *Compression*
  - *Made on the matrix row can achieve better compression ratio and quality*
  - ***Spherical harmonics-based PRT [Sloan et al. 02]***
  - (Local) factorization and PCA
  - Non-linear wavelet approximation

SH-based PRT

Better light integration and transport

- dynamic, env. lights
- self-shadowing
- interreflections



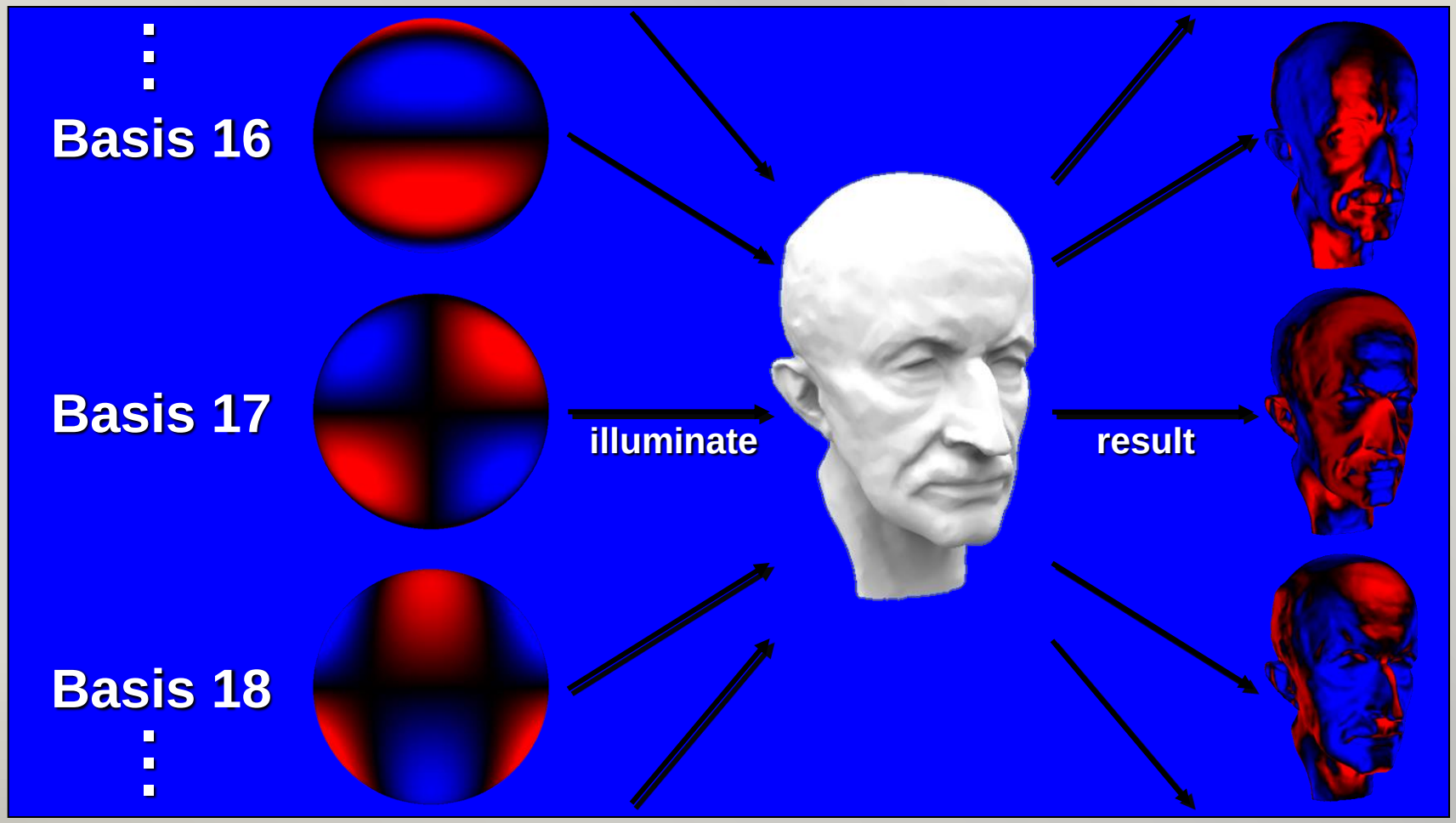
point light

Env. light

Env. lighting, no shadows

Env. lighting, shadows

SH-based PRT: Idea

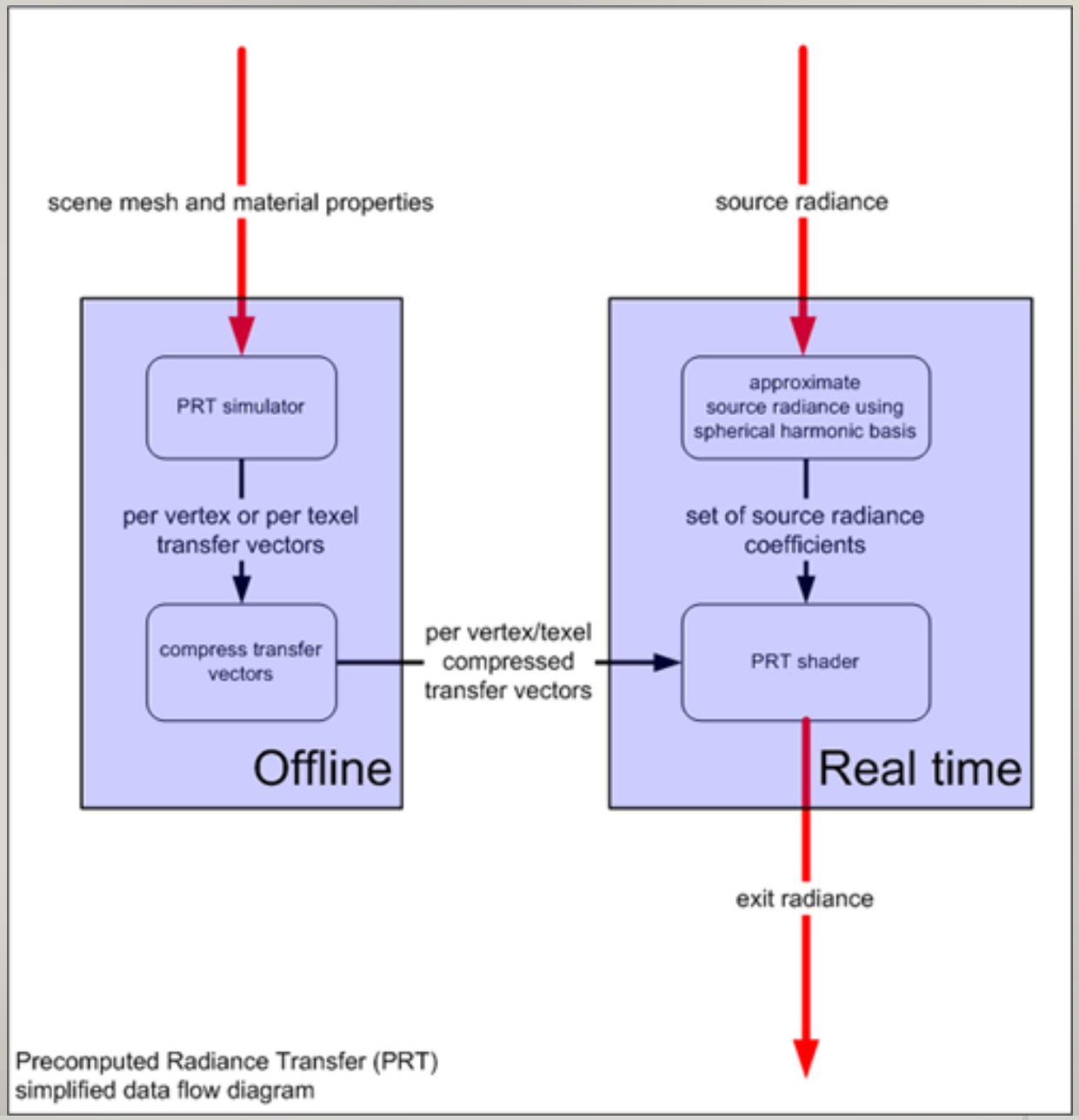


Idea of SH-based PRT

- The *L* vector is projected onto low-frequency components (say 25). Size greatly reduced.
- Hence, only 25 matrix columns
- But each pixel/vertex still treated separately
  - One RGB value per pixel/vertex:

    Diffuse shading / arbitrary BRDF shading w/ fixed view direction
  - SH coefficients of transferred radiance (25 RGB values per pixel/vertex for order 4 SH)

    Arbitrary BRDF shading w/ variable view direction
- Good technique (becoming common in games) but useful only for broad low-frequency lighting



Precomputed Radiance Transfer (PRT) simplified data flow diagram

Diffuse Transfer Results

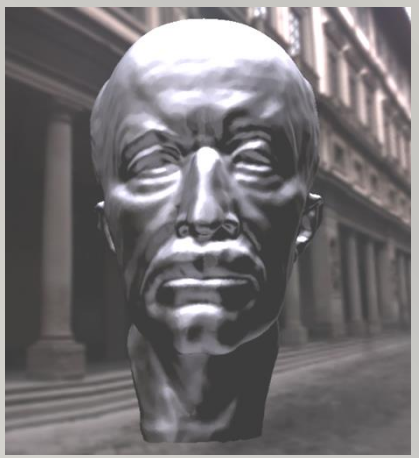
No Shadows/Inter

Shadows

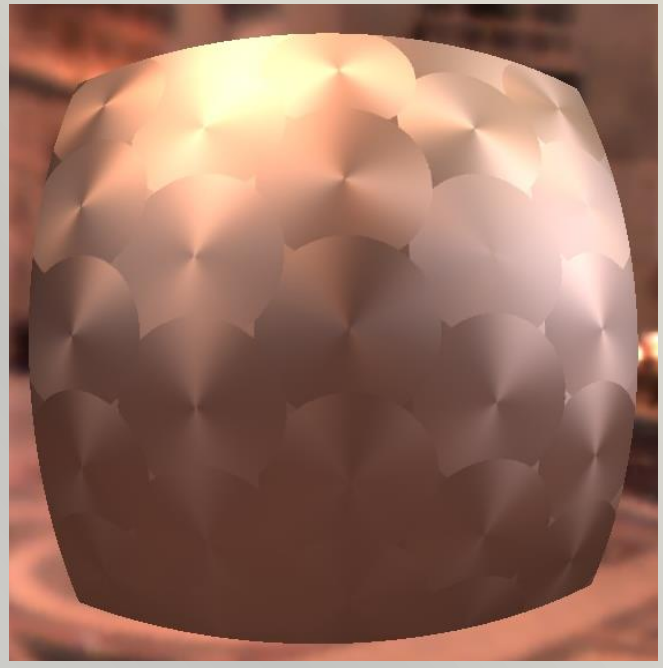
Shadows+Inter

Arbitrary BRDF Results

Anisotropic BRDFs



Other BRDFs



Spatially Varying

Summary

- Introduce ray-tracing which is a popular method for offline rendering
- Data structures for spatial partitioning as an essential mean to speed up ray-tracing
  - Octtree
  - BSP tree
- Brief idea of global illumination, its effect and issues
- PRT uses precomputation to achieve complex lighting effects in real-time