13. Advanced Rendering

Overview

- Ray Tracing
- Radisoity
- Parallel Rendering
- Reading: ANG ch. 13

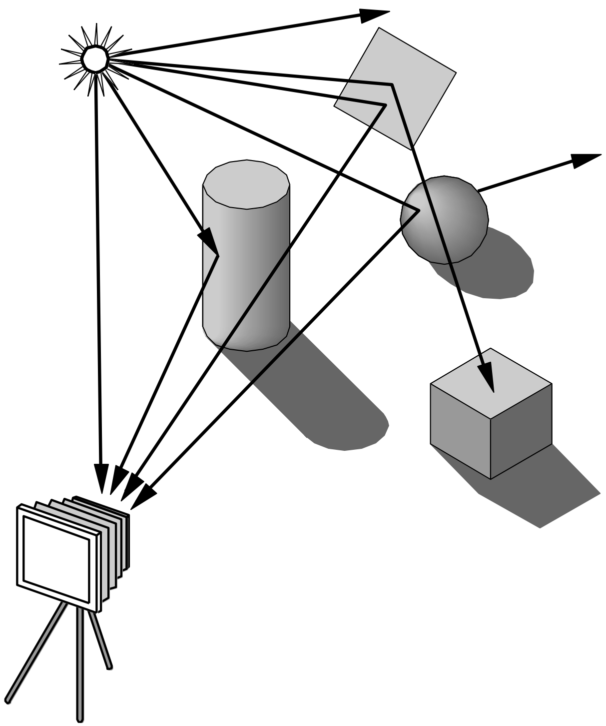
Ray Tracing

Introduction

- OpenGL is based on a pipeline model in which primitives are rendered one at time
  - No shadows (except by tricks or multiple renderings)
  - No multiple reflections
- Global approaches
  - Rendering equation
  - Ray tracing
  - Radiosity

Ray Tracing

- Follow rays of light from a point source
- Can account for reflection and transmission

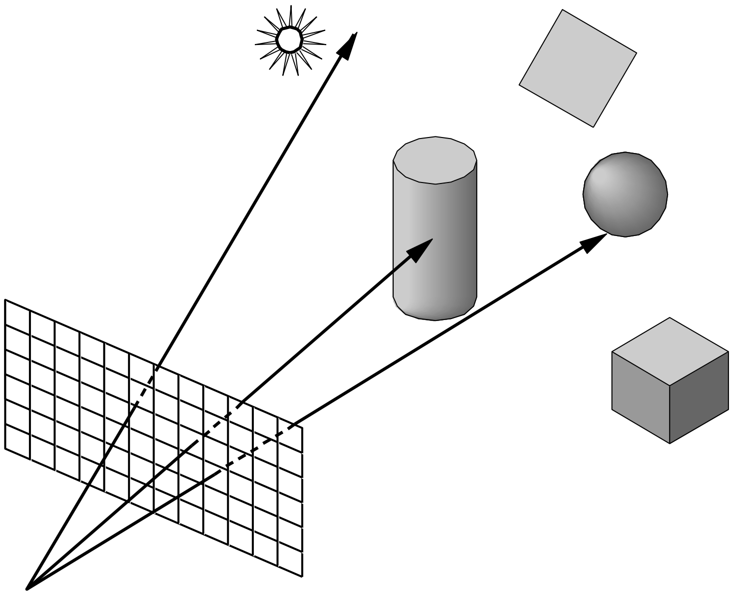


Computation

- Should be able to handle all physical interactions
- Ray tracing paradigm is not computational
- Most rays do not affect what we see
- Scattering produces many (infinite) additional rays
- Alternative: ray casting

Ray Casting

- Only rays that reach the eye matter
- Reverse direction and cast rays
- Need at least one ray per pixel

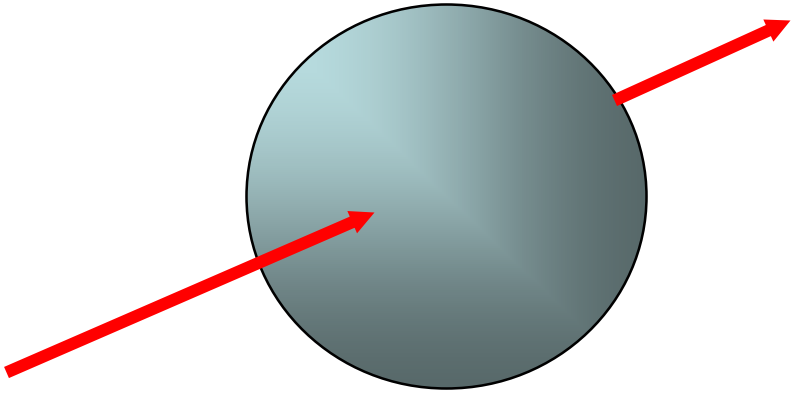


Ray Casting Quadrics

- Ray casting has become the standard way to visualize quadrics which are implicit surfaces in CSG systems
- Constructive Solid Geometry
  - Primitives are solids
  - Build objects with set operations
  - Union, intersection, set difference

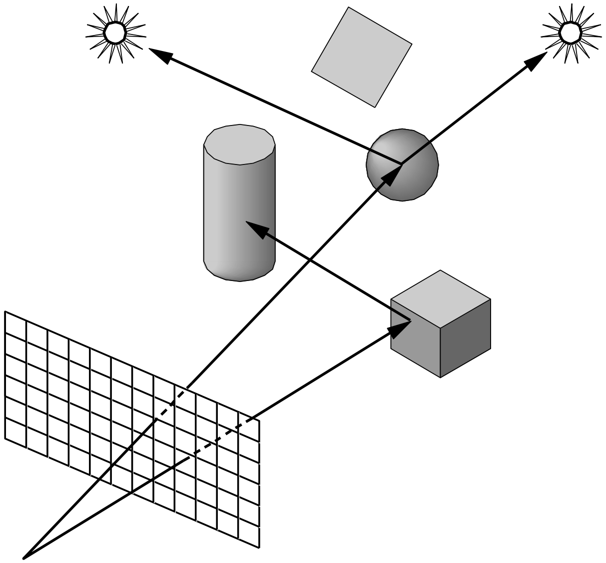
Ray Casting a Sphere

- Ray is parametric
- Sphere is quadric
- Resulting equation is a scalar quadratic equation which gives entry and exit points of ray (or no solution if ray misses)



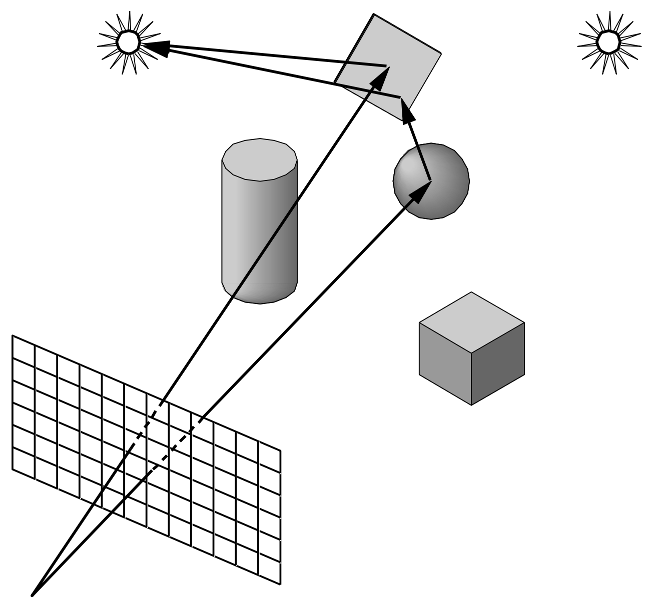
Shadow Rays

- Even if a point is visible, it will not be lit unless we can see a light source from that point
- Cast shadow or feeler rays

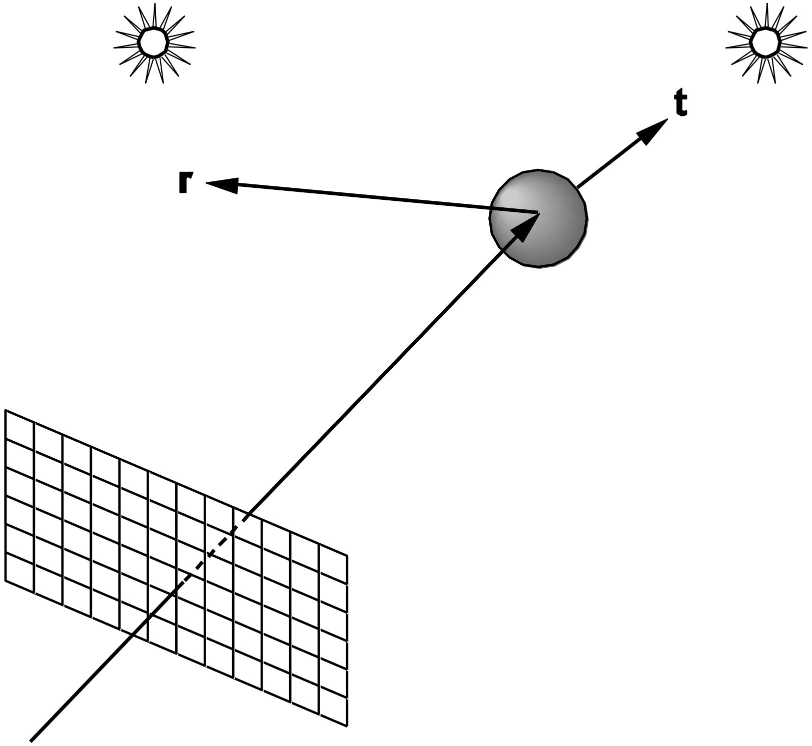


Reflection

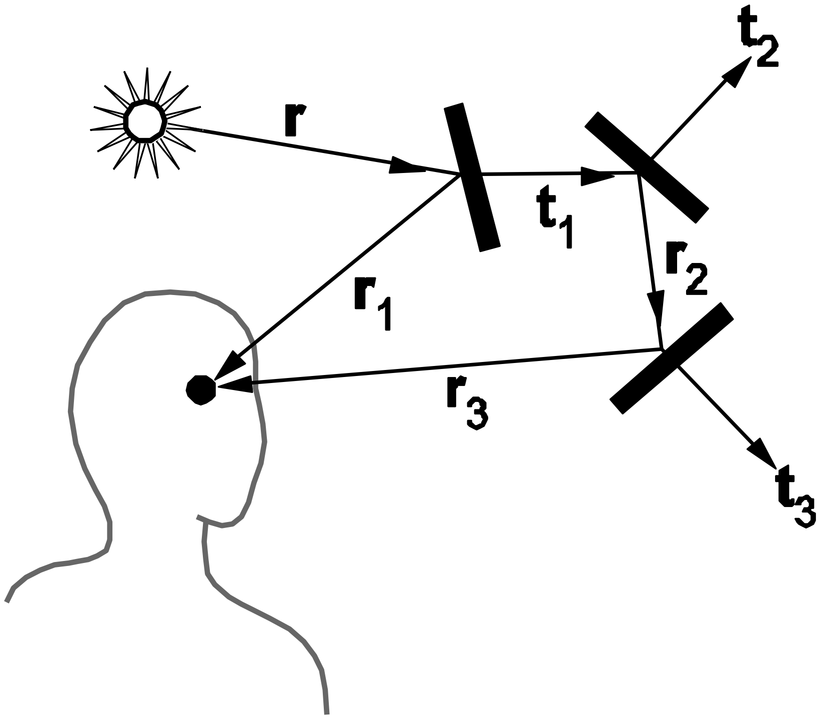
- Must follow shadow rays off reflecting or transmitting surfaces
- Process is recursive



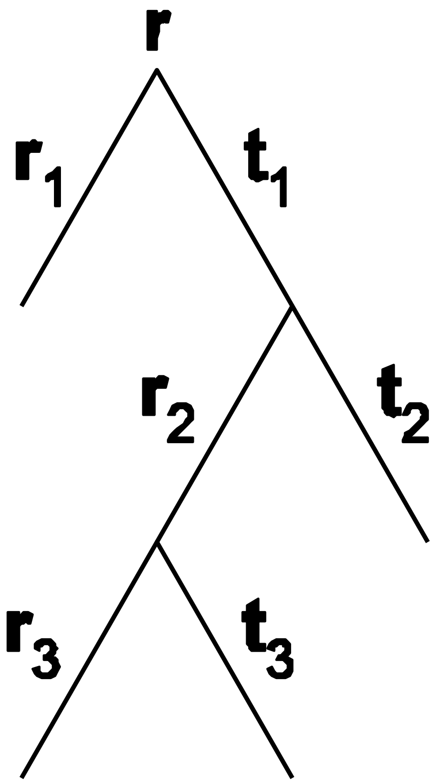
Reflection and Transmission



Ray Trees



Ray Tree



Diffuse Surfaces

- Theoretically the scattering at each point of intersection generates an infinite number of new rays that should be traced
- In practice, we only trace the transmitted and reflected rays but use the Phong model to compute shade at point of intersection
- Radiosity works best for perfectly diffuse (Lambertian) surfaces

Building a Ray Tracer

- Best expressed recursively
- Can remove recursion later
- Image based approach
  - For each ray …….
- Find intersection with closest surface
  - Need whole object database available
  - Complexity of calculation limits object types
- Compute lighting at surface
- Trace reflected and transmitted rays

When to stop

- Some light will be absorbed at each intersection
  - Track amount left
- Ignore rays that go off to infinity
  - Put large sphere around problem
- Count steps

Recursive Ray Tracer

```
color c = trace(point p, vector
 d, int step)
{
  color local, reflected,
 transmitted;
  point q;
  normal n;
  if(step > max)
 return(background_color);
```

Recursive Ray Tracer

```
q = intersect(p, d, status);
if(status==light_source)
 return(light_source_color);
if(status==no_intersection)
 return(background_color);

n = normal(q);
r = reflect(q, n);
t = transmit(q,n);
```

Recursive Ray Tracer

```
local = phong(q, n, r);
reflected = trace(q, r, step+1);
transmitted = trace(q,t,
 step+1);

return(local+reflected+
transmitted);
```

Computing Intersections

- Implicit Objects
  - Quadrics
- Planes
- Polyhedra
- Parametric Surfaces

Implicit Surfaces

Ray from $\mathbf{p}_0$ in direction $\mathbf{d}$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

General implicit surface

$$f(\mathbf{p}) = 0$$

Solve scalar equation

$$f(\mathbf{p}(t)) = 0$$

General case requires numerical methods

Quadrics

General quadric can be written as

$$\mathbf{p}^T\mathbf{A}\mathbf{p} + b^T\mathbf{p} + c = 0$$

Substitute equation of ray

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

to get quadratic equation

Sphere

$$(\mathbf{p} - \mathbf{p}_c) \bullet (\mathbf{p} - \mathbf{p}_c) - r^2 = 0$$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

$$\mathbf{p}_0 \bullet \mathbf{p}_0\, t^2 + 2\, \mathbf{p}_0 \bullet (\mathbf{d} - \mathbf{p}_0)\, t + (\mathbf{d} - \mathbf{p}_0) \bullet (\mathbf{d} - \mathbf{p}_0) - r^2 = 0$$

Planes

$$\mathbf{p} \bullet \mathbf{n} + c = 0$$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

$$t = \mathbf{-(p}_0 \bullet \mathbf{n} + c)/\ \mathbf{d} \bullet \mathbf{n}$$

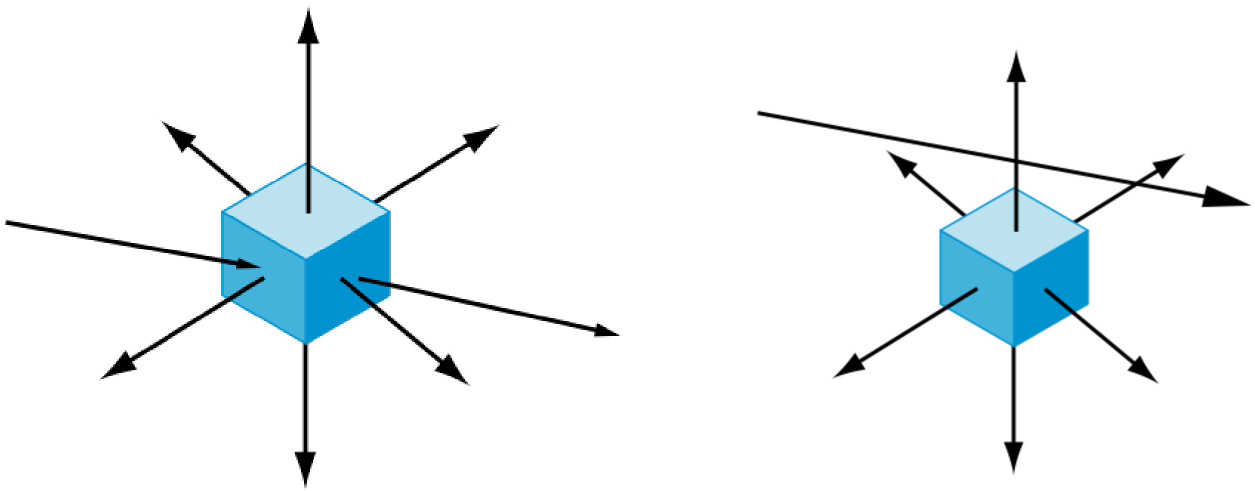
Polyhedra

- Generally we want to intersect with closed objects such as polygons and polyhedra rather than planes
- Hence we have to worry about inside/outside testing
- For convex objects such as polyhedra there are some fast tests

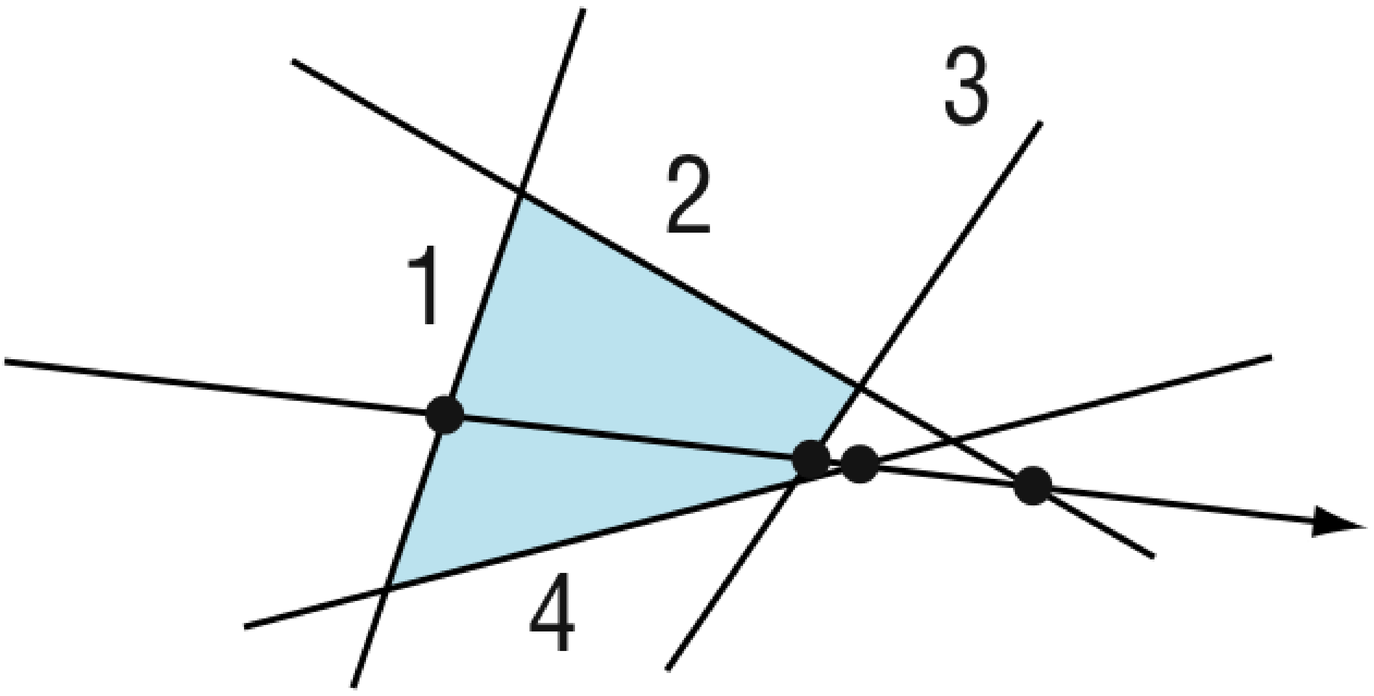
Ray Tracing Polyhedra

- If ray enters an object, it must enter a front facing polygon and leave a back facing polygon
- Polyhedron is formed by intersection of planes
- Ray enters at furthest intersection with front facing planes
- Ray leaves at closest intersection with back facing planes
- If entry is further away than exit, ray must miss the polyhedron

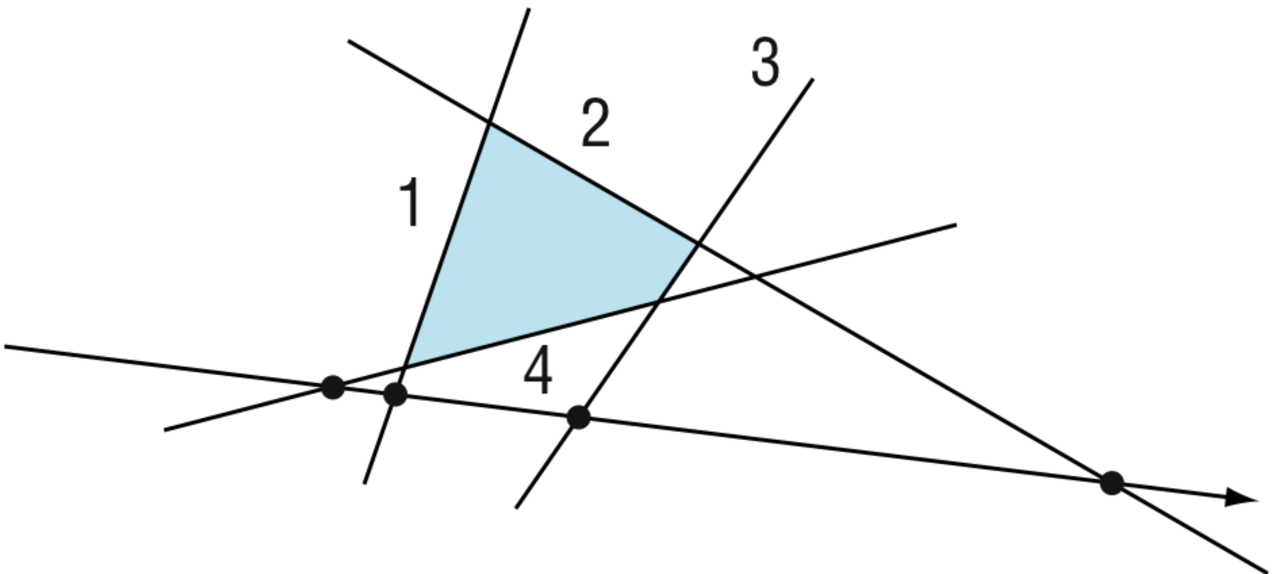
Ray Tracing Polyhedra



Ray Tracing a Polygon



Ray Tracing a Polygon



Radisoity

Introduction

- Ray tracing is best with many highly specular sufaces
  - Not characteristic of real scenes
- Rendering equation describes general shading problem
- Radiosity solves rendering equation for perfectly diffuse surfaces

Terminology

- Energy ~ light (incident, transmitted)
  - Must be conserved
- Energy flux = luminous flux = power = energy/unit time
  - Measured in **lumens**
  - Depends on wavelength so we can integrate over spectrum using **luminous efficiency curve** of sensor
- Energy density (Φ) = energy flux/unit area

Terminology

Intensity ~ brightness

– Brightness is perceptual

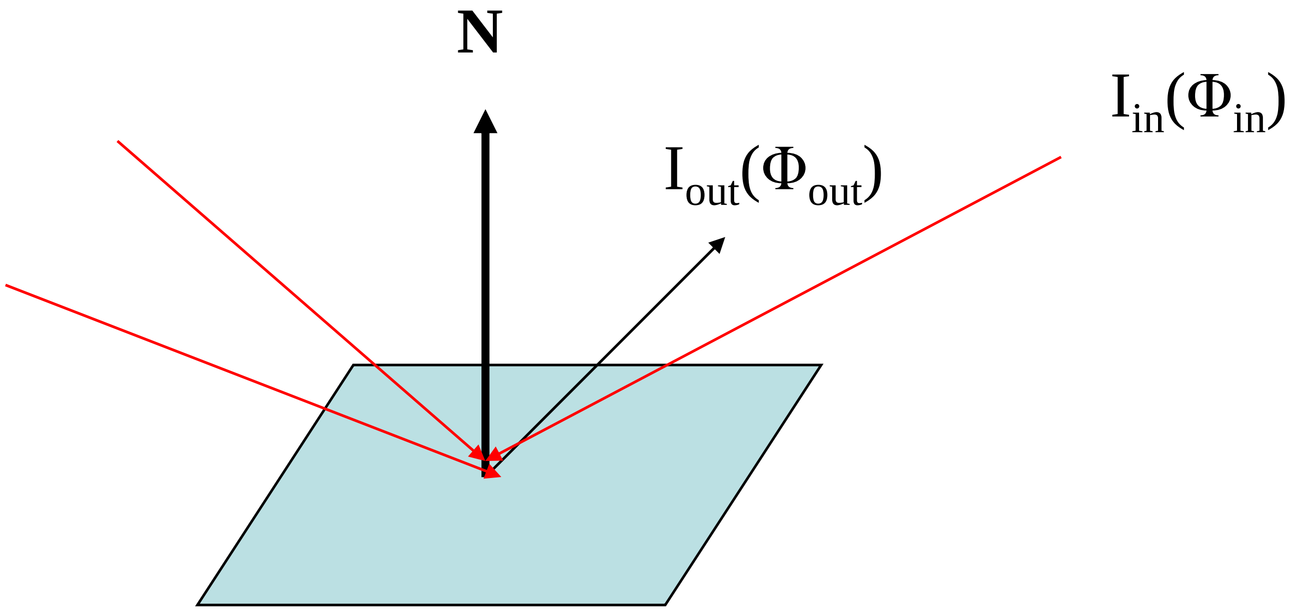
= flux/area-solid angle = power/unit projected area per solid angle

– Measured in **candela**

$$\Phi = \int \int I \, dA \, d\omega$$

Rendering Eqn (Kajiya)

- Consider a point on a surface



Rendering Equation

- Outgoing light is from two sources
  - Emission
  - Reflection of incoming light
- Must integrate over all incoming light
  - Integrate over hemisphere
- Must account for foreshortening of incoming light

Rendering Equation

$$I_{out}(\Phi_{out}) = E(\Phi_{out}) + \int_{2\pi} R_{bd}(\Phi_{out}, \Phi_{in})I_{in}(\Phi_{in}) \cos\theta \, d\omega$$

emission

angle between normal and Φin

bidirectional reflection coefficient

Note that angle is really two angles in 3D and wavelength is fixed

Rendering Equation

- Rendering equation is an energy balance
  - Energy in = energy out
- Integrate over hemisphere
- Fredholm integral equation
  - Cannot be solved analytically in general
- Various approximations of R_{bd} give standard rendering models
- Should also add an occlusion term in front of right side to account for other objects blocking light from reaching surface

Another version

Consider light at a point **p** arriving from **p**'

$$i(\mathbf{p}, \mathbf{p}') = \upsilon(\mathbf{p}, \mathbf{p}')(\varepsilon(\mathbf{p},\mathbf{p}') + \int \rho(\mathbf{p}, \mathbf{p}', \mathbf{p}'')i(\mathbf{p}', \mathbf{p}'')d\mathbf{p}'')$$

emission from **p**' to **p**

occlusion = 0 or $1/d^2$

light reflected at **p**' from all points **p**'' towards **p**

Radiosity

- Consider objects to be broken up into flat patches (which may correspond to the polygons in the model)
- Assume that patches are perfectly diffuse reflectors
- Radiosity = flux = energy/unit area/ unit time leaving patch

Notation

n patches numbered 1 to n

b_i = radiosity of patch I

a_i = area patch I

total intensity leaving patch i = $b_i a_i$

$e_i a_i$ = emitted intensity from patch I

ρ_i = reflectivity of patch I

f_{ij} = form factor = fraction of energy leaving patch j that reaches patch i

Radiosity Equation

energy balance

$$b_i a_i = e_i a_i + \rho_i \sum f_{ji} b_j a_j$$

reciprocity

$$f_{ij} a_i = f_{ji} a_j$$

radiosity equation

$$b_i = e_i + \rho_i \sum f_{ij} b_j$$

Matrix Form

$$\mathbf{b} = [b_i]$$

$$\mathbf{e} = [e_i]$$

$$\mathbf{R} = [r_{ij}] \qquad r_{ij} = \rho_i \text{ if } i \neq j \quad r_{ii} = 0$$

$$\mathbf{F} = [f_{ij}]$$

Matrix Form

$$\mathbf{b} = \mathbf{e} - \mathbf{RFb}$$

formal solution

$$\mathbf{b} = [\mathbf{I}-\mathbf{RF}]^{-1}\mathbf{e}$$

Not useful since n is usually very large
Alternative: use observation that F is sparse

We will consider determination of form factors later

Solving the Radiosity Equation

For sparse matrices, iterative methods usually require only O(n) operations per iteration

Jacobi's method

$$\mathbf{b}^{k+1} = \mathbf{e} - \mathbf{RFb}^{k}$$

Gauss-Seidel: use immediate updates

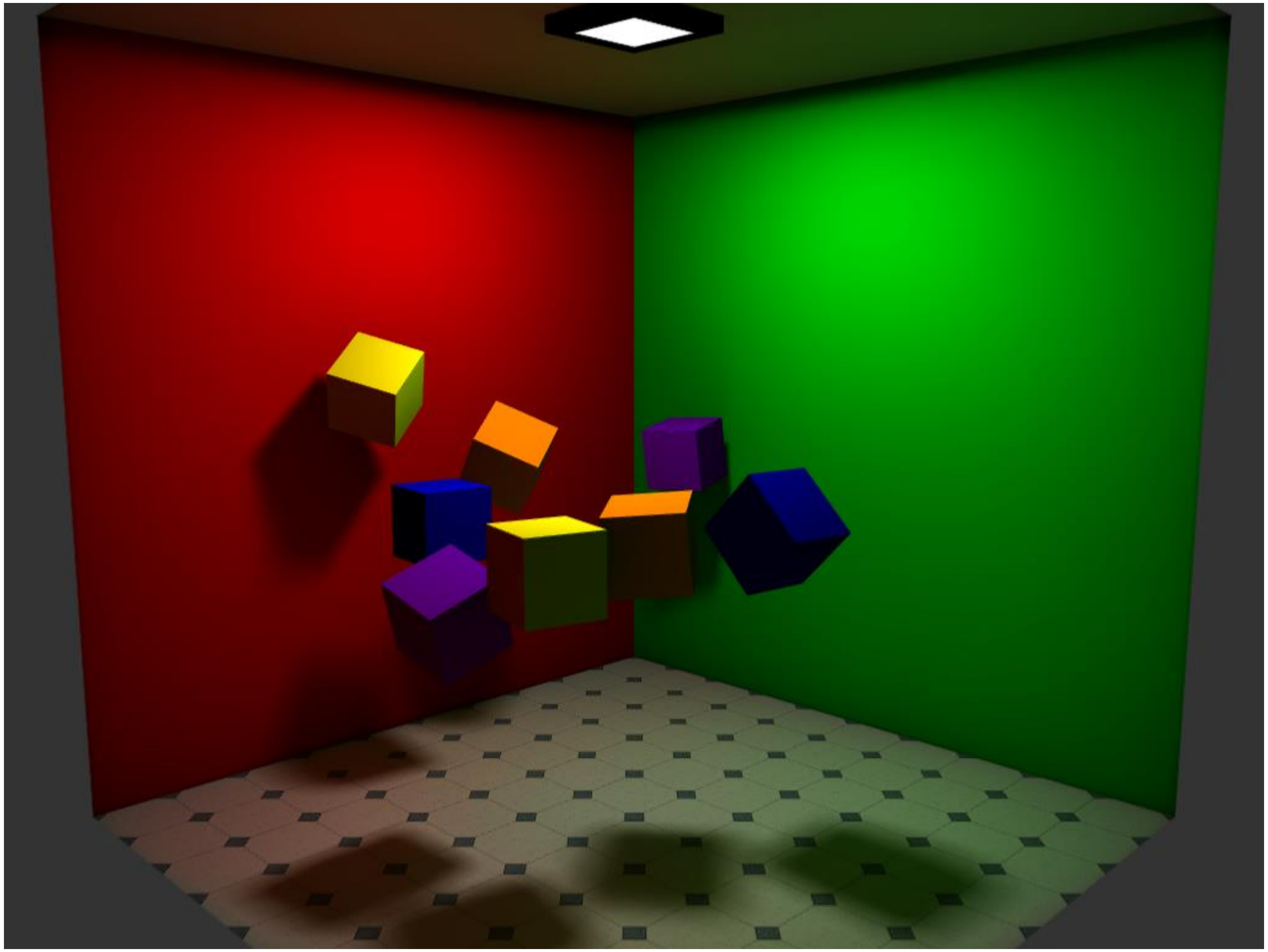
Series Approximation

$$1/(1-x) = 1 + x + x^2 + \ldots\ldots$$

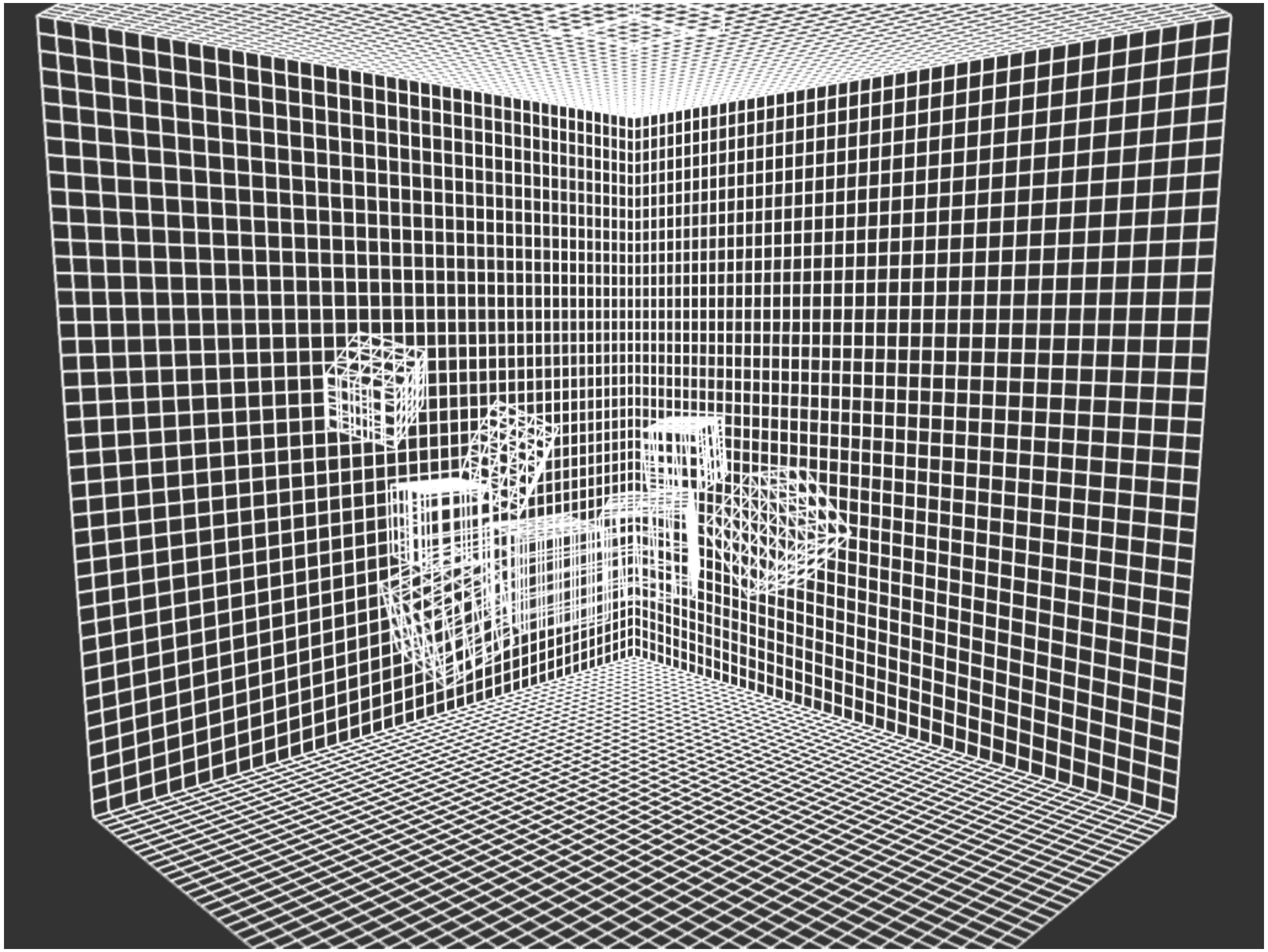
$$[\mathbf{I}-\mathbf{RF}]^{-1} = \mathbf{I} + \mathbf{RF} + (\mathbf{RF})^2 + \ldots$$

$$\mathbf{b} = [\mathbf{I}-\mathbf{RF}]^{-1}\mathbf{e} = \mathbf{e} + \mathbf{RFe} + (\mathbf{RF})^2\mathbf{e} + \ldots$$

Rendered Image

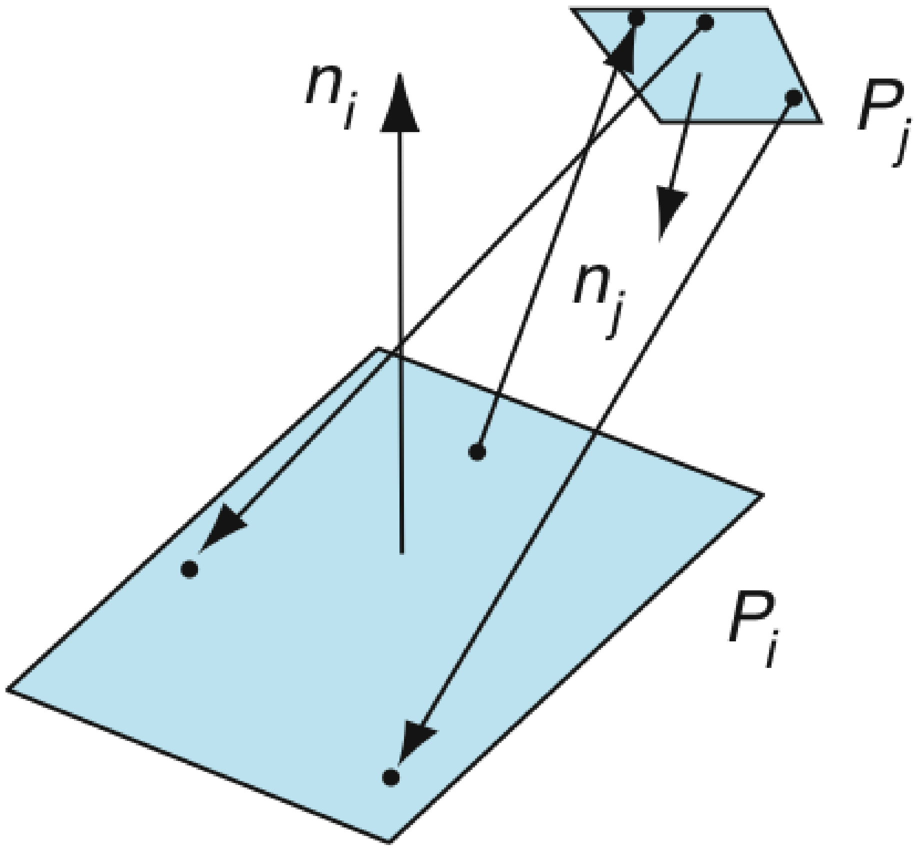


Patches

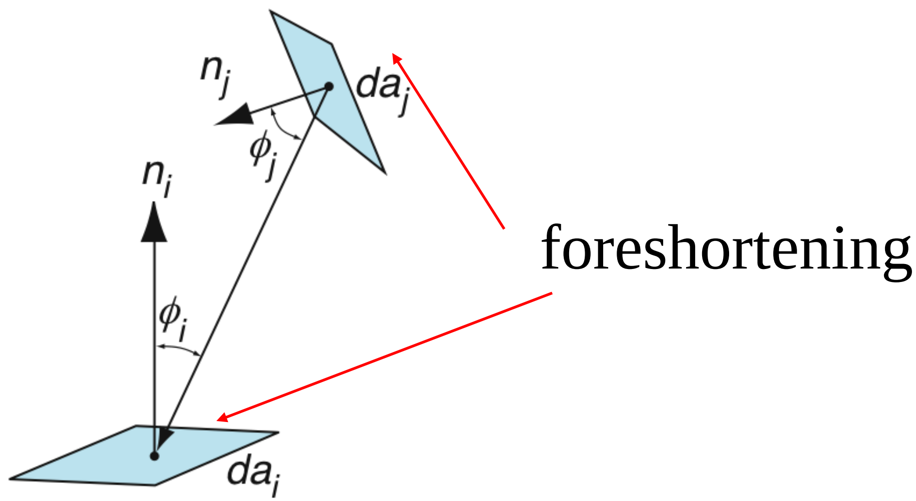


Computing Form Factors

- Consider two flat patches



Using Differential Patches



Form Factor Integral

$$f_{ij} = (1/a_i) \int_{ai} \int_{ai} (o_{ij} \cos \theta_i \cos \theta_j / \pi r^2) da_i \, da_j$$

occlusion

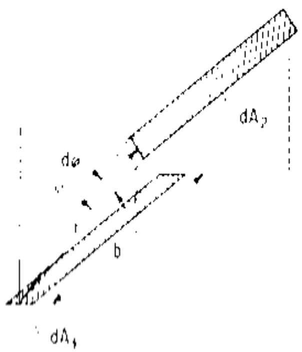
foreshortening of patch j

foreshortening of patch i

Solving the Intergral

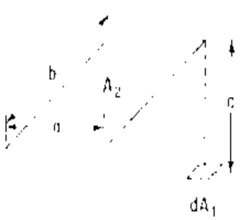
- There are very few cases where the integral has a (simple) closed form solution
  - Occlusion further complicates solution
- Alternative is to use numerical methods
- Two step process similar to texture mapping
  - Hemisphere
  - Hemicube

Form Factor Examples 1



Strip of finite length b and of differential width, to differential strip of same length on parallel generating line.

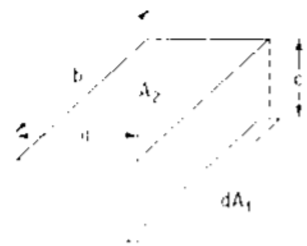
$$dF_{d1-d2} = \frac{\cos\phi}{\pi}\, d\phi \tan^{-1}\frac{b}{r}$$



Plane element dA_1 to plane parallel rectangle; normal to element passes through corner of rectangle.

$X = \frac{a}{c}$ $\quad Y = \frac{b}{c}$

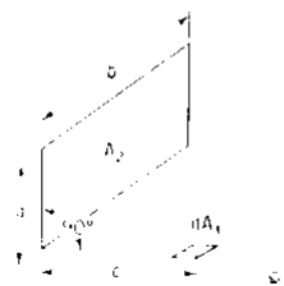
$$F_{d1-2} = \frac{1}{2\pi}\left(\frac{X}{\sqrt{1+X^2}}\tan^{-1}\frac{Y}{\sqrt{1+X^2}} + \frac{Y}{\sqrt{1+Y^2}}\tan^{-1}\frac{X}{\sqrt{1+Y^2}}\right)$$



Strip element to rectangle in plane parallel to strip; strip is opposite one edge of rectangle.

$X = \frac{a}{c}$ $\quad Y = \frac{b}{c}$

$$F_{d1-2} = \frac{1}{\pi Y}\left[\sqrt{1+Y^2}\tan^{-1}\frac{X}{\sqrt{1+Y^2}} - \tan^{-1}X + \frac{XY}{\sqrt{1+X^2}}\tan^{-1}\frac{Y}{\sqrt{1+X^2}}\right]$$



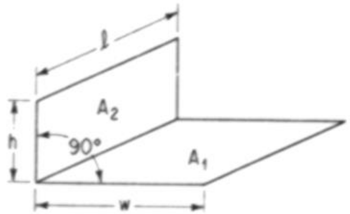
Plane element dA_1 to rectangle in plane 90° to plane of element.

$X = \frac{a}{b}$ $\quad Y = \frac{c}{b}$

$$F_{d1-2} = \frac{1}{2\pi}\left[\tan^{-1}\frac{1}{Y} - \frac{Y}{\sqrt{X^2+Y^2}}\tan^{-1}\frac{1}{\sqrt{X^2+Y^2}}\right]$$

Form Factor Examples 2

12

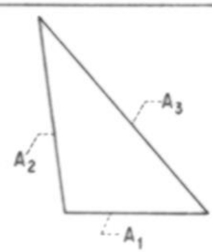


Two finite rectangles of same length, having one common edge, and at an angle of 90° to each other.

$$H = \frac{h}{l} \qquad W = \frac{w}{l}$$

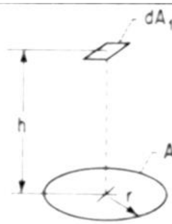
$$F_{1-2} = \frac{1}{\pi W}\left(W \tan^{-1}\frac{1}{W} + H \tan^{-1}\frac{1}{H} - \sqrt{H^2 + W^2}\tan^{-1}\frac{1}{\sqrt{H^2 + W^2}} + \frac{1}{4}\ln\left\{\left[\frac{(1 + W^2)(1 + H^2)}{(1 + W^2 + H^2)}\right]\left[\frac{W^2(1 + W^2 + H^2)}{(1 + W^2)(W^2 + H^2)}\right]^{W^2}\left[\frac{H^2(1 + H^2 + W^2)}{(1 + H^2)(H^2 + W^2)}\right]^{H^2}\right\}\right)$$

13



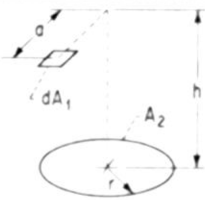
Infinitely long enclosure formed by three plane areas.

$$F_{1-2} = \frac{A_1 + A_2 - A_3}{2A_1}$$



Plane element dA_1 to circular disk in plane parallel to element; normal to element passes through center of disk.

$$F_{d1-2} = \frac{r^2}{h^2 + r^2}$$

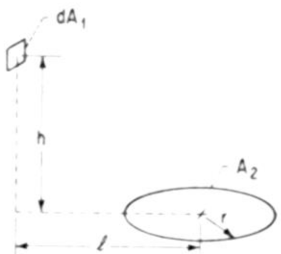


Plane element dA_1 to circular disk in plane parallel to element.

$$H = \frac{h}{a} \qquad R = \frac{r}{a}$$

$$Z = 1 + H^2 + R^2$$

$$F_{d1-2} = \frac{1}{2}\left(1 - \frac{1 + H^2 - R^2}{\sqrt{Z^2 - 4R^2}}\right)$$



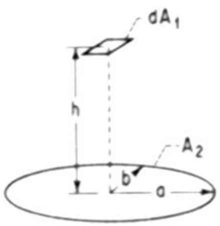
Plane element dA_1 to circular disk; planes containing element and disk intersect at 90°.

$$H = \frac{h}{l} \qquad R = \frac{r}{l}$$

$$Z = 1 + H^2 + R^2$$

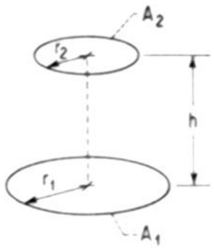
$$F_{d1-2} = \frac{H}{2}\left(\frac{Z}{\sqrt{Z^2 - 4R^2}} - 1\right)$$

Form Factor Examples 3



Plane element dA_1 to elliptical plate in plane parallel to element; normal to element passes through center of plate.

$$F_{d1-2} = \frac{ab}{\sqrt{(h^2 + a^2)(h^2 + b^2)}}$$

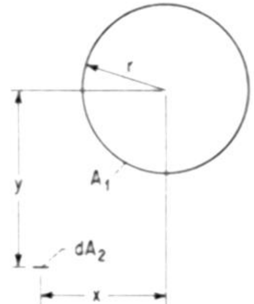


Parallel circular disks with centers along the same normal.

$$R_1 = \frac{r_1}{h} \qquad R_2 = \frac{r_2}{h}$$

$$X = 1 + \frac{1 + R_2^2}{R_1^2}$$

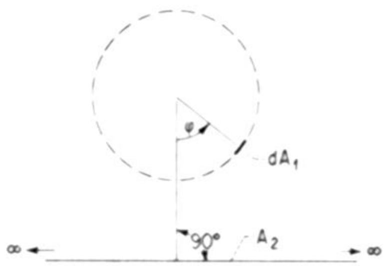
$$F_{1-2} = \frac{1}{2}\left[X - \sqrt{X^2 - 4\left(\frac{R_2}{R_1}\right)^2}\right]$$



Strip element dA_2 of any length to infinitely long cylinder.

$$X = \frac{x}{r} \qquad Y = \frac{y}{r}$$

$$F_{d2-1} = \frac{Y}{X^2 + Y^2}$$



Element of any length on cylinder to plane of infinite length and width.

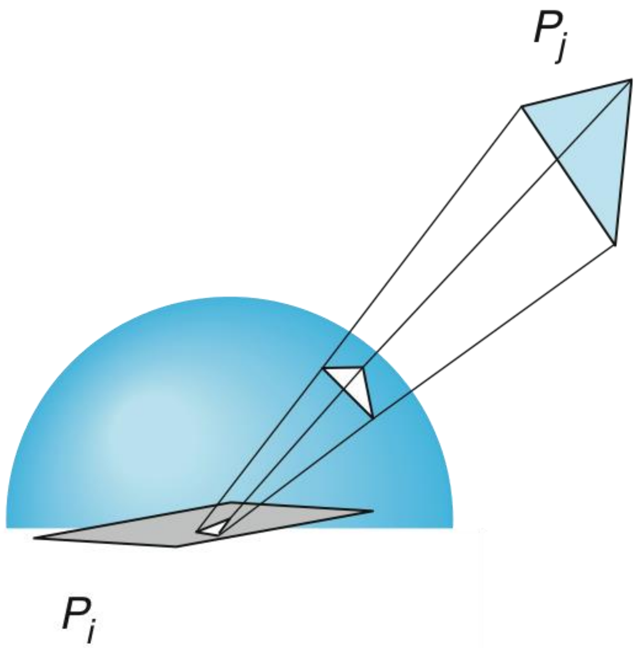
$$F_{d1-2} = \tfrac{1}{2}(1 + \cos\varphi)$$

Hemisphere

- Use illuminating hemisphere
- Center hemisphere on patch with normal pointing up
- Must shift hemisphere for each point on patch

Hemisphere

- Projecting patch on a hemisphere

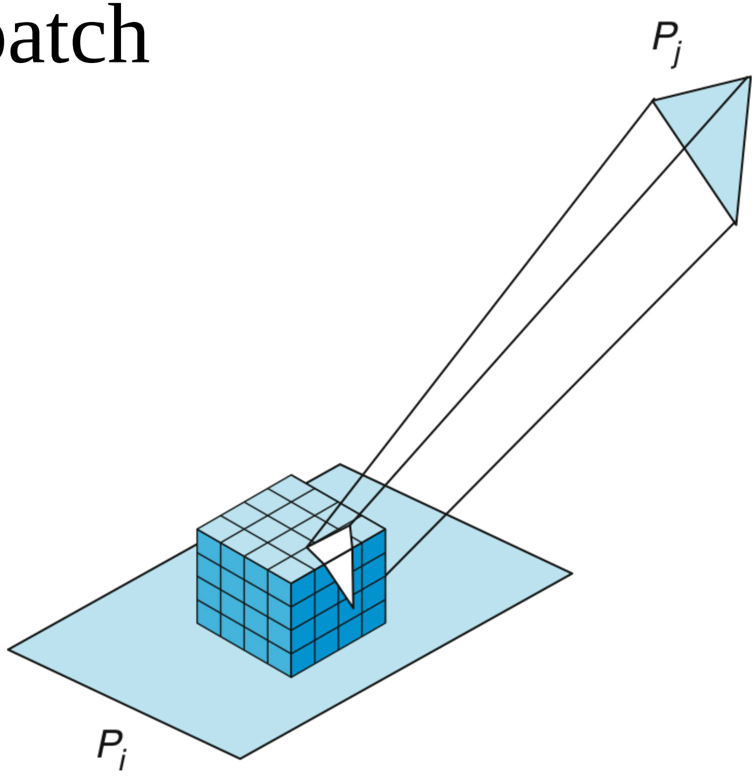


Hemicube

- Easier to use a hemicube instead of a hemisphere
- Rule each side into “pixels”
- Easier to project on pixels which give *delta form factors* that can be added up to give desired from factor
- To get a delta form factor we need only cast a ray through each pixel

Hemicube

- Projecting patch on a hemicube and onto another patch



Instant Radiosity

- Want to use graphics system if possible
- Suppose we make one patch emissive
- The light from this patch is distributed among the other patches
- Shade of other patches ~ form factors
- Must use multiple OpenGL point sources to approximate a uniformly emissive patch

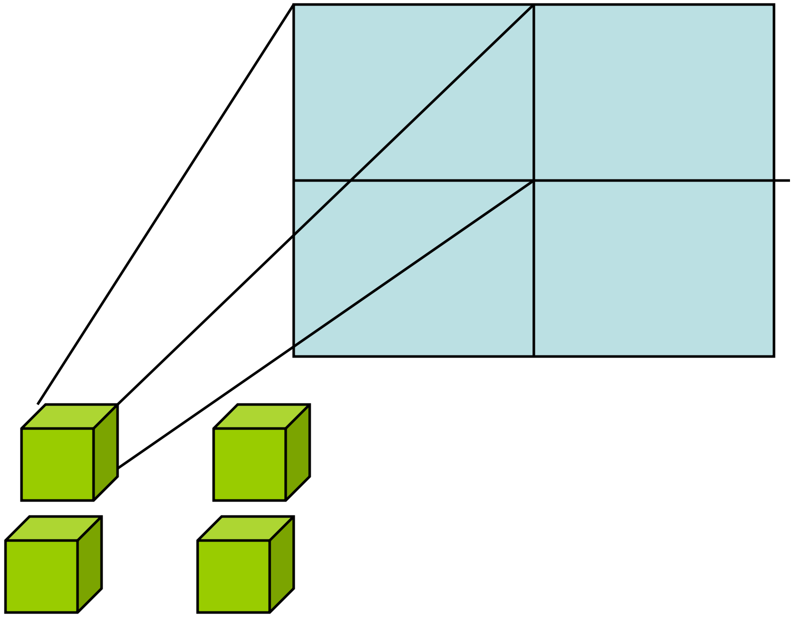
Parallel Rendering

Introduction

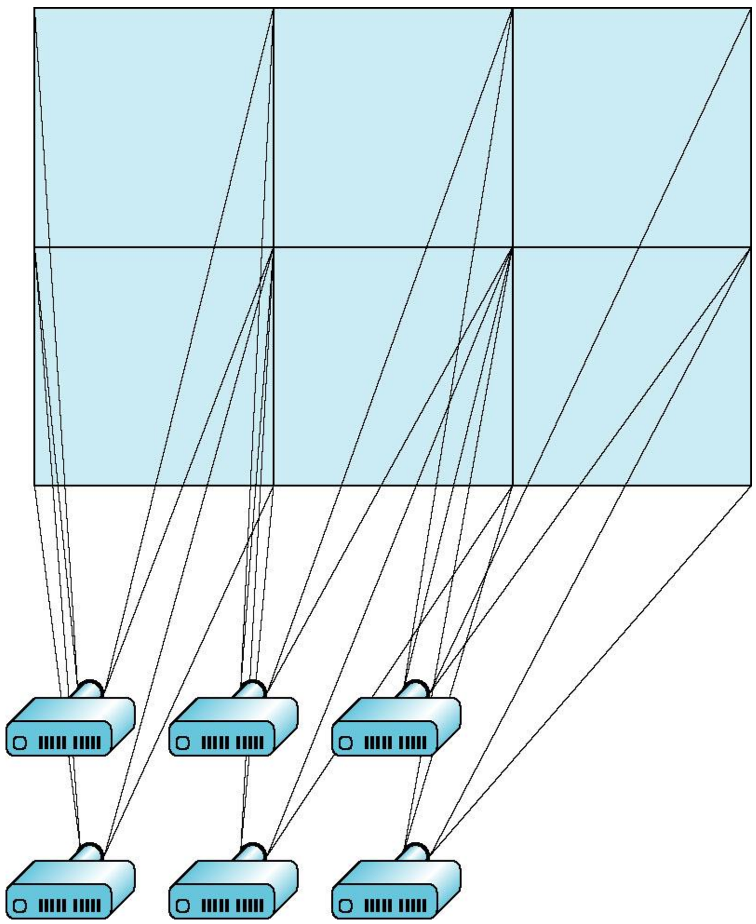
- In many situations, a standard rendering pipeline might not be sufficient
  - –Need higher resolution display
  - –More primitives than one pipeline can handle
- Want to use commodity components to build a system that can render in parallel
- Use standard network to connect

Power Walls

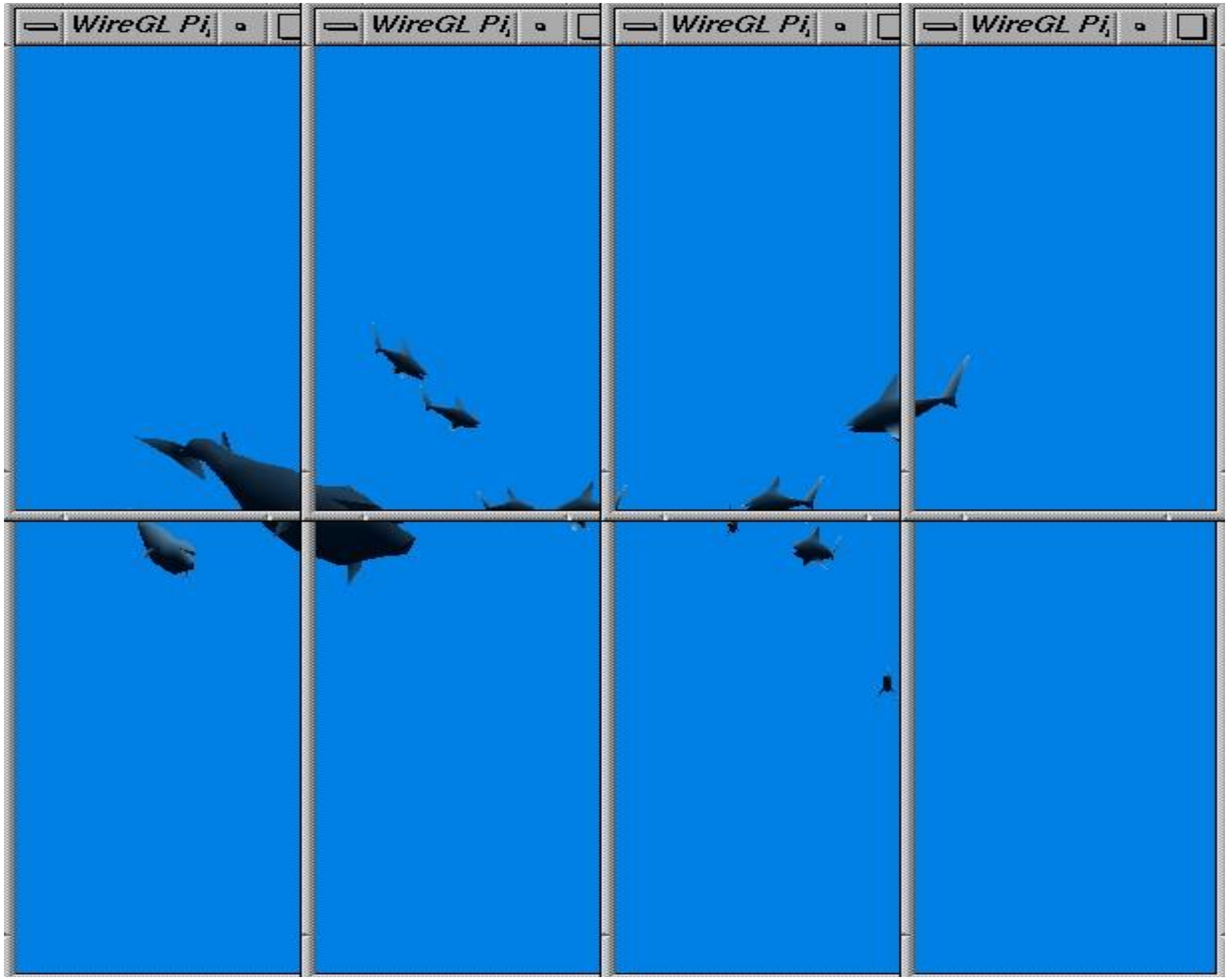
- Where do we display large data sets?
  - CRTs have low resolution (1 Mpixel)
  - LCD panels improving but still expensive
  - Need resolution comparable to data set to see detail
    - CT/MRI/MEG
    - Ocean data
- Solution?
  - Multiple projectors
    - Commodity
    - High-end
    - See IEECE CG & A (July)



Power wall using six orojectors



Tiled Display

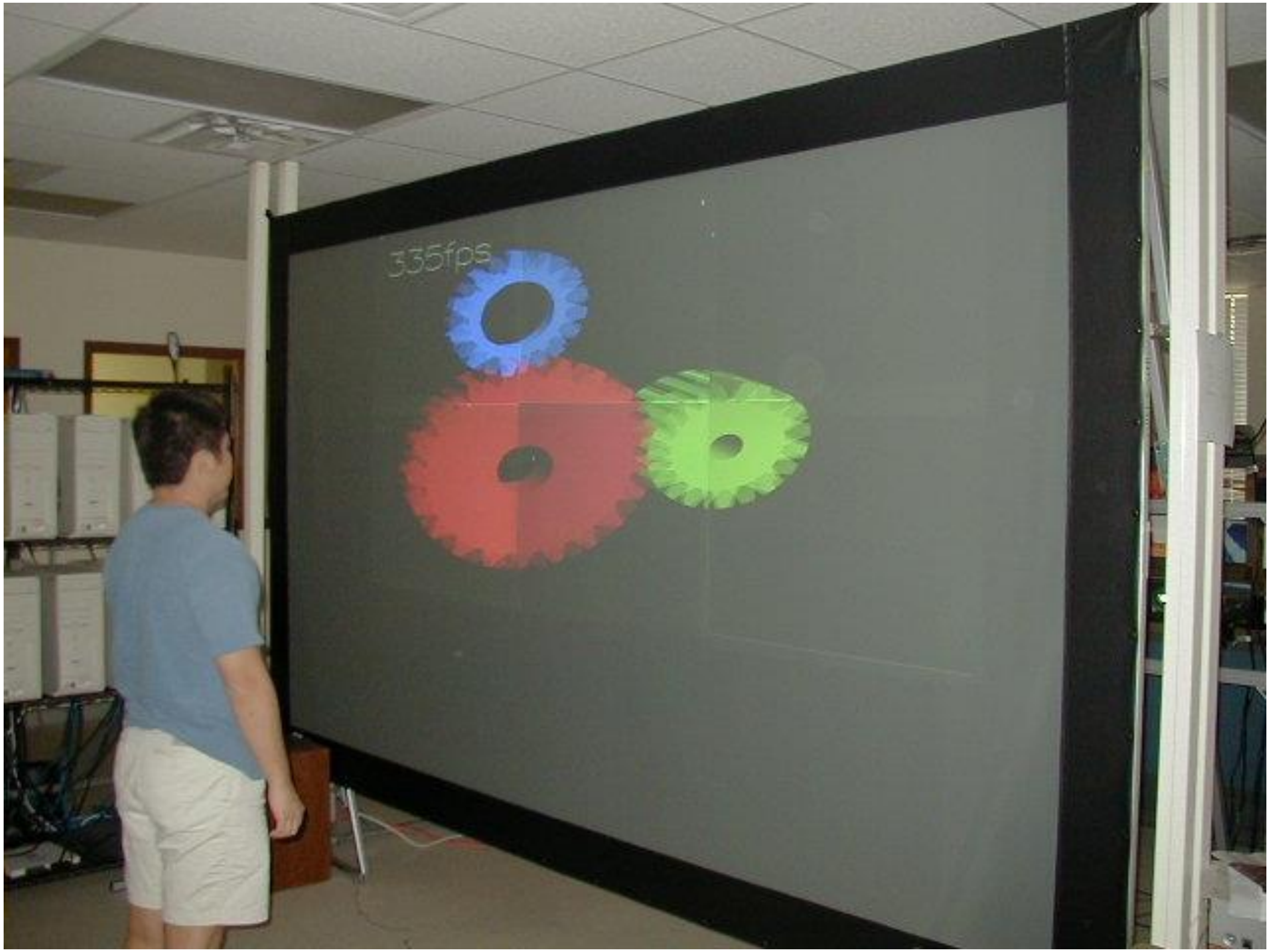


CS Power Wall

- 6 dual processor Intellestations
- G Force 3 Graphics cards
- 6 commodity projectors (1024 x 768)
- Gigabit ethernet
- Back projected screen
- Shared facility with scalable system group
  – Investigate OS and network issues

CS Power Wall

CS Power Wall

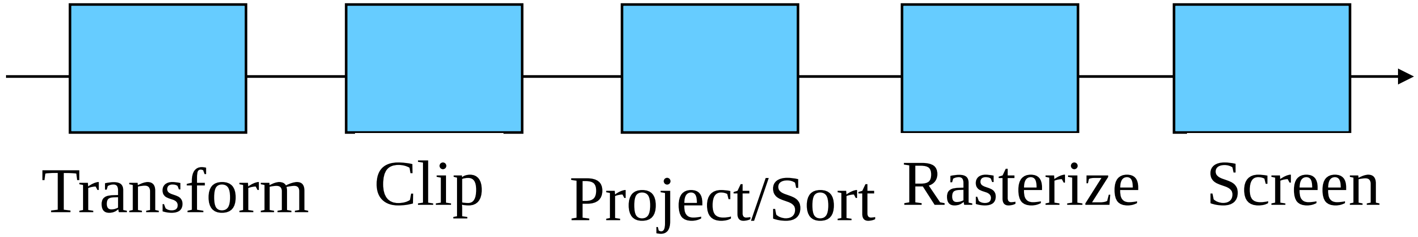


Power Wall

- Inexpensive solution but there are some problems
  - Color matching
  - Vignetting
  - Alignment
    - Overlap areas
  - Synching
  - Dark field

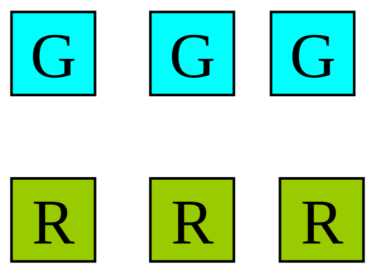
Graphics Architectures

- Pipeline Architecture
  - SGI Geometry Engine
  - Geometry passes through pipeline
  - Hardware for
    - clipping
    - transformations
    - texture mapping



Building Blocks

- Graphics processors consist of geometric blocks and rasterizers
- Geometric units: transformations, clipping, lighting
- Rasterization: scan conversion, shading
- Parallelize by using mutiple blocks
- Where to do depth check?

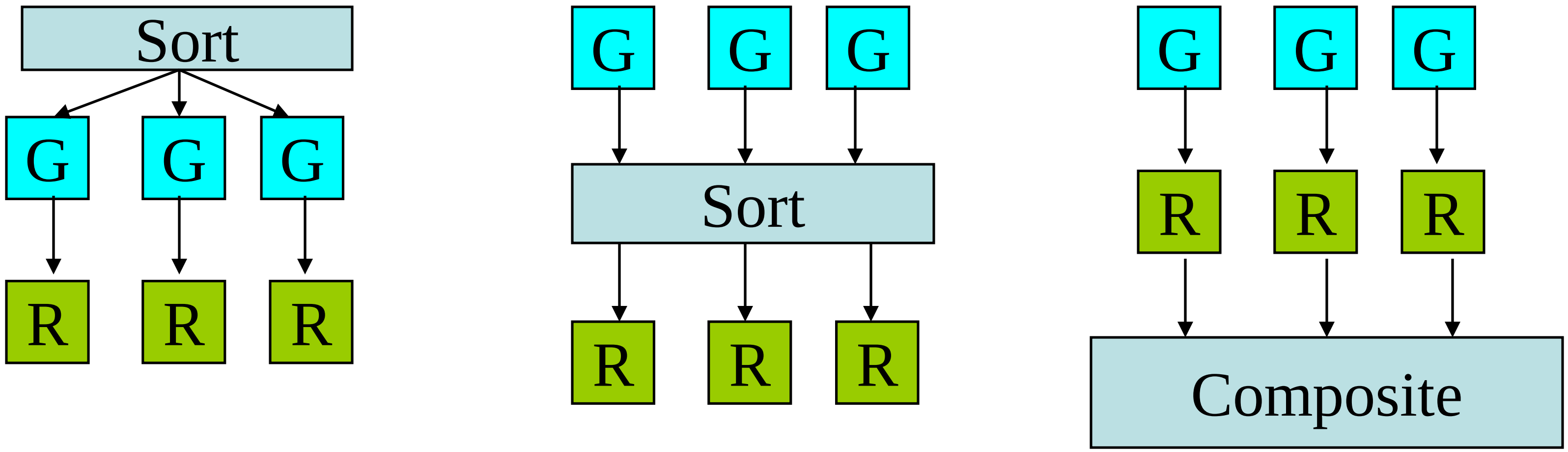


Sorting Paradigm

- We can categorize different ways of interconnecting blocks using a sorting paradigm: each projector is responsible for one area of the screen. Hence, we must sort the primitives and assign them to the proper projector

- Algorithms can be categorized by where this sorting occurs

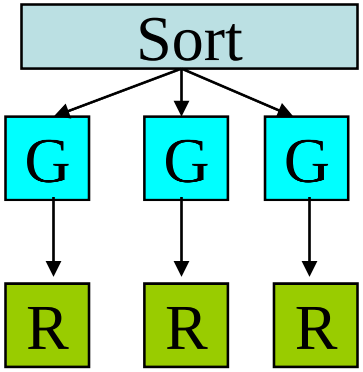
Three Rendering Methods



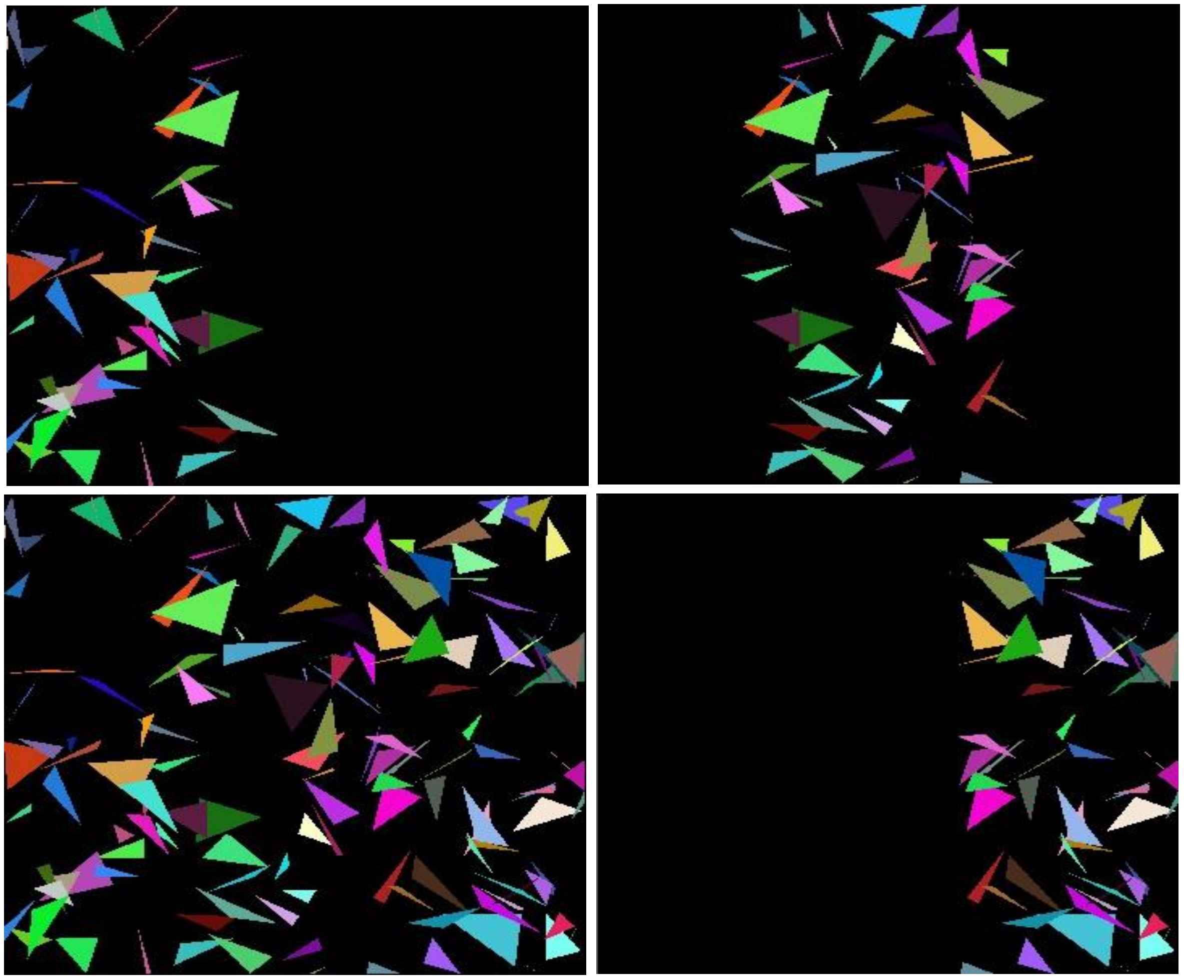
Sort-First Rendering **Sort-Middle Rendering** **Sort-Last Rendering**

Sort First

- Each rasterization unit assigned to an area of the screen
- Each geometric unit coupled to its own rasterizer
- Must sort primitives first
- Can use commodity cards

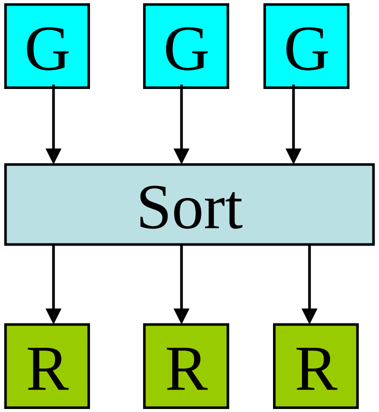


Sort-First Rendering for a Random Triangles Application



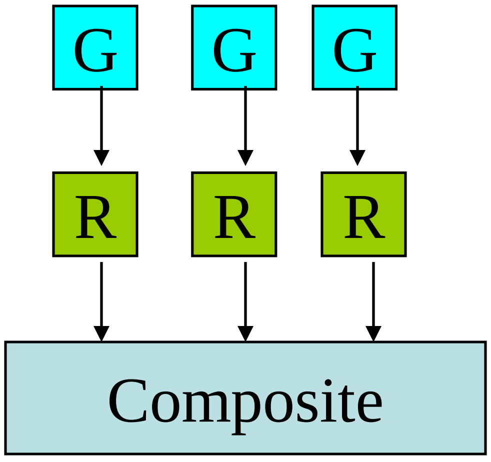
Sort Middle

- Geometric units and rasterization units decoupled
- Each geometric unit can be assigned any group of objects
- Each rasterizer is assigned to an area of the screen
- Must sort between stages



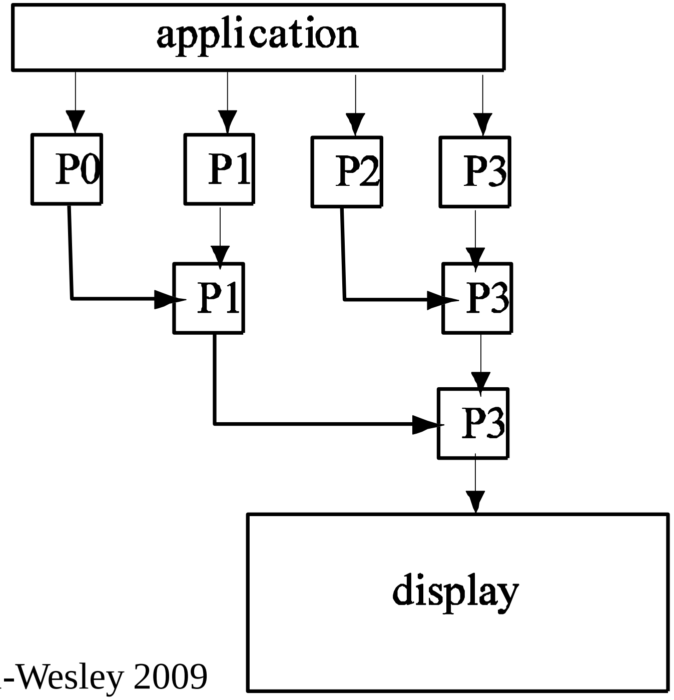
Sort Last

- Couple rasterizers and geometric units
- Assign objects to geometric units to load balance or via application
- Composite results at end



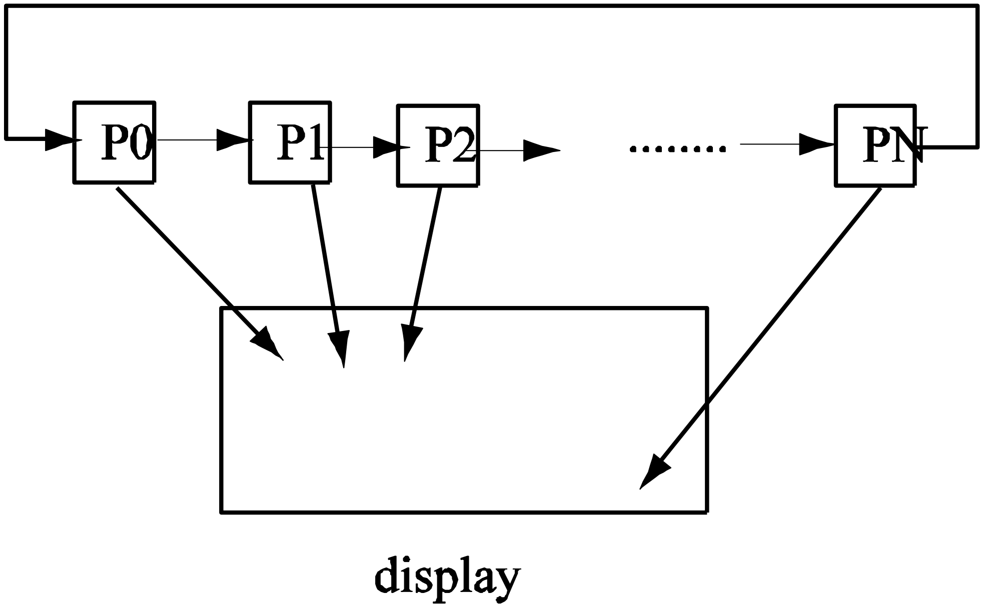
Tree Compositing

- Composite in pairs
- Send color and depth buffers
- Each time half processors become idle

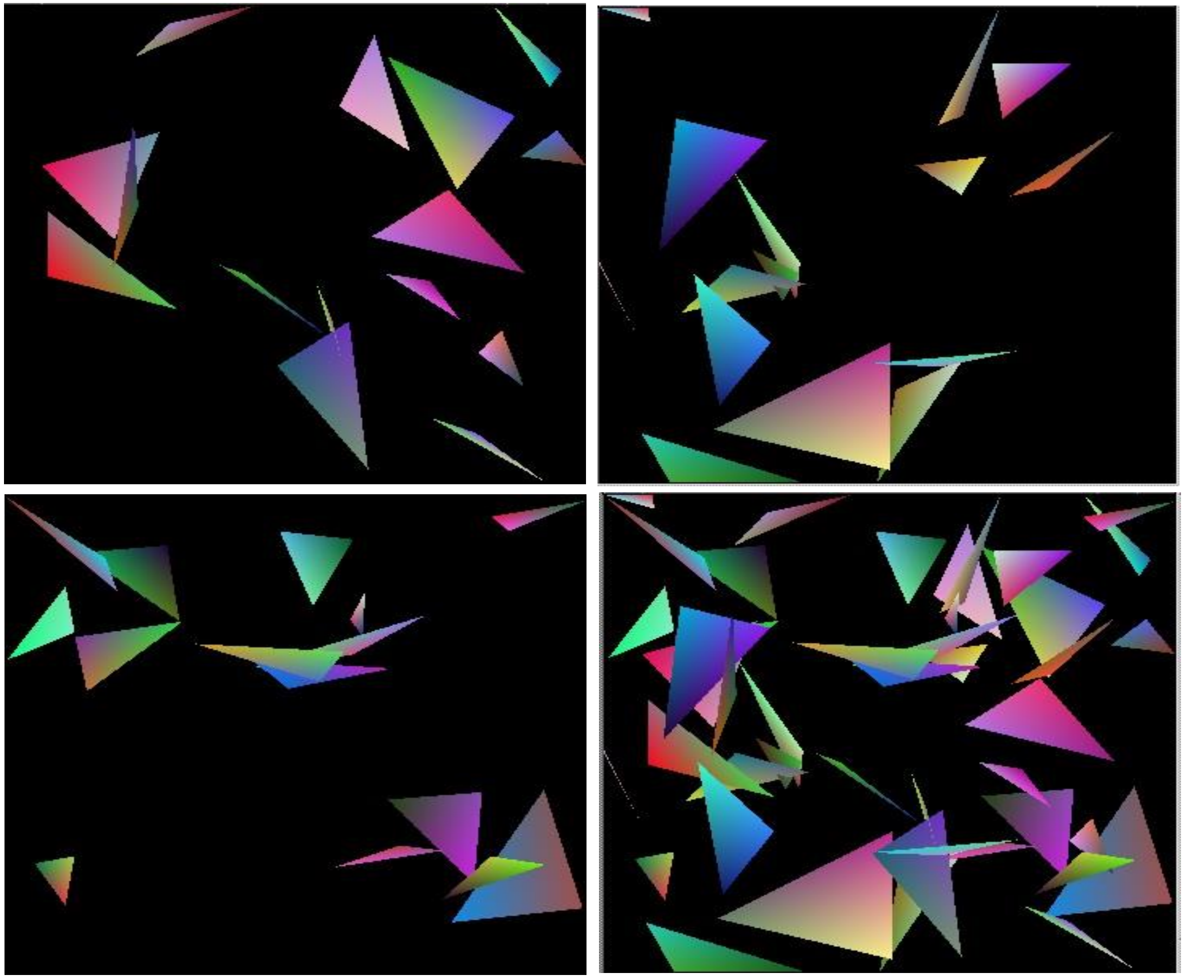


Binary Swap Compositing

- Each processor responsible for one part of display
- Pass data to right n times



Sort-Last Rendering for a Random Triangles Application



Comparison

- Sort first
  - Appealing but hard to implement
- Sort middle
  - Used in hardware pipelines
  - More difficult to implement with add-on commodity cards
- Sort last
  - Easy to implement with a compositing stage
  - High network traffic

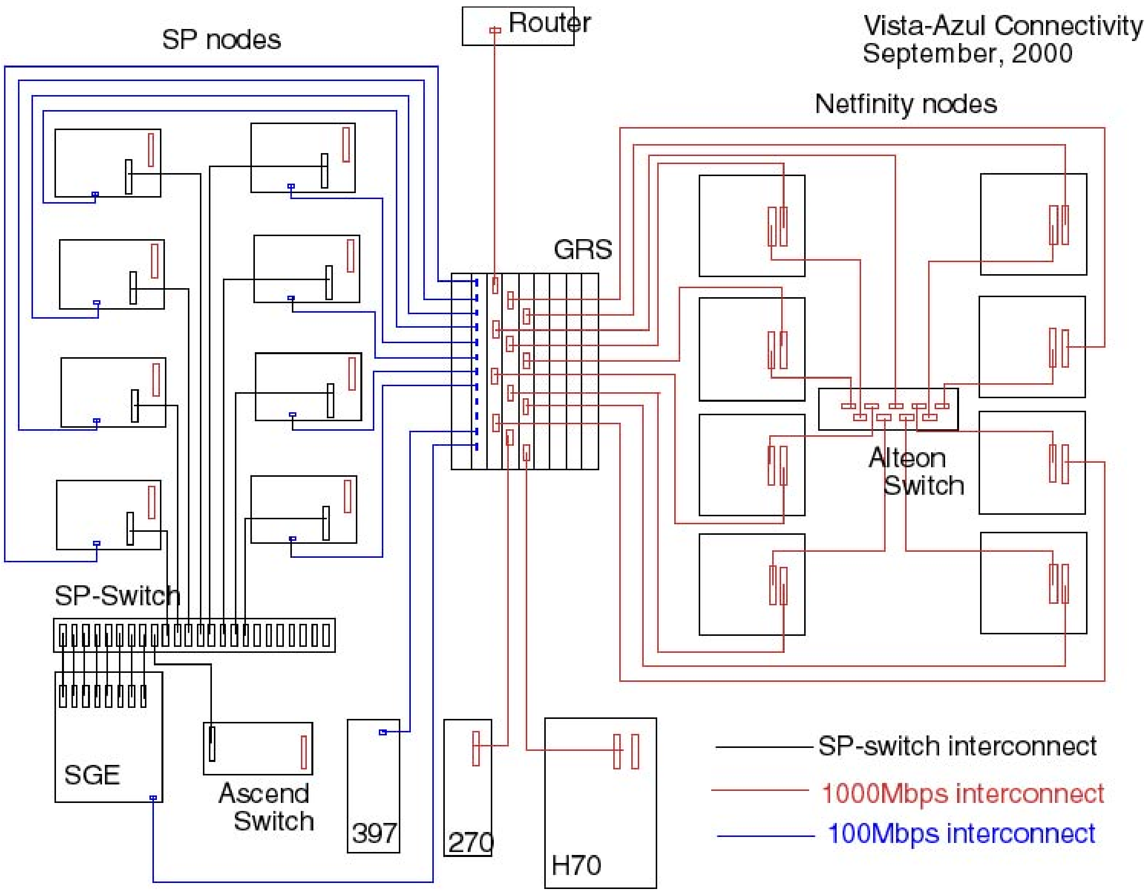
Mapping to Clusters

- Different architectures
  - Shared vs distributed memory
  - Communication overhead
  - Parallel vs distributed algorithms
- Easy to do sort last
- Must evaluate communication cost
- Standard visualization strategies are incorrect if transparency used

Vista Azul

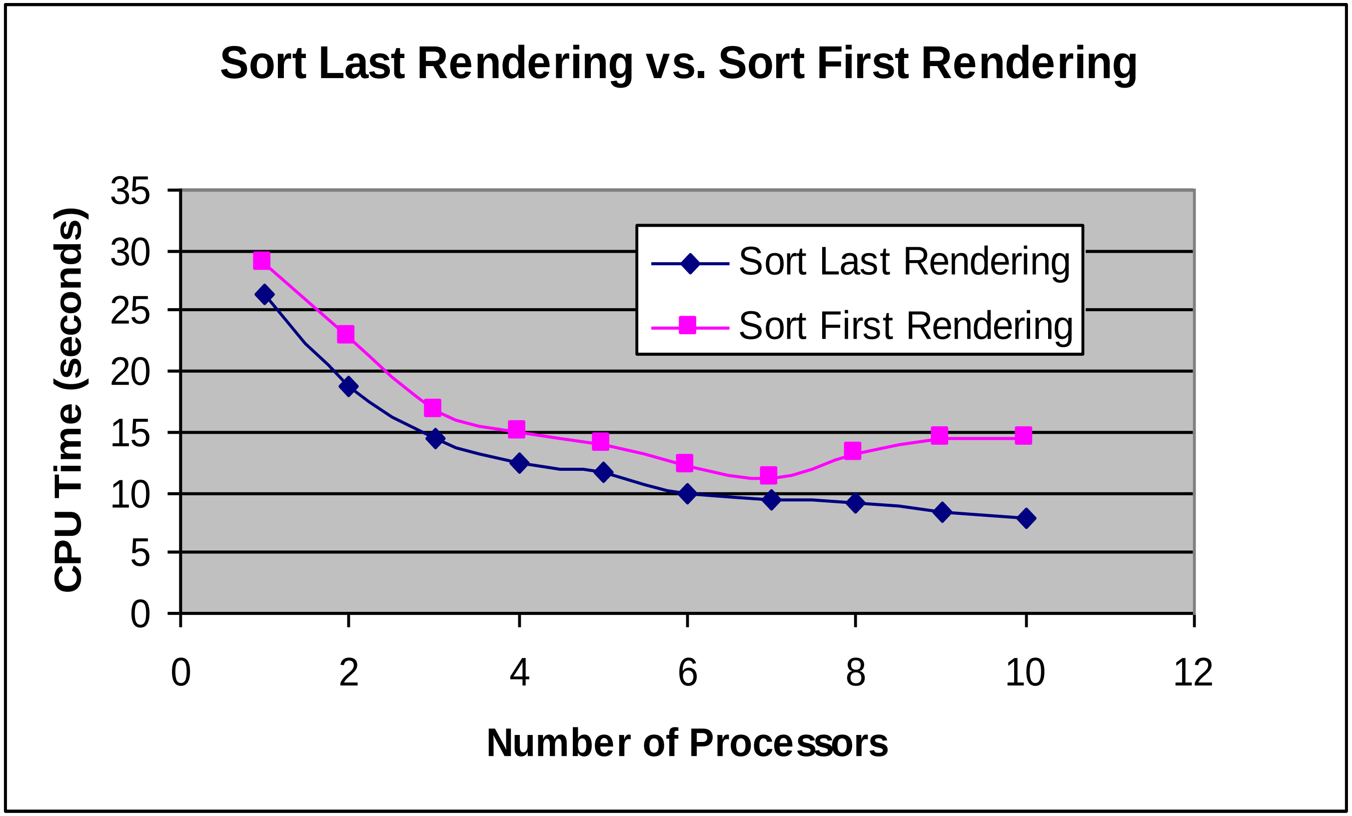
- Experimental architecture from IBM donated to AHPCC
- Half Intel nodes, half AIX nodes
- Only one (PCI) graphics card per four processors
- Contained a Scalable Graphics Engine (SGE): high speed-high resolution color buffer that is accessible by all processors

Vista Azul



Router
SP nodes
Vista-Azul Connectivity
September, 2000
Netfinity nodes
GRS
Alteon
Switch
SP-Switch
SGE
Ascend
Switch
397
270
H70
SP-switch interconnect
1000Mbps interconnect
100Mbps interconnect

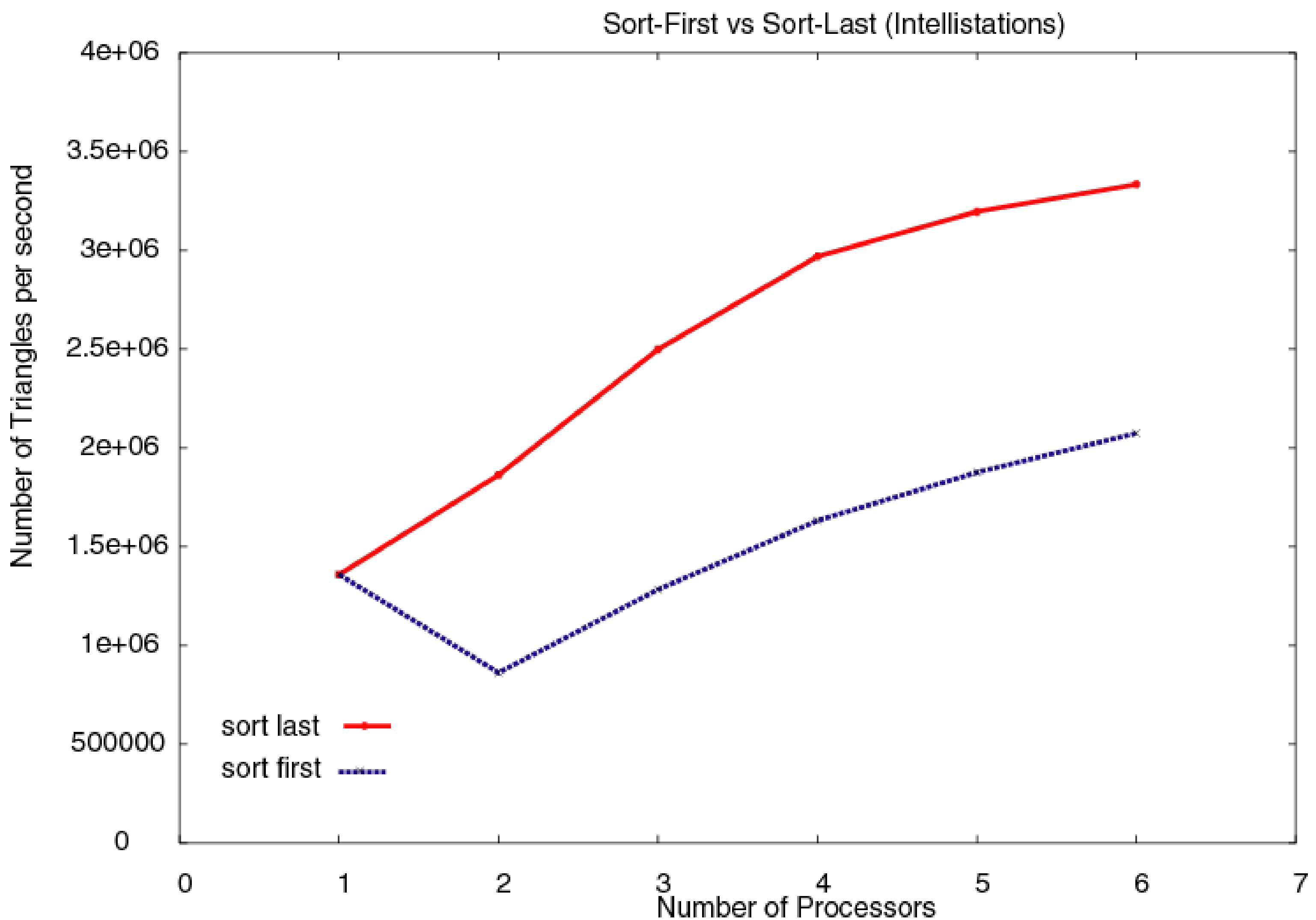
Comparison Between Sort-First and Sort-Last



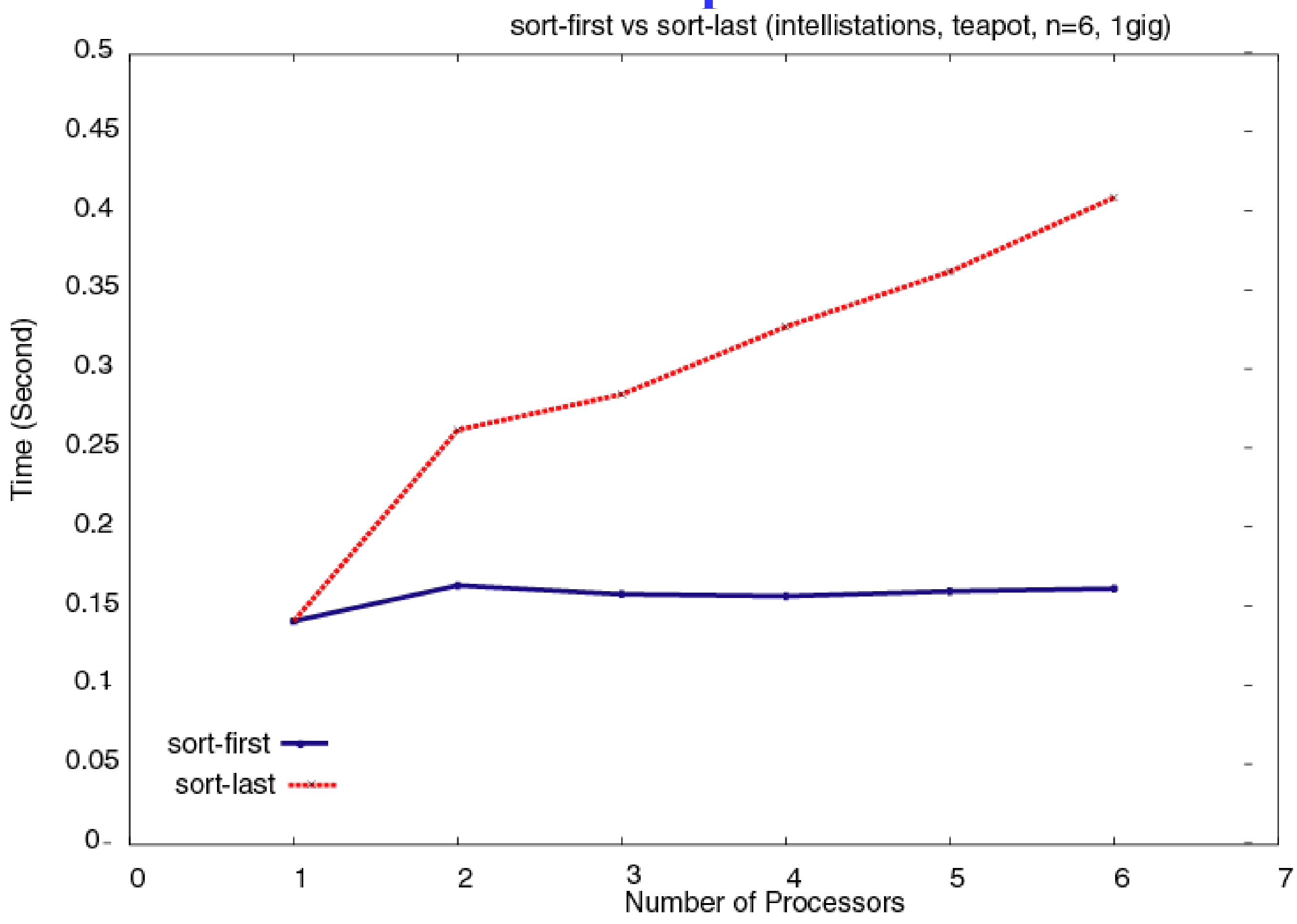
Performance on a PC Cluster

- Following experiments were done by Ye Cong on the CS cluster
  - 6 Intellestations
  - Gigabit Ethernet
  - GForce 3 graphics
- Show the effect of network

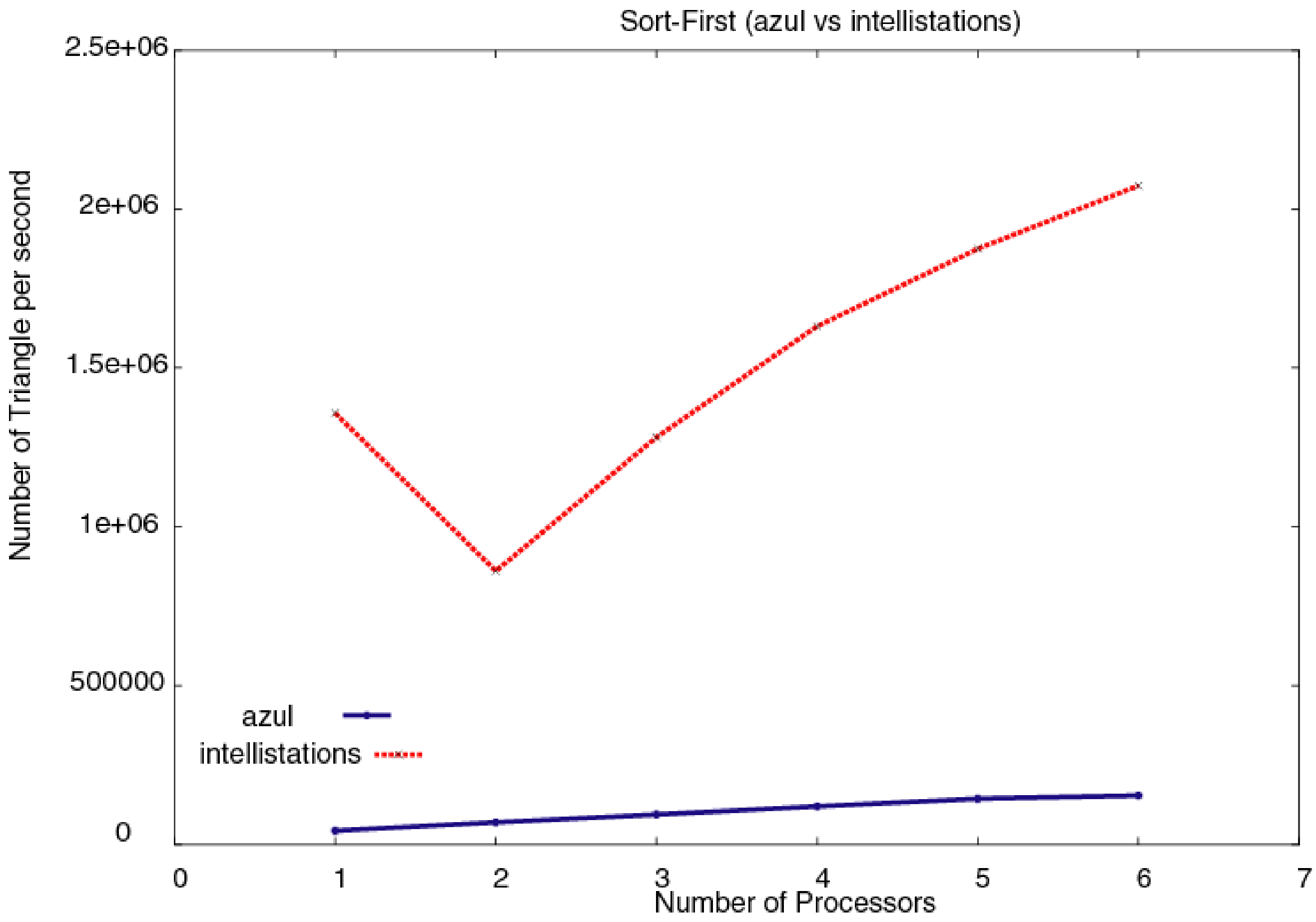
Sort-First vs Sort Last Random Triangles



Sort First vs Sort Last Teapot



Azul vs Intellistations



Software for Parallel Rendering

- Write your own sort-first sort-last
- WireGL/Chromium (Stanford)
- Embed inside package (VTK)

WireGL: A Distributed Graphics System

- A software-based parallel rendering system that unifies the rendering power of a collection of cluster nodes
- Scalability is achieved by integrating parallel applications into its sort-first parallel renders
- Each node in the cluster can be either a rendering client or a rendering server
- Clients submit OpenGL commands concurrently to servers, which render the final physical image

Chromium

- Successor toWireGL
- Allows both sort first and sort last rendering
- Implemented on CS cluster
- Most of gain in performance is bacause Chromium and WireGL can group state-changing commands separately from rendering commands

Chromium vs Sort First