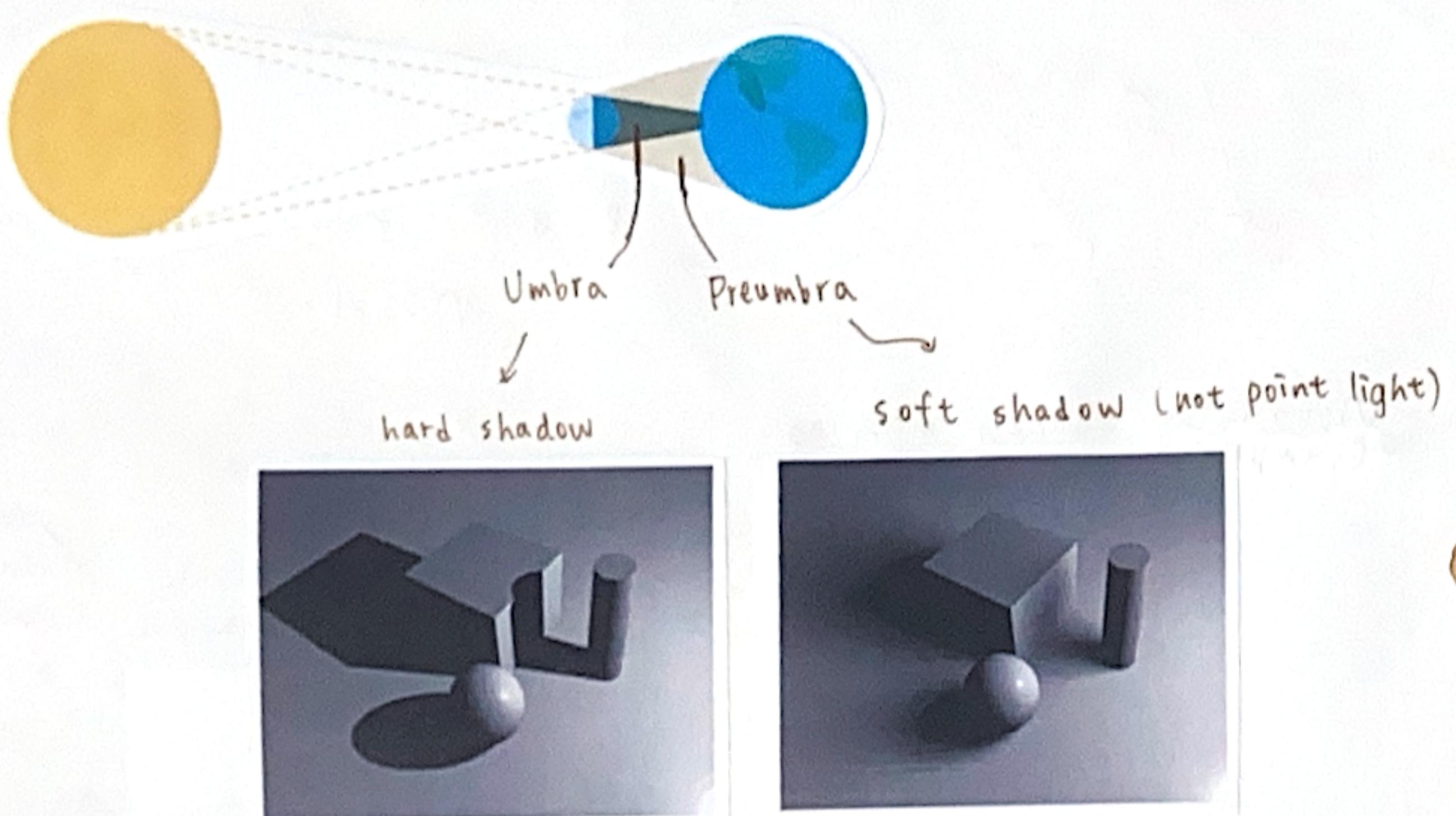


Basic shadowing technique for early animations (Toy Story, etc.) and  
EVERY 3D video game.



Problems with shadow maps:

1. Hard shadows (for point lights only)
2. Quality depends on shadow map resolution
3. Involves equality comparison of floating point depth values means issues of scale, bias, tolerance.

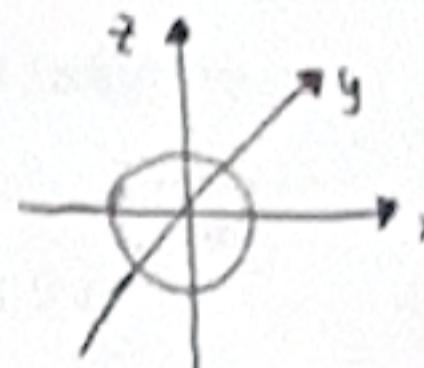


## • Geometry

There are many ways to represent geometry:

↪ Implicit : points satisfy some specified relationship

ex.  $x^2 + y^2 + z^2 = 1$  (More generally,  $f(x,y,z) = x^2 + y^2 + z^2 - 1 = 0$ )



pros: Inside / outside test easy; good for ray-to-surface intersection

$f\left(\frac{3}{4}, \frac{1}{2}, \frac{1}{4}\right) = -\frac{1}{8} < 0 \rightarrow \text{inside}$

cons: complex shapes are hard to represent

### 1. Algebraic Surface:

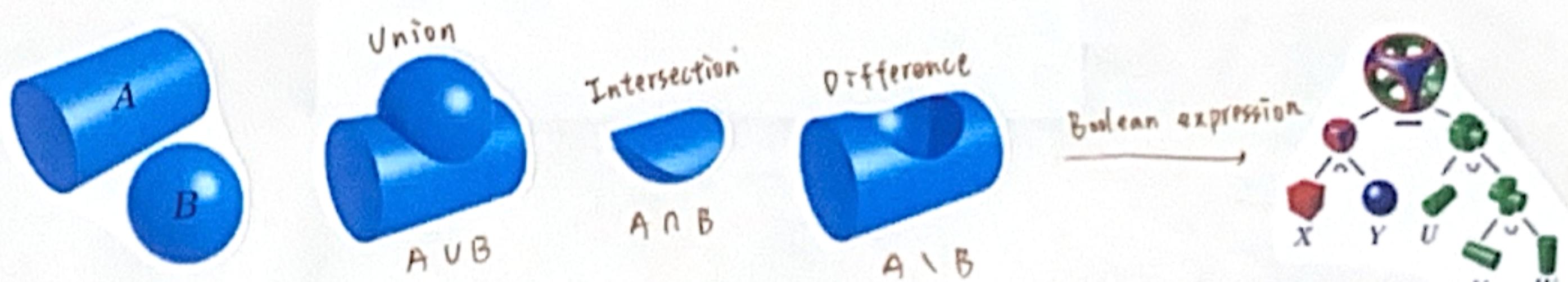
ex.  $x^2 + y^2 + z^2 = 1 \rightarrow \text{sphere}$

$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2 \rightarrow \text{甜甜圈}$

$(x^2 + \frac{9y^2}{4} + z^2 - 1)^3 = x^2 z^3 + \frac{9y^2 z^3}{80} \rightarrow \text{甜甜圈}$

### 2. Constructive Solid Geometry : (CSG)

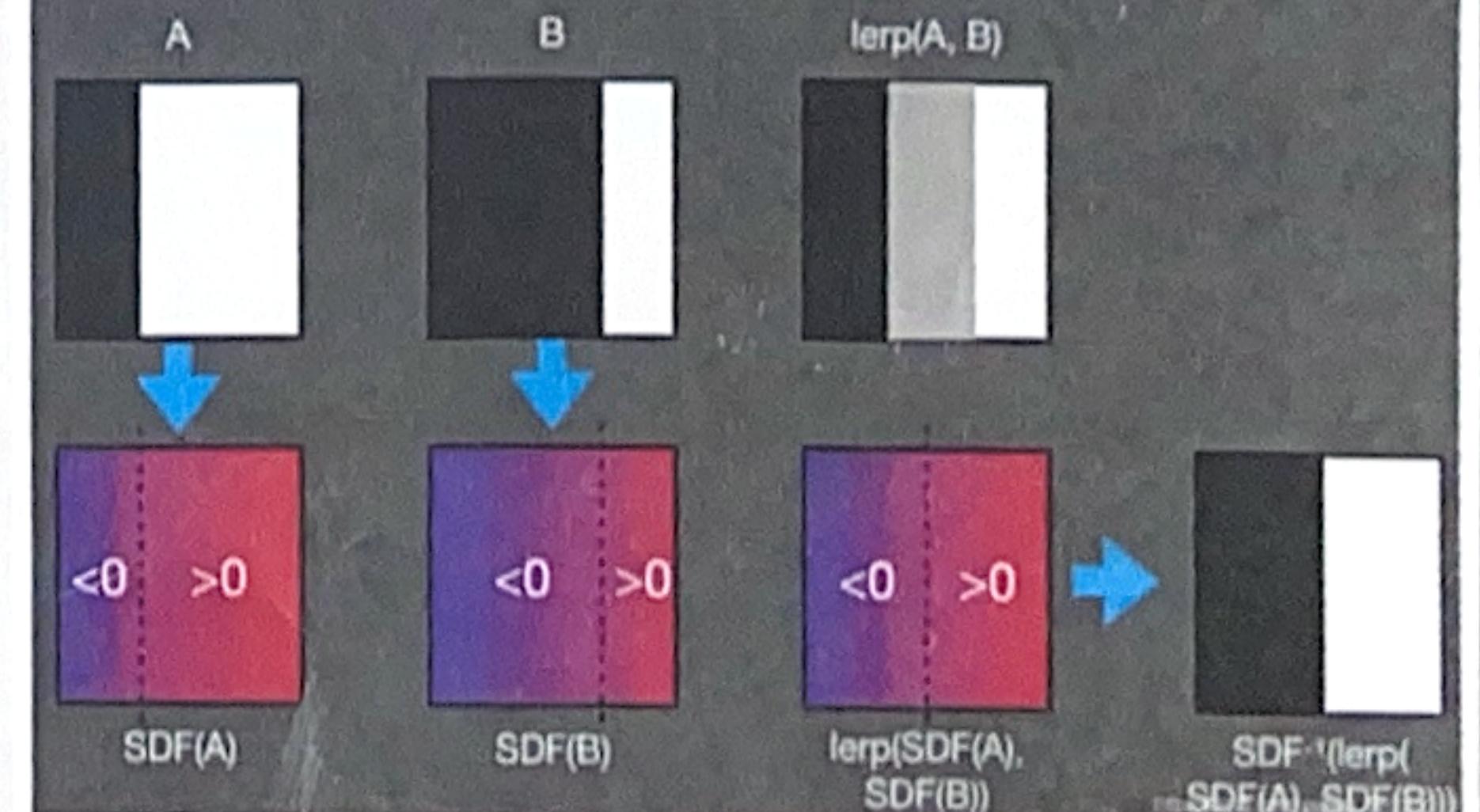
Combine implicit geometry via Boolean operations



### 3. Distance Functions:

Giving minimum distance (could be signed distance) from anywhere to objects

#### An Example: Blending (linear interp.) a moving boundary



#### 4. Level Set Methods: (like distance function)

Store a grid values approximating function

-30	-20	-10	0	10
-30	-20	-10	0	10
-30	-20	-10	0	10
-30	-20	-10	0	10

surface is found when interpolated  
 $f(x) = 0$  values equal zero  
 → provide much more explicit control over shape (such as texture)

Also use at medical data: CT, MRI...

physical simulation: find air-liquid boundary

#### 5. Fractals:

Exhibit self-similarity, detail at all scales.

"Language" for describing natural phenomena

Hard to control shape!

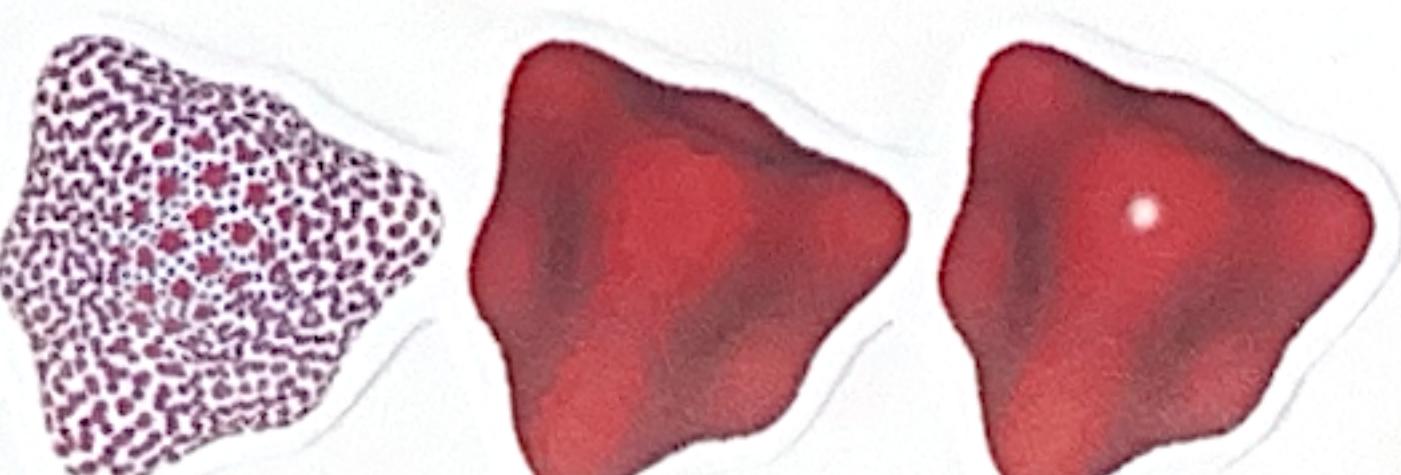


顯示

\* Explicit:

##### 1. Point cloud:

- Easiest representation: list of points  $(x, y, z)$
- Easily represent any kind of geometry
- Useful for LARGE datasets ( $\gg 1$  point/pixel)
- Often converted into polygon mesh
- Difficult to draw in undersampled regions



L72

#### 2. Polygon mesh:

- Store vertices & polygons (often triangles or quads)
- Easier to do processing / simulation, adaptive sampling
- More complicated data structures
- Perhaps most common representation in graphics

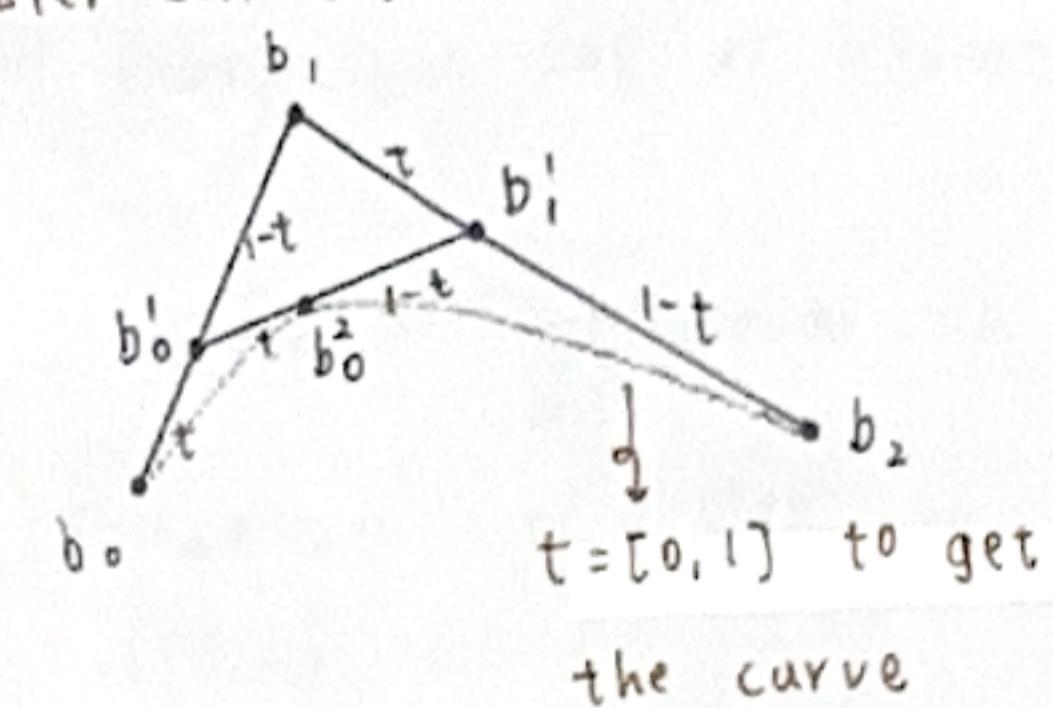


ex. Wavefront Object File (.obj) Format

- Commonly used in Graphics research
- Just a text file that specifies vertices, normals, texture coordinates and their connectivities

#### 3. Curves:

##### \* Bézier curve:



$b_0, b_1, b_2$

↓

$$b'_0(t) = (1-t)b_0 + tb_1$$

↓

$$b''_0(t) = (1-t)^2 b_0 + 2t(1-t) b_1 + t^2 b_2$$

$t = [0, 1]$  to get the curve

$$\Rightarrow b''_0(t) = (1-t)^2 b_0 + 2t(1-t) b_1 + t^2 b_2$$

$\Rightarrow$  Bernstein form of a Bézier curve of order  $n$ :

$$b^n(t) = b''_0(t) = \sum_{j=0}^n b_j B_j^n(t), \quad B_j^n(t) = \binom{n}{j} t^j (1-t)^{n-j}$$

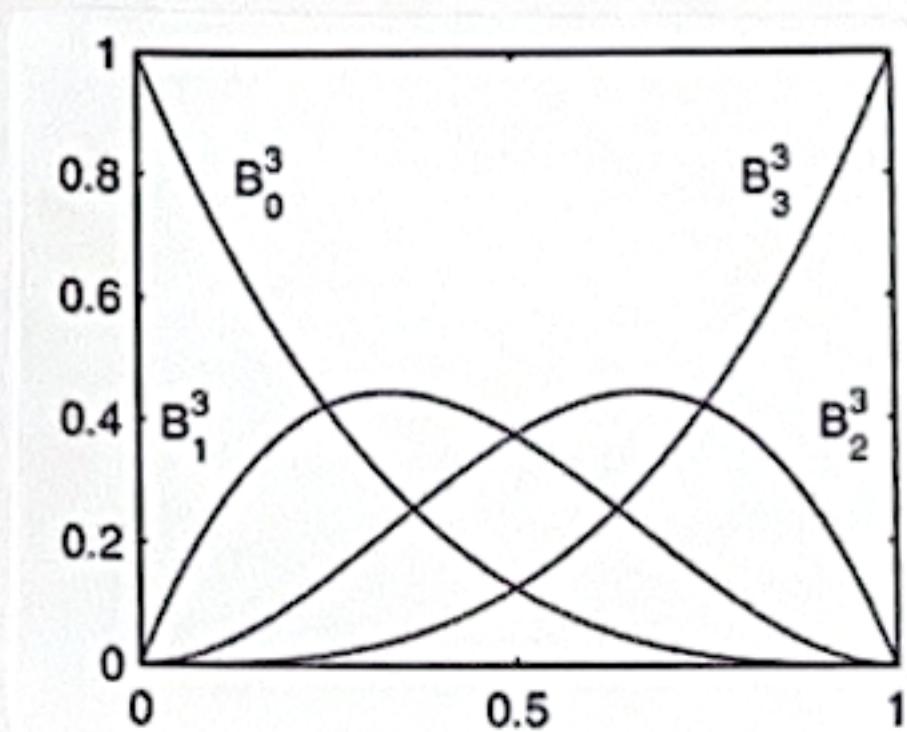
Bézier curve order  $n$

(vector polynomial of degree  $n$ )

Bernstein polynomial

(scalar polynomial of degree  $n$ )

Bézier control point (vector in  $R^n$ )



properties of Bézier curves:

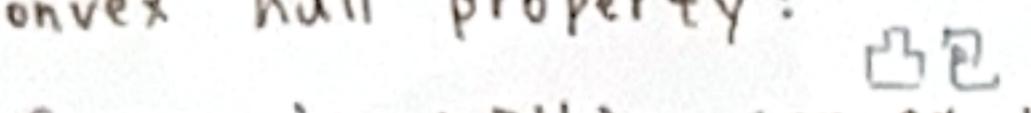
1. Interpolates endpoints: ex. for cubic Bézier:  $\begin{cases} b(0) = b_0 \\ b(1) = b_3 \end{cases}$

2. Tangent to end segments: ex. for cubic Bézier:  $\begin{cases} b'(0) = 3(b_1 - b_0) \\ b'(1) = 3(b_3 - b_2) \end{cases}$

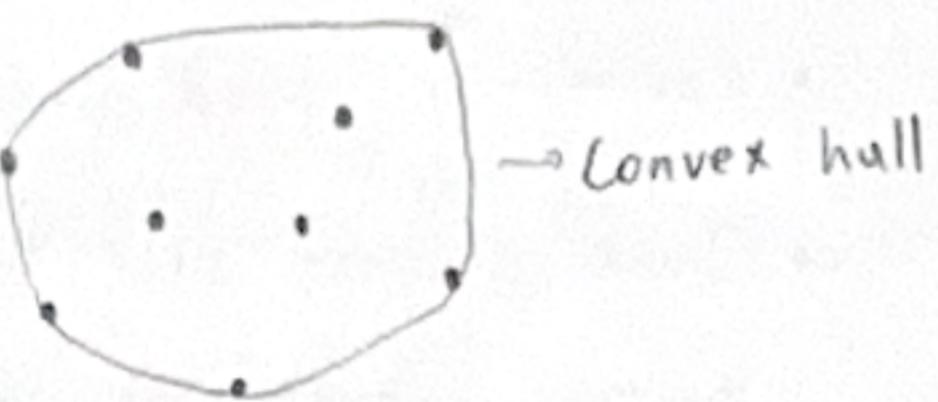
3. Affine transformation property:

Transform curve by transforming control point

4. Convex hull property:



Curve is within convex hull of control points



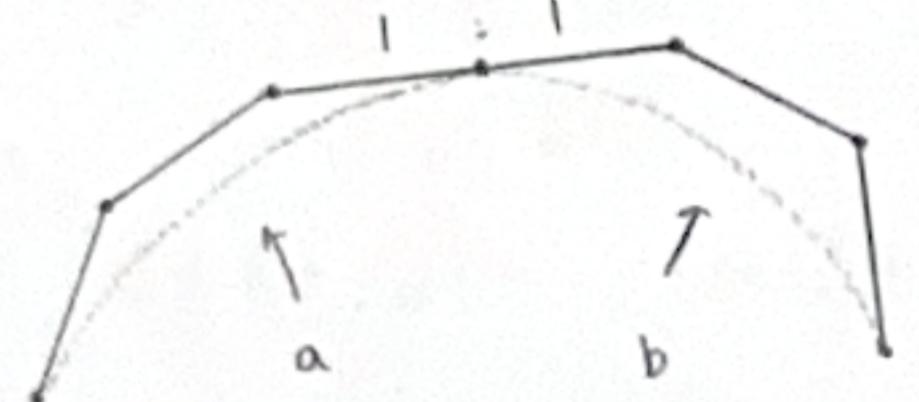
⇒ piecewise Bézier curves.

Chain many low-order Bézier curves to get high order Bézier curve.

$C^0$  continuity:  $a_n = b_0$

$C^1$  continuity:  $a_n = b_0 = \frac{1}{2}(a_{n-1} + b_1)$

↓  
可視高階連續性連續



\* Other types of splines:

Spline: a continuous curve constructed so as to pass through a given set of points and have a certain number of continuous derivatives.

B-splines (Basis splines):  
 • Require more information than Bézier curves.  
 • Satisfy all important properties that Bézier curves have.

NURBS (non-uniform rational B-splines)

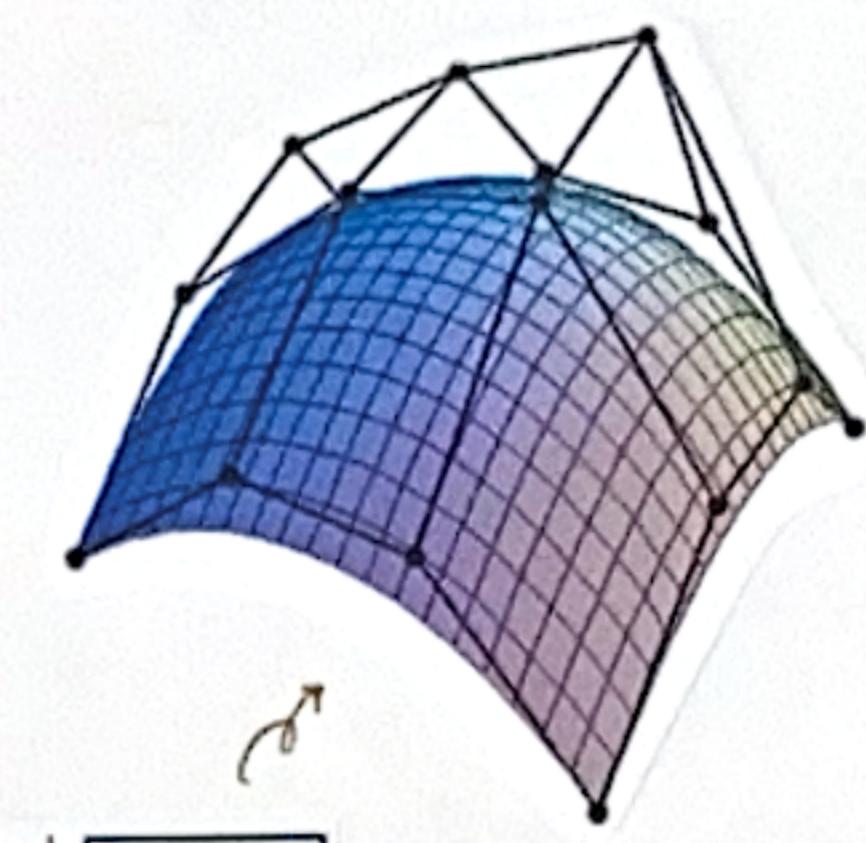
G3

4. Surface:

\* Bézier surface:

For bi-cubic Bézier surface patch,

Input:  $4 \times 4$  control points



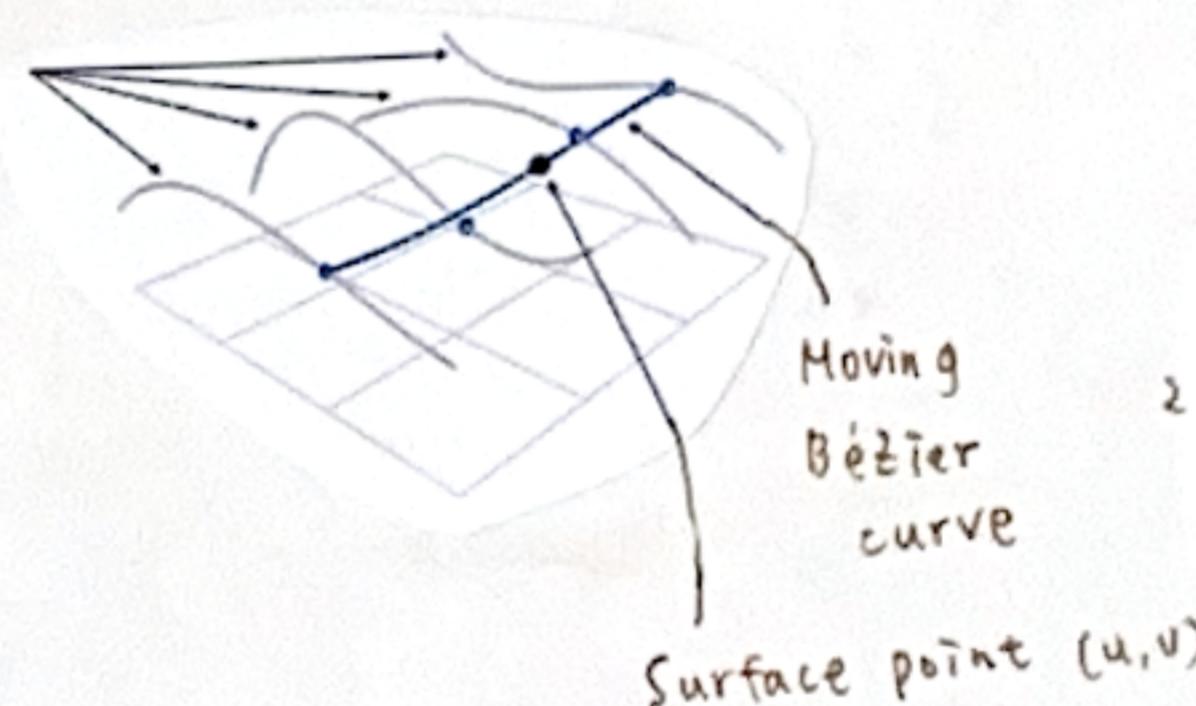
Output: 2D surface parameterized by  $(u, v)$  in  $[0, 1]^2$

Method: Separable 1D de Casteljau Algorithm

Goal: Evaluate surface position corresponding to  $(u, v)$

$(u, v)$ -separable application of de Casteljau algorithm:

1. Use de Casteljau to evaluate point  $u$  on each of the 4 Bézier curves in  $u$ . This gives 4 control points for the "moving" Bézier curve.



2. Use 1D de Casteljau to evaluate point  $v$  on the "moving" curve.

Geometry processing: mesh operations:



Increase resolution (upsampling)

Decrease resolution (downsampling)

Try to preserve shape/appearance

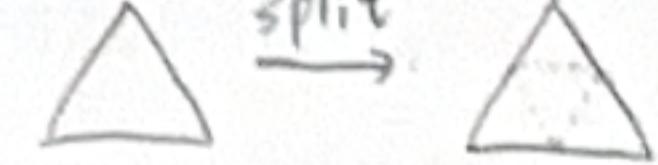
Same #triangle

Modify sample distribution to improve quality.

## 1. Subdivision:

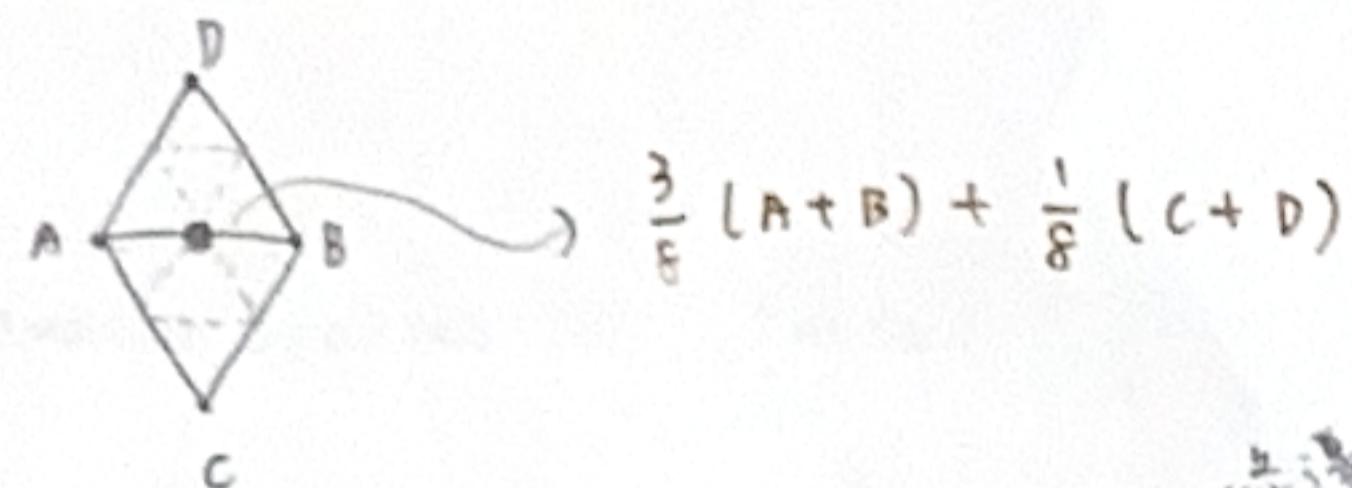
- \* Loop subdivision: Common subdivision rule for triangle meshes

$\lambda \frac{3}{8}$

step 1. Create more triangles (vertices): 

step 2. Tune their positions:

Update the new vertex:

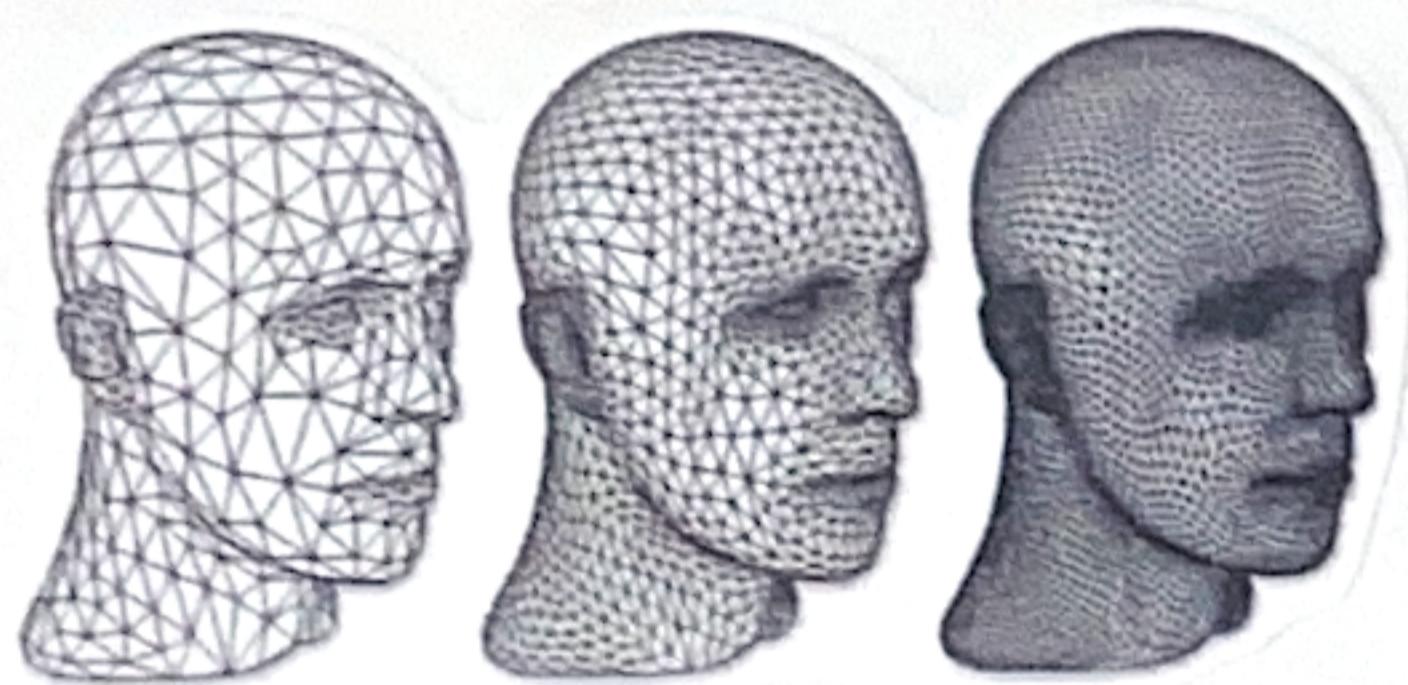


Update the old vertex:

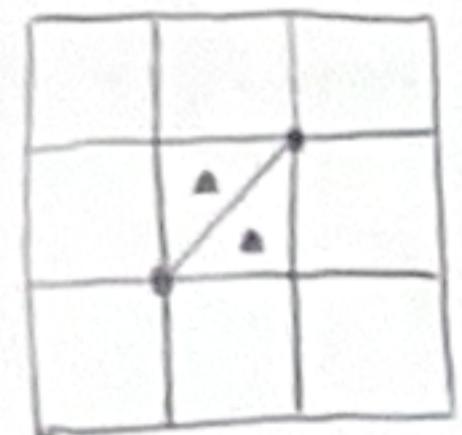
$$\text{new position} = (1-n \cdot u) \cdot \text{original-position} + u \cdot \text{neighbor-position-sum}$$

$$u = \begin{cases} \frac{3}{16}, & n=3 \\ \frac{3}{8n}, & \text{otherwise} \end{cases}$$

Result:



- \* Catmull-Clark subdivision: for general meshes



Add vertex in each face  
Add midpoint on each edge  
Connect all the vertices

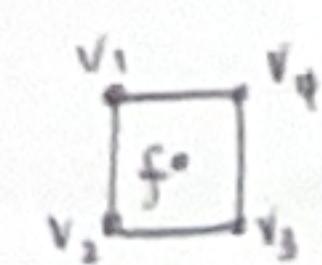


After one subdivision:

$N_f + N_e$  extraordinary points  
 $\downarrow$   
 $n=3$      $n=5$

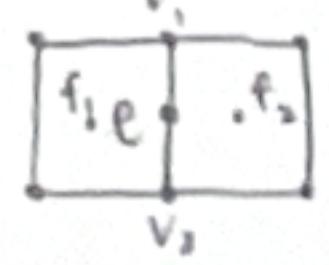
No more non-square face.

Update face point



$$f = \frac{1}{4}(v_1 + v_2 + v_3 + v_4)$$

Update edge point



$$e = \frac{1}{4}(v_1 + v_2 + f_1 + f_2)$$

Update vertex point



$$v = \frac{1}{16}(f_1 + f_2 + f_3 + f_4 + 2(m_1 + m_2 + m_3 + m_4) + 4p)$$

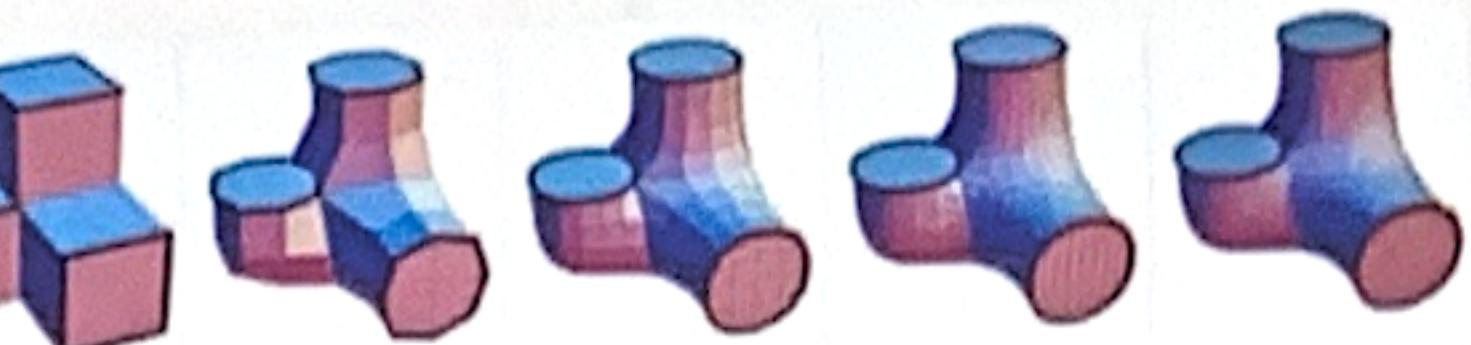
old vertex point

64

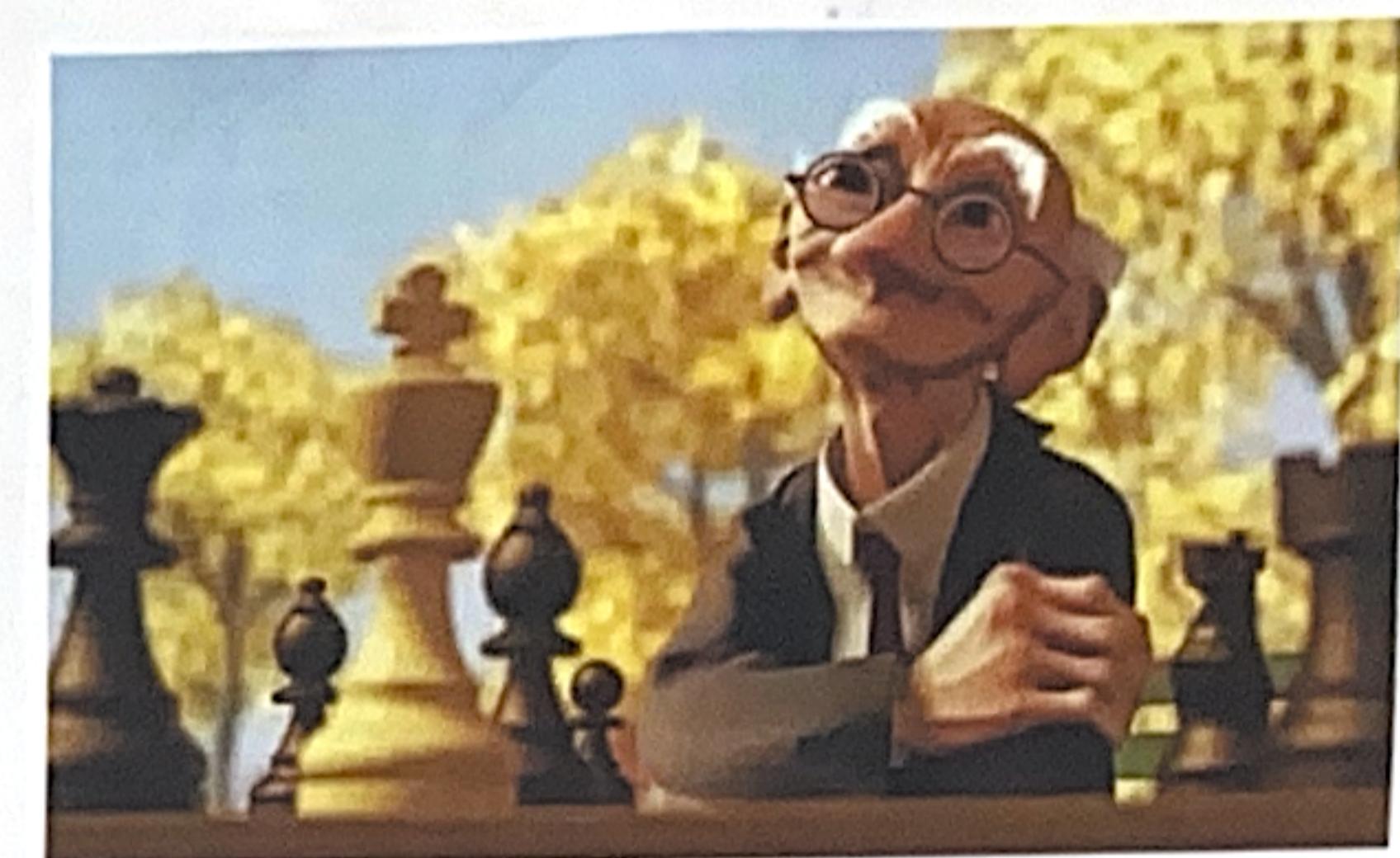
To sum up subdivision:



Loop with sharp creases



Catmull-Clark with sharp creases



Subdivision in action (Pixar's "A Bug's Life")

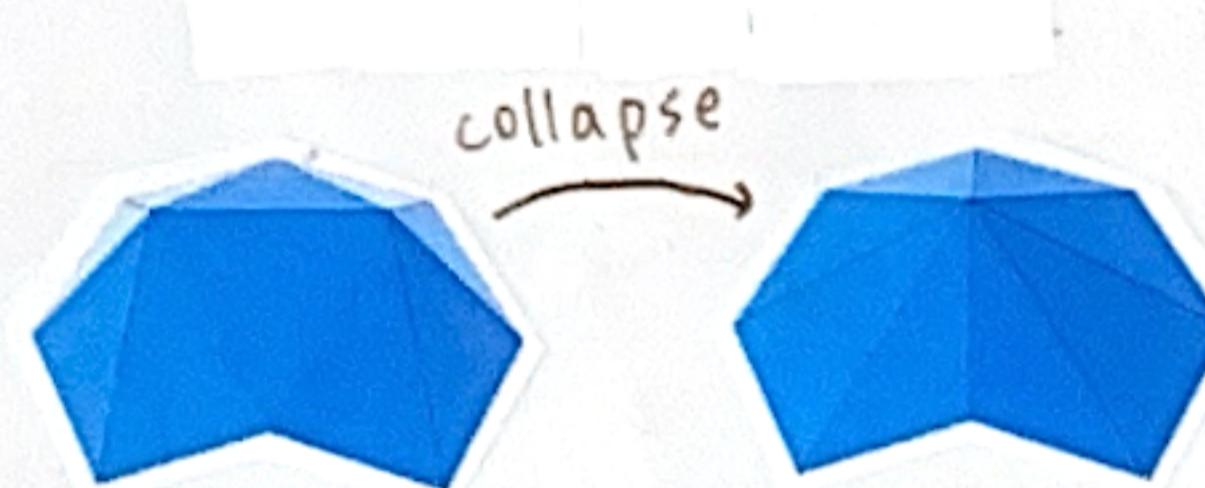
## 2. Simplification:

Goal: reduce number of mesh elements while maintaining the overall shape



Use edge collapsing!!!

→ Method: Quadric error metrics = 误差度量



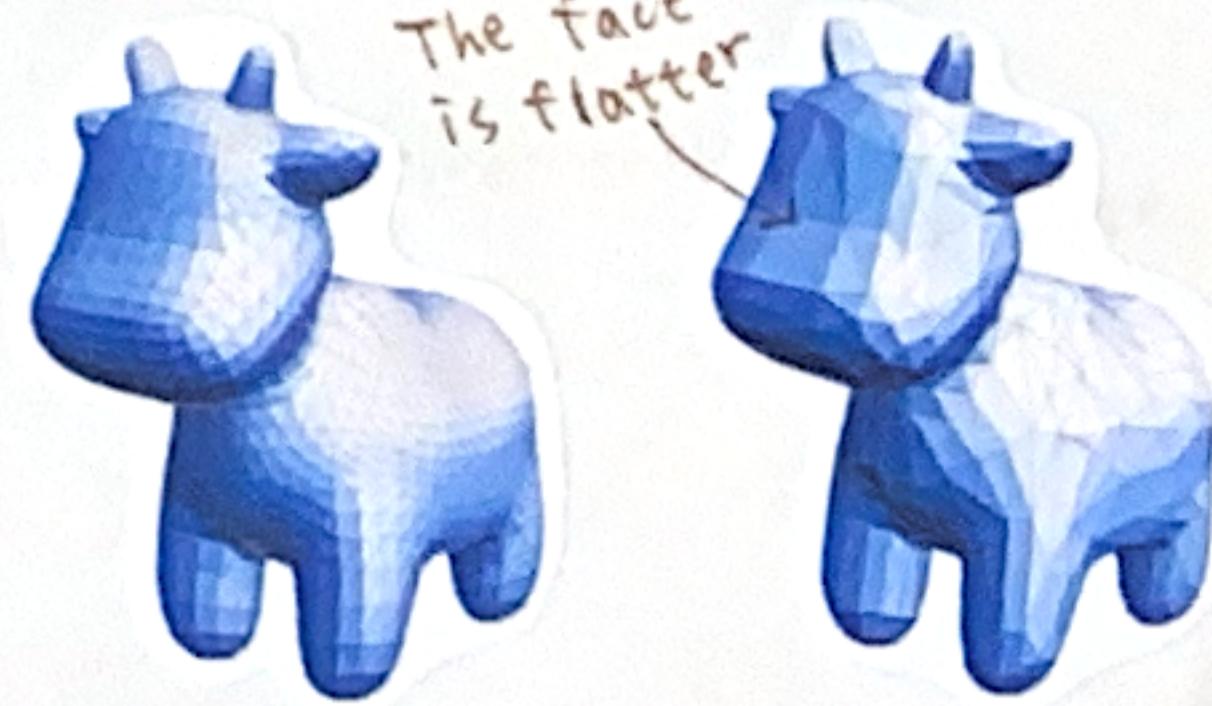
The new vertex should minimize its sum of square distance to previously related triangle planes!

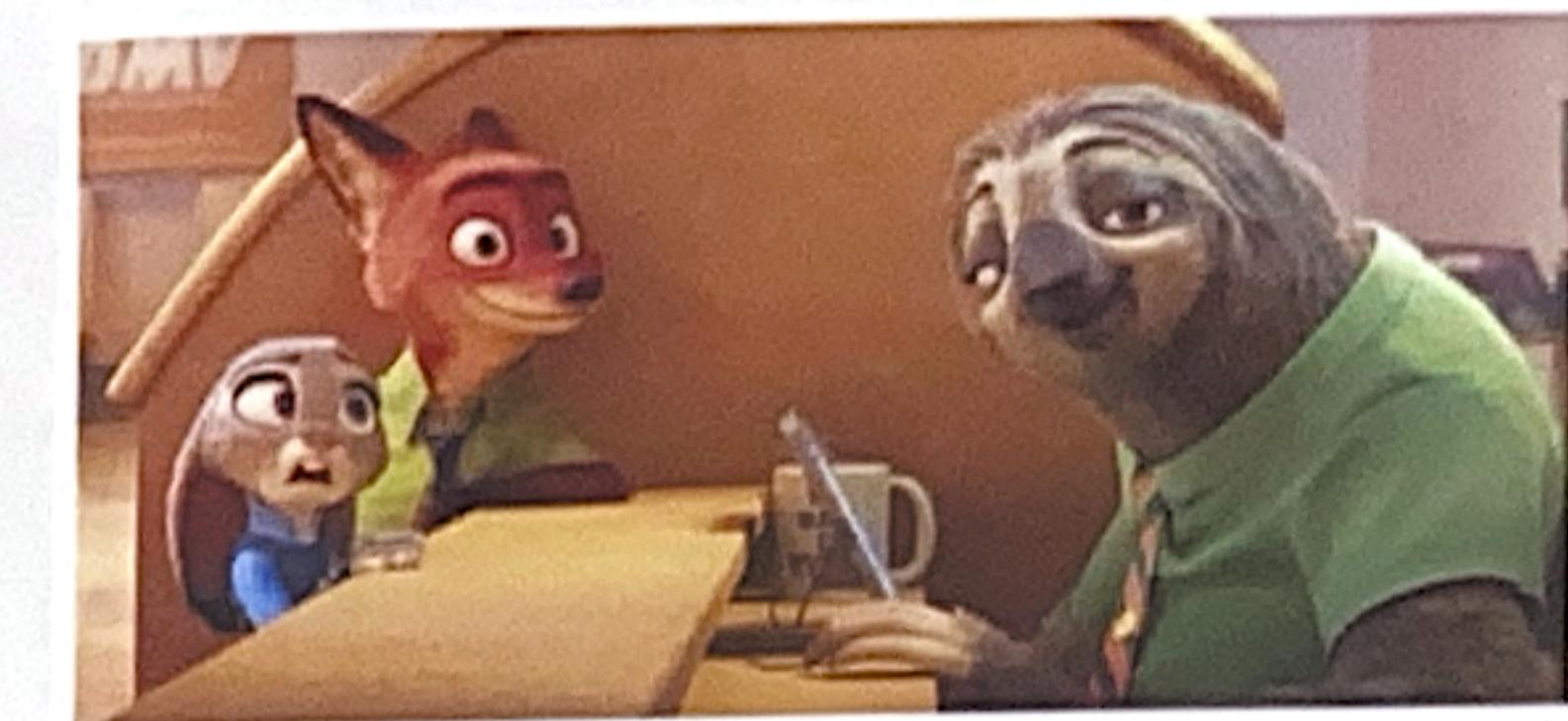
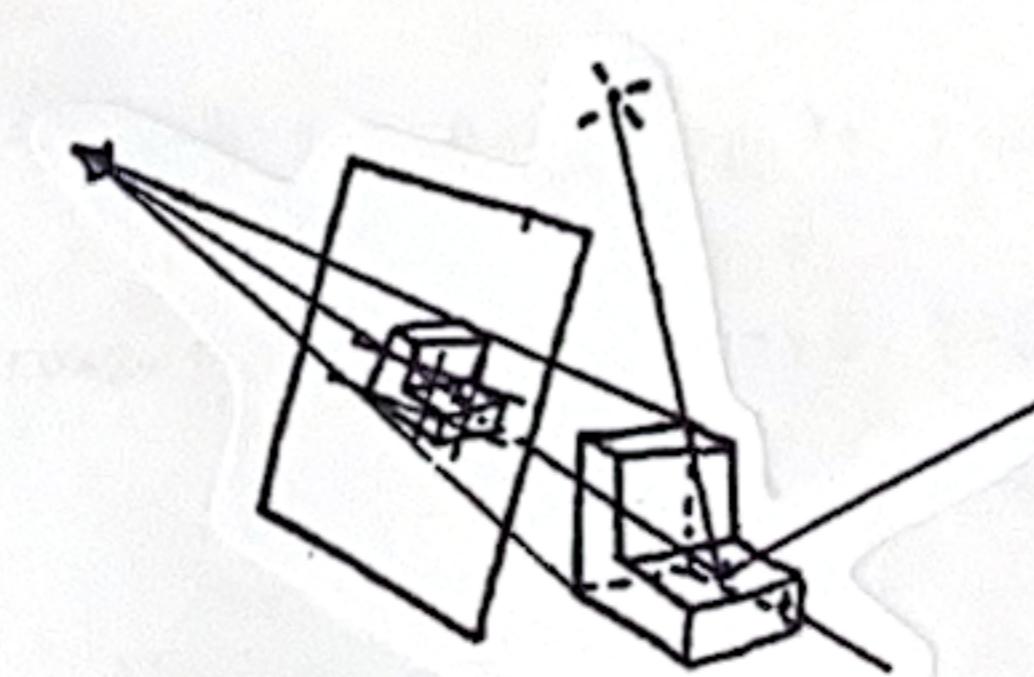
→ Iteratively collapse edges:

Step 1. which edges? Assign score with quadric error metric by approximating distance to surface as sum of distances to planes containing triangles.

→ use stack in data structure  
Step 2. iteratively collapse edge with smallest score.

Step 3: greedy algorithm ... → great result!

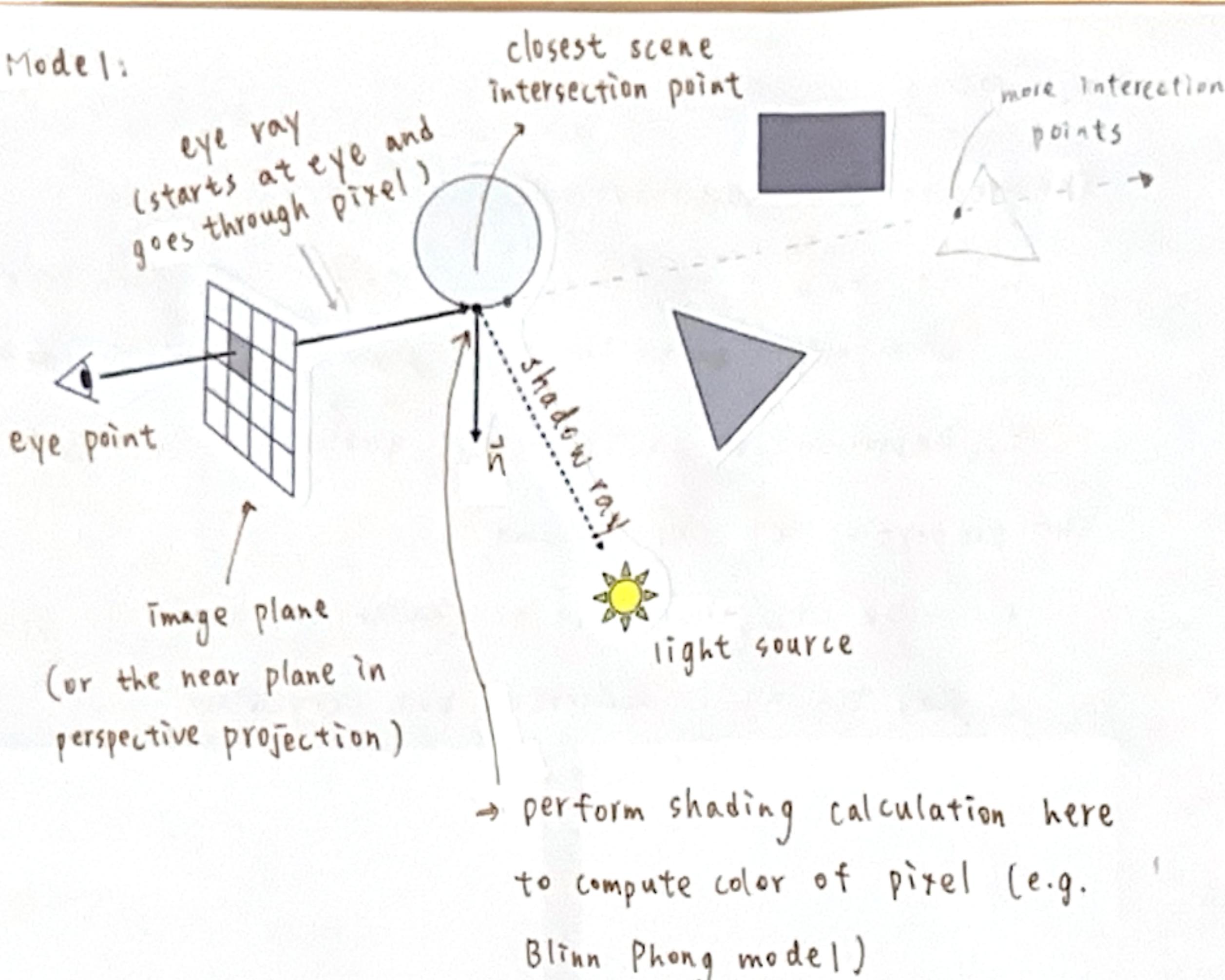


- RT<sub>1</sub>
- Ray Tracing:
- The properties of rasterization:
1. It couldn't handle global effects well, such like (soft shadow) or especially when the light bounces more than once
  2. Rasterization is fast, but quality is relatively low.
- The properties of ray tracing:
1. ~10K CPU core hours to render one frame in production.
  2. Ray tracing is accurate, but very slow.
- 

- \* Rasterization: real-time      \* ray tracing: offline
- Basic Ray-Tracing Algorithm:
- \* About light rays:
    1. Light travels in straight lines
    2. Light rays do not "collide" with each other if they cross.
    3. Light rays travel from the light sources to the eye  
→ but the physics is invariant under path reversal - reciprocity.
- "And if you gaze along into an abyss, the abyss also gazes into you" — Friedrich Wilhelm Nietzsche
- \* Ray casting: (Generating eye rays)
- 

step.1. Generate an image by casting one ray per pixel

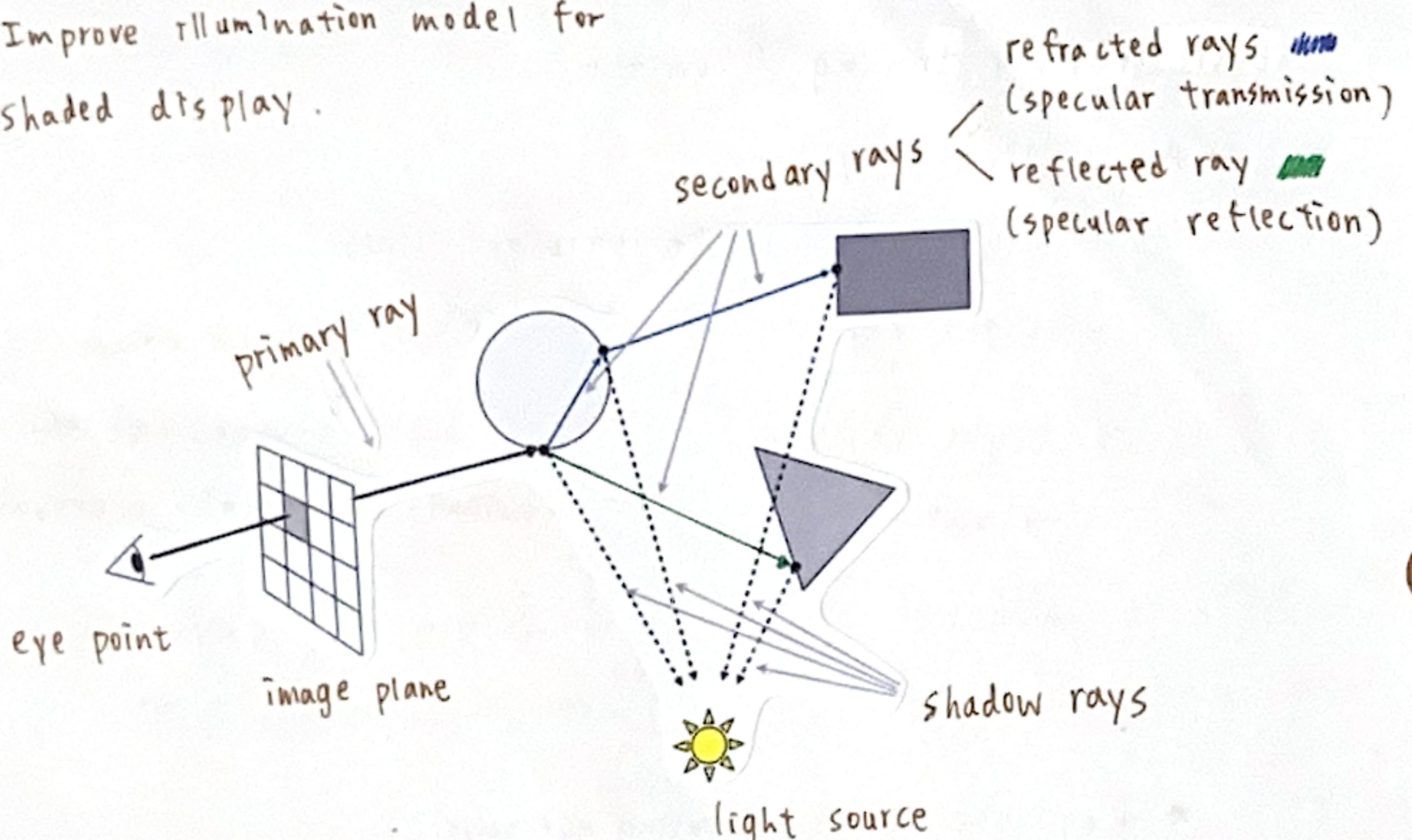
Step 2. Check for shadows by sending a ray to the light

Pinhole Camera Model:

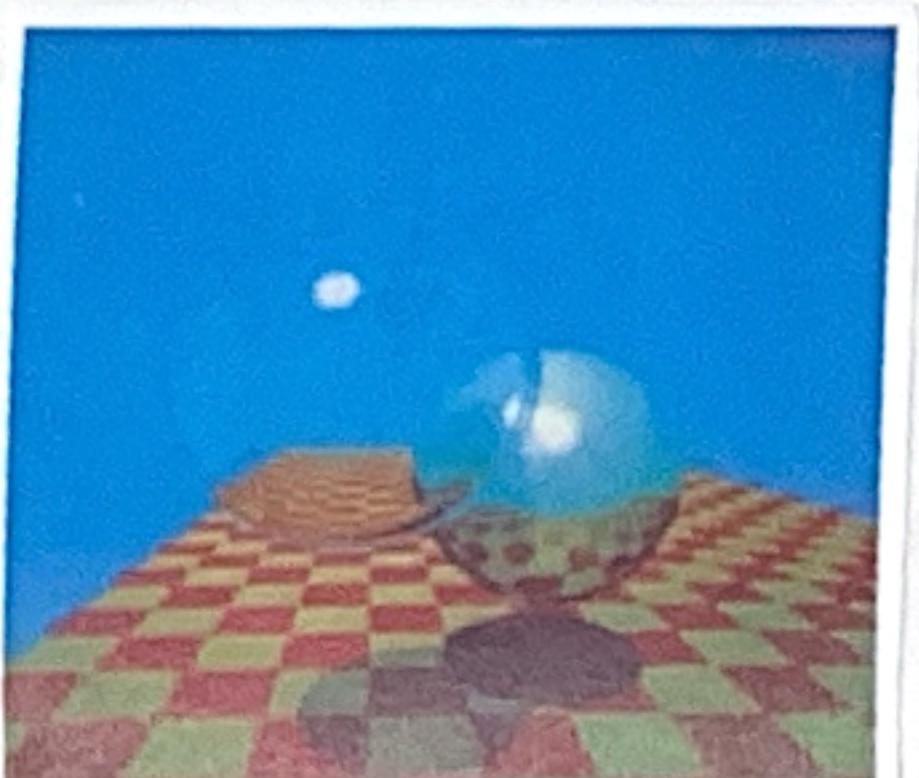


\* Recursive (Whitted-Style) ray tracing:

→ Improve illumination model for shaded display.



Result:



Sphere and checkerboard,  
T. Whitted, 1979

The time of generating one frame:

VAX 11/780 (1979): 74 minutes

PC (2006): 6 seconds

GPU (2012):  $\frac{1}{70}$  seconds

△ Ray-Surface Intersection:

$$\text{Ray equation: } \vec{r}(t) = \vec{o} + t\vec{d}, \quad 0 \leq t < \infty$$

↑ time  
point along ray  
↓ origin (normalized) direction

$$\text{Sphere equation: } \vec{p} : (\vec{p} - \vec{c})^2 - R^2 = 0$$

→ The intersection  $P$  must satisfy both ray equation and sphere equation.

$$\text{General implicit surface: } \vec{p} : f(\vec{p}) = 0$$

$$\rightarrow \text{solve } f(\vec{o} + t\vec{d}) = (\vec{o} + t\vec{d} - \vec{c})^2 - R^2 = 0$$

and find  $t$  is real and positive.

Ray intersection with triangle mesh

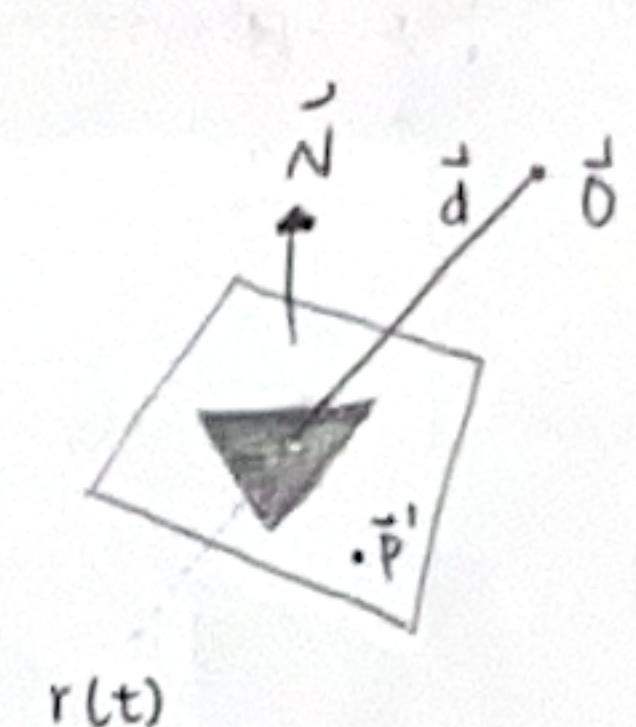
Why? Rendering = visibility, shadows, lighting ....  
Geometry: inside/outside test

Step 1. Ray-plane intersection

Step 2. Test if hit point is inside triangle

$$\text{Plane equation: } \vec{p} : (\vec{p} - \vec{p}') \cdot \vec{N} = 0$$

↓ all point on the plane      ↓ one point on the plane      ↑ normal vector



$$\rightarrow \text{solve } f(\vec{p}) = f(\vec{o} + t\vec{d}) = (\vec{o} + t\vec{d} - \vec{p}') \cdot \vec{N} = 0$$

$$\rightarrow t = \frac{(\vec{p}' - \vec{o}) \cdot \vec{N}}{\vec{d} \cdot \vec{N}}, \text{ and check } 0 \leq t < \infty$$

△ Möller Trumbore Algorithm:

A faster approach, giving barycentric coordinate directly.

$$\vec{o} + t\vec{d} = (1 - b_1 - b_2)\vec{p}_0 + b_1\vec{p}_1 + b_2\vec{p}_2$$

重心座標  
ray

$$\rightarrow \text{solve } \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{s}_1 \cdot \vec{E}_1} \begin{bmatrix} \vec{s}_2 \cdot \vec{E}_2 \\ \vec{s}_1 \cdot \vec{S} \\ \vec{s}_2 \cdot \vec{D} \end{bmatrix}, \text{ where } \begin{cases} \vec{E}_1 = \vec{p}_1 - \vec{p}_0 \\ \vec{E}_2 = \vec{p}_2 - \vec{p}_0 \\ \vec{S} = \vec{o} - \vec{p}_0 \\ \vec{D} = \vec{d} \times \vec{E}_2 \\ \vec{s}_1 = \vec{S} \times \vec{E}_1 \\ \vec{s}_2 = \vec{S} \times \vec{E}_2 \end{cases}$$

→ check  $\begin{cases} 0 \leq t < \infty \\ 0 \leq b_1 < 1 \\ 0 \leq b_2 < 1 \\ 0 \leq 1 - b_1 - b_2 < 1 \end{cases}$

## Accelerating Ray-Surface Intersection:

The problem of simple-scene intersection:

Exhaustively test ray-intersection with every triangle

→ Naive algorithm = # pixels × # triangles (x # bounces)

→ Very SLOW!!

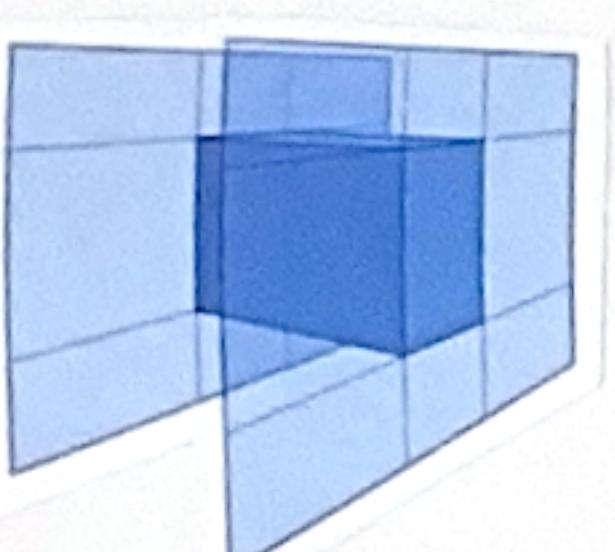
⇒ For generality, we use the term objects instead of triangles.

\* Bounding Volumes:

Quick way to avoid intersections: bound complex object with a simple volume.



⇒ Ray-Intersection with Box



It is the intersection of 3 pairs of slabs.

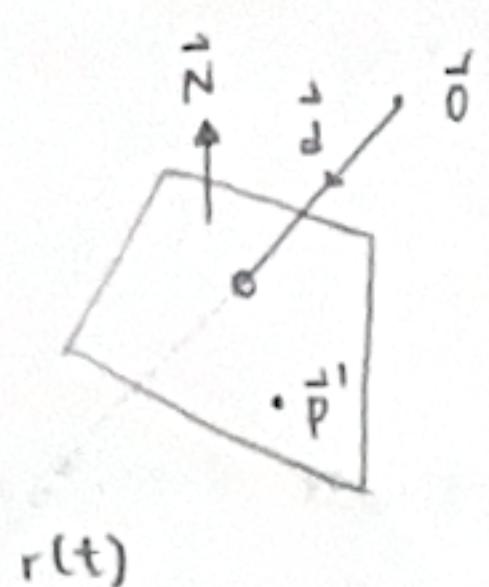
Specifically: We often use an Axis-Aligned

Bounding Box (AABB), (軸對齊包圍盒)

Any side of the Bounding Box is along either x, y or z axis.

\* Why Axis-Aligned?

General

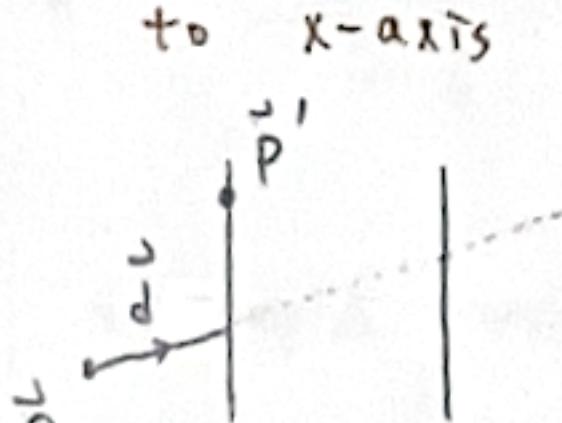


$$t = \frac{(\vec{P}' - \vec{O}) \cdot \vec{N}}{\vec{d} \cdot \vec{N}}$$

3 subtractions, 6 multiplies,

1 division

Slabs perpendicular to x-axis



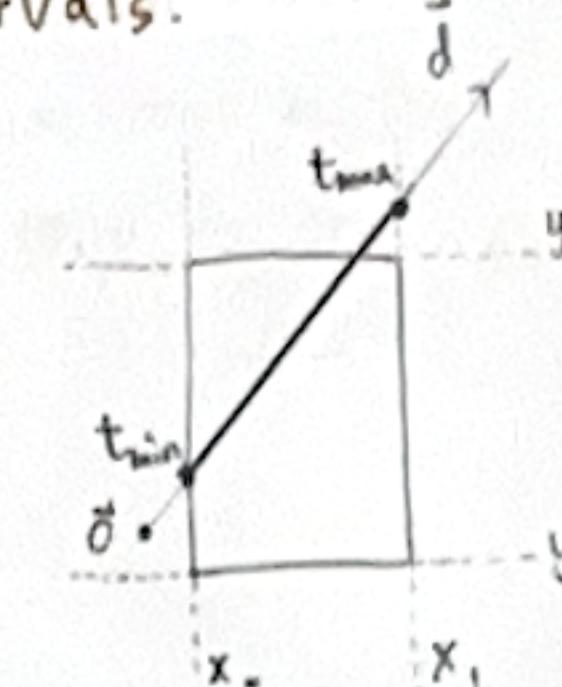
$$t = \frac{P'_x - O_x}{d_x}$$

1 subtraction, 1 division

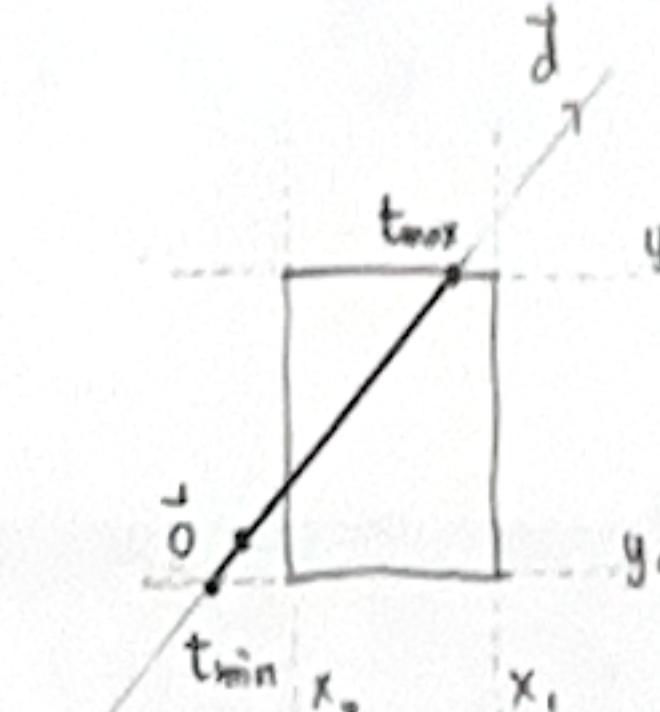
RT3

→ Ray Intersection with Axis-Aligned Box

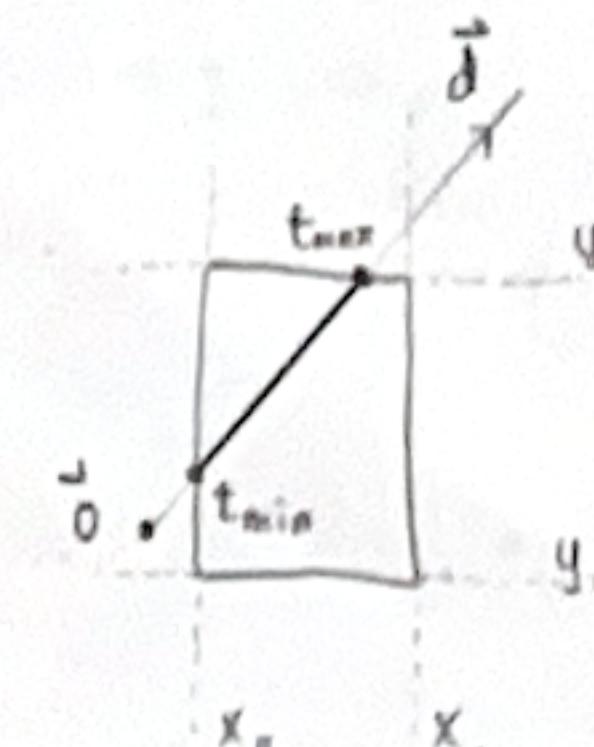
Compute intersections with slabs and take intersection of  $t_{\min}/t_{\max}$  intervals.



Intersection with x plane.



Intersection with y plane.



Final intersection result. (20)

The ray enters the box only when it enters all pairs of slabs.

The ray exits the box as long as it exists any pair of slabs.

$$\text{for 3D} \rightarrow \begin{cases} t_{\min,x}, t_{\min,y}, t_{\min,z} \\ t_{\max,x}, t_{\max,y}, t_{\max,z} \end{cases} \rightarrow \begin{cases} t_{\text{enter}} = \max(t_{\min}) \\ t_{\text{exit}} = \min(t_{\max}) \end{cases}$$

→ if  $t_{\text{enter}} < t_{\text{exit}}$ , we know the ray stays a while in the box.

► Check whether t is negative for physical correctness!

► If  $t_{\text{exit}} < 0$ , the box is "behind" the ray → no intersection!

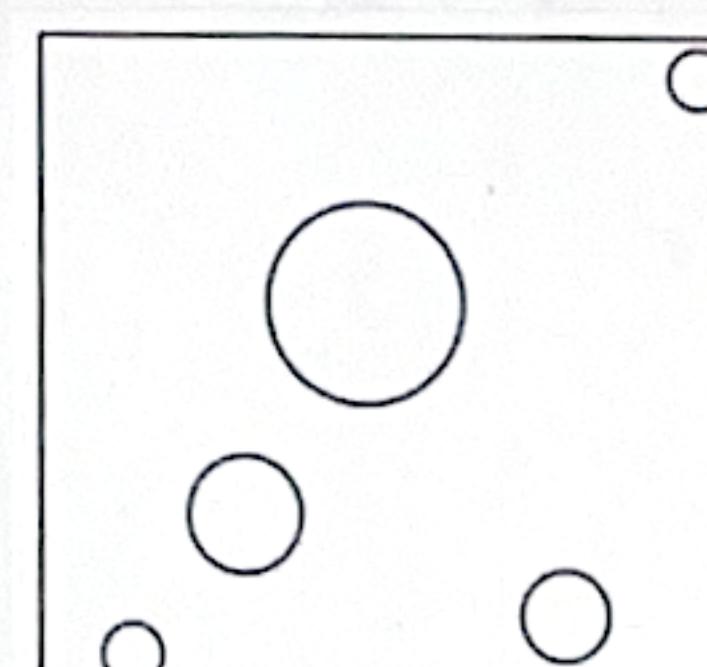
► If  $\begin{cases} t_{\text{exit}} > 0 \\ t_{\text{enter}} < 0 \end{cases}$ , the ray's origin is inside the box → have intersection!

→ Ray and AABB intersect if  $t_{\text{enter}} < t_{\text{exit}} \& t_{\text{exit}} \geq 0$ .

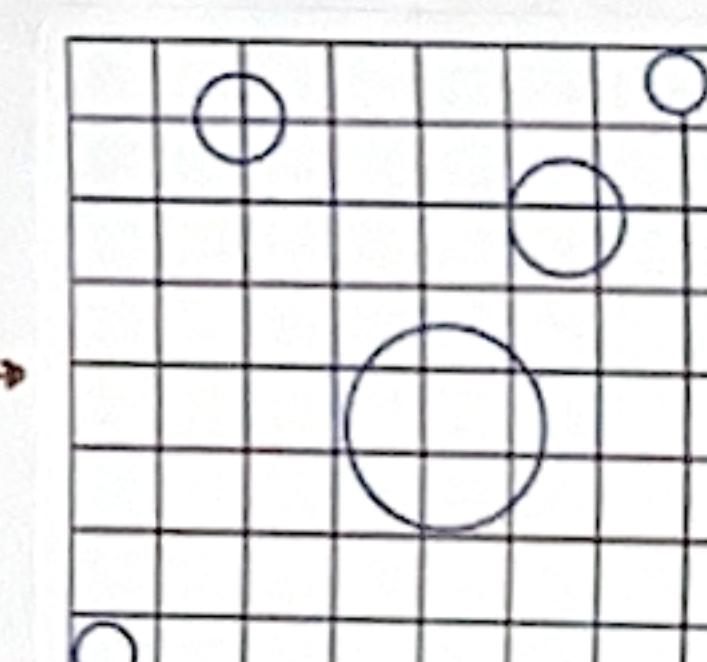
\* Use AABBs to accelerate ray tracing:

1. Uniform spatial partitions grids:

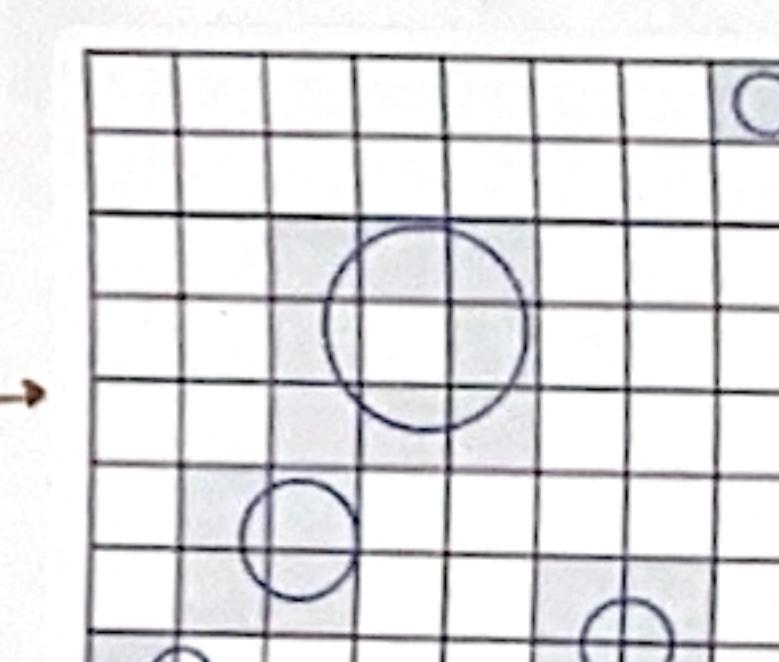
Step 1. Build acceleration grid



a. Find bounding box



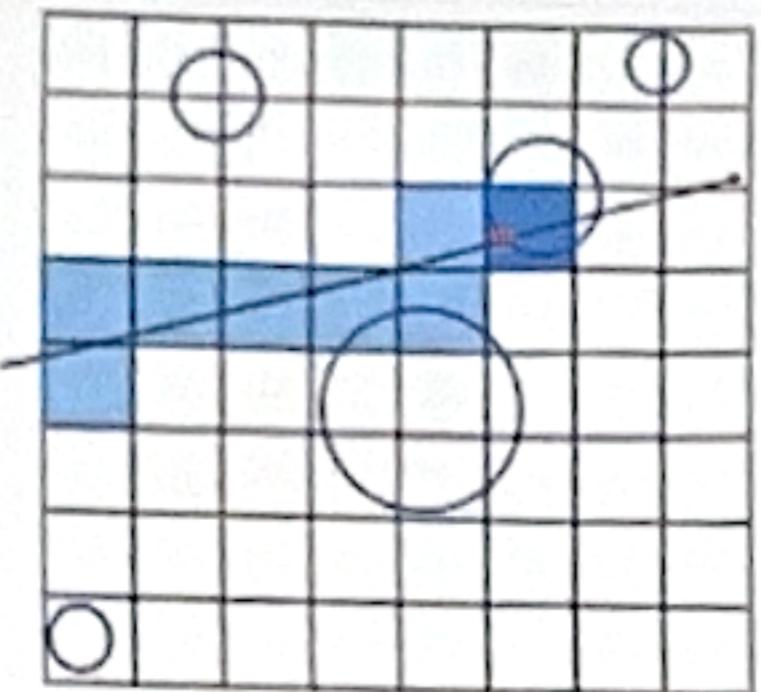
b. Create grid



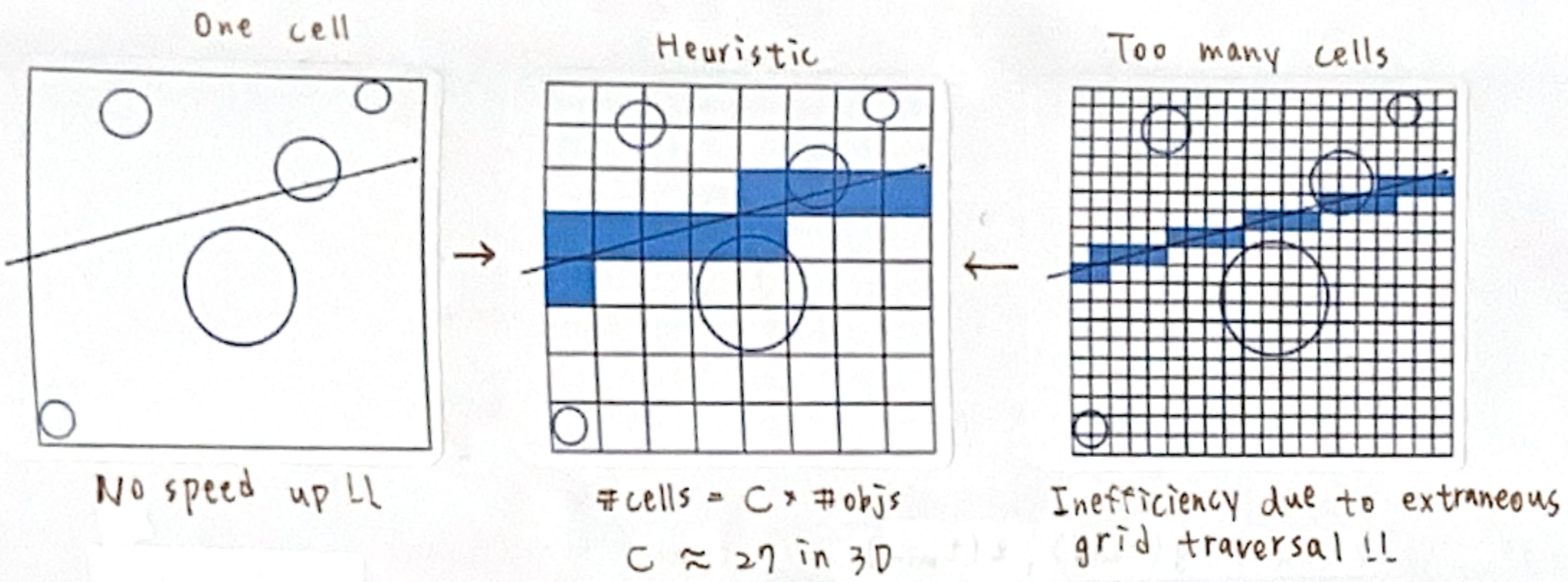
c. Store each object in overlapping cells

## Step.2 Ray - Scene Intersection

- a. Step through grid in ray traversal order.



- Grid resolution:



- Results:

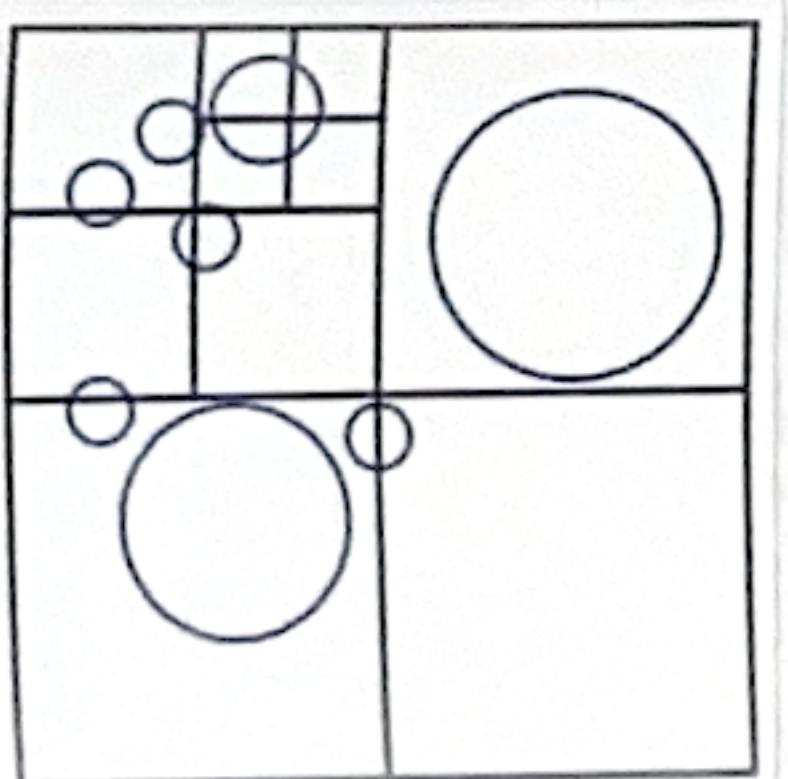


Grids work well on large collection of objects that are distributed evenly in size and space.

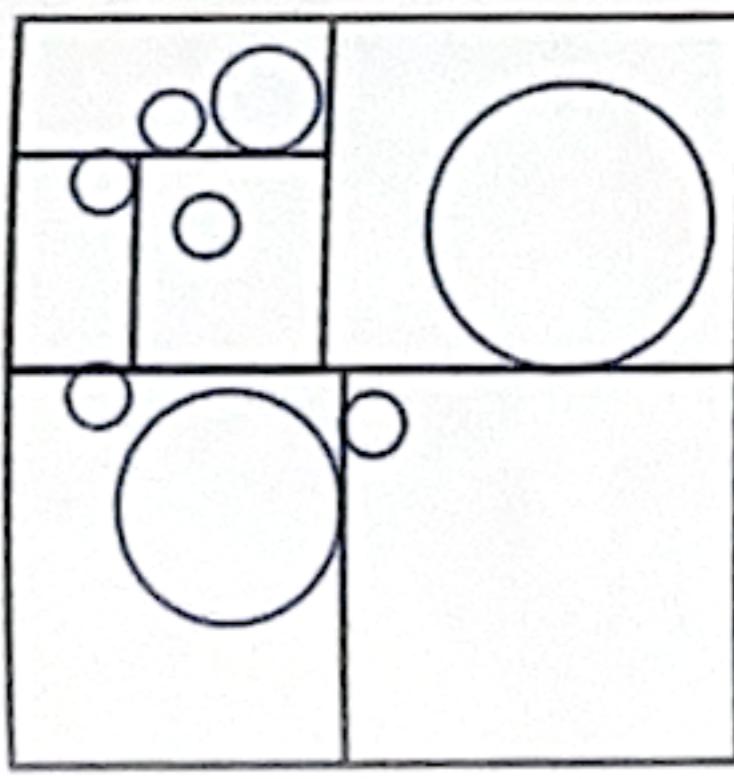


"Teapot in a stadium" PROBLEM!!

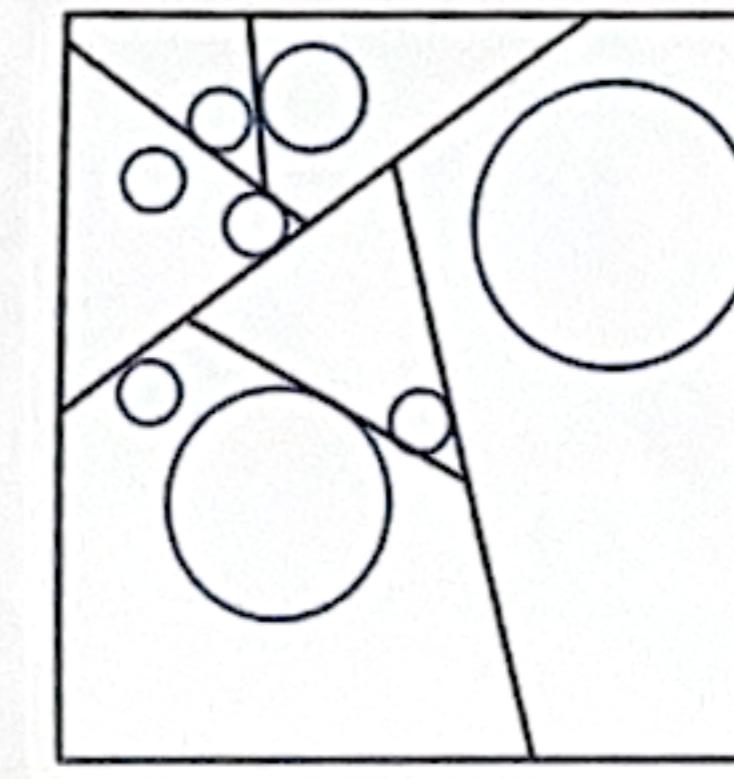
- 2- Spatial partitions:



Oct-Tree



KD-Tree

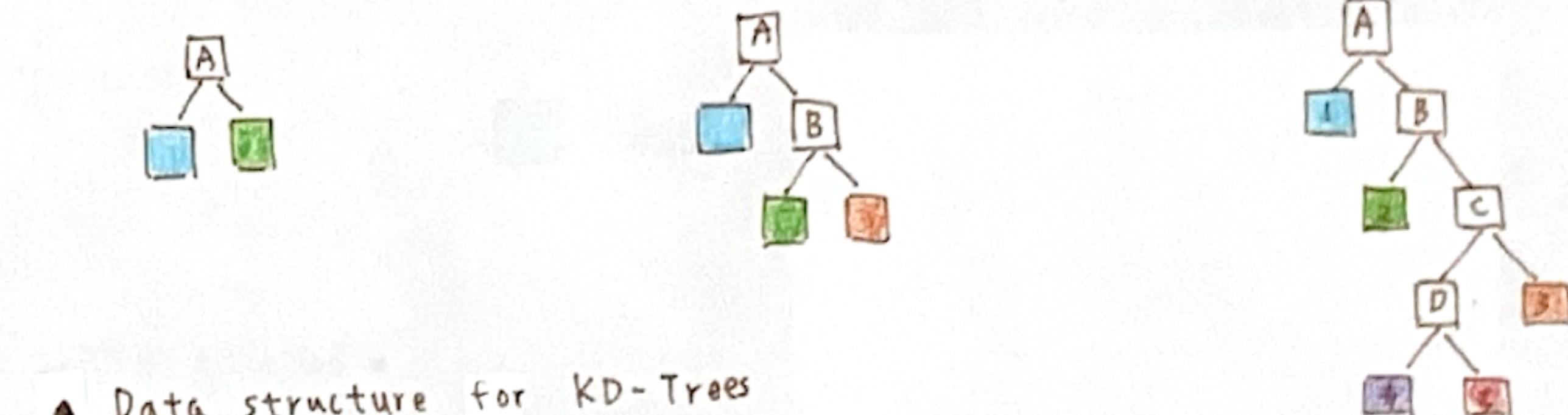
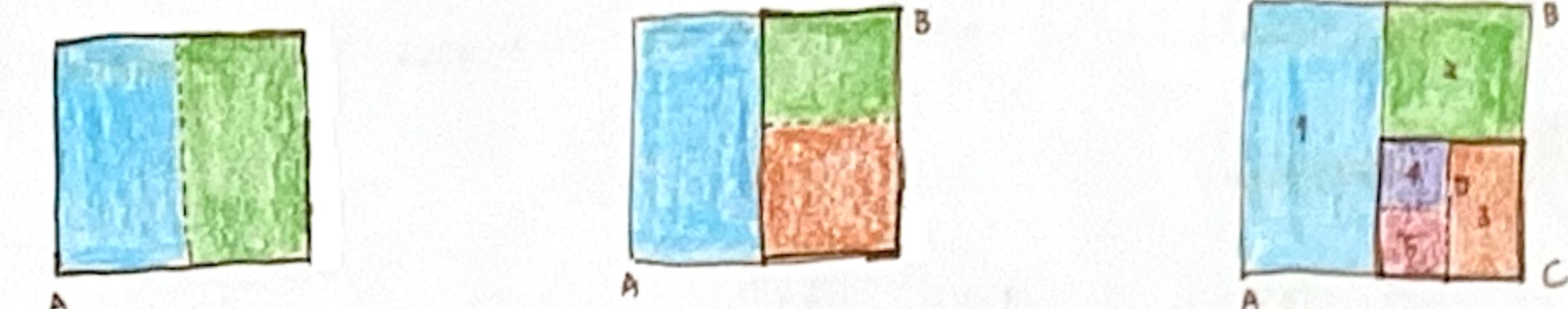


BSP-Tree

p14

## \* KD-Tree

### step.1 KD-Tree Pre-Processing



### ▲ Data structure for KD-Trees

#### 1. Internal node store

a. split axis: x-, y-, or z-axis

b. split position: coordinate of split plane along axis

c. children: pointers to children nodes

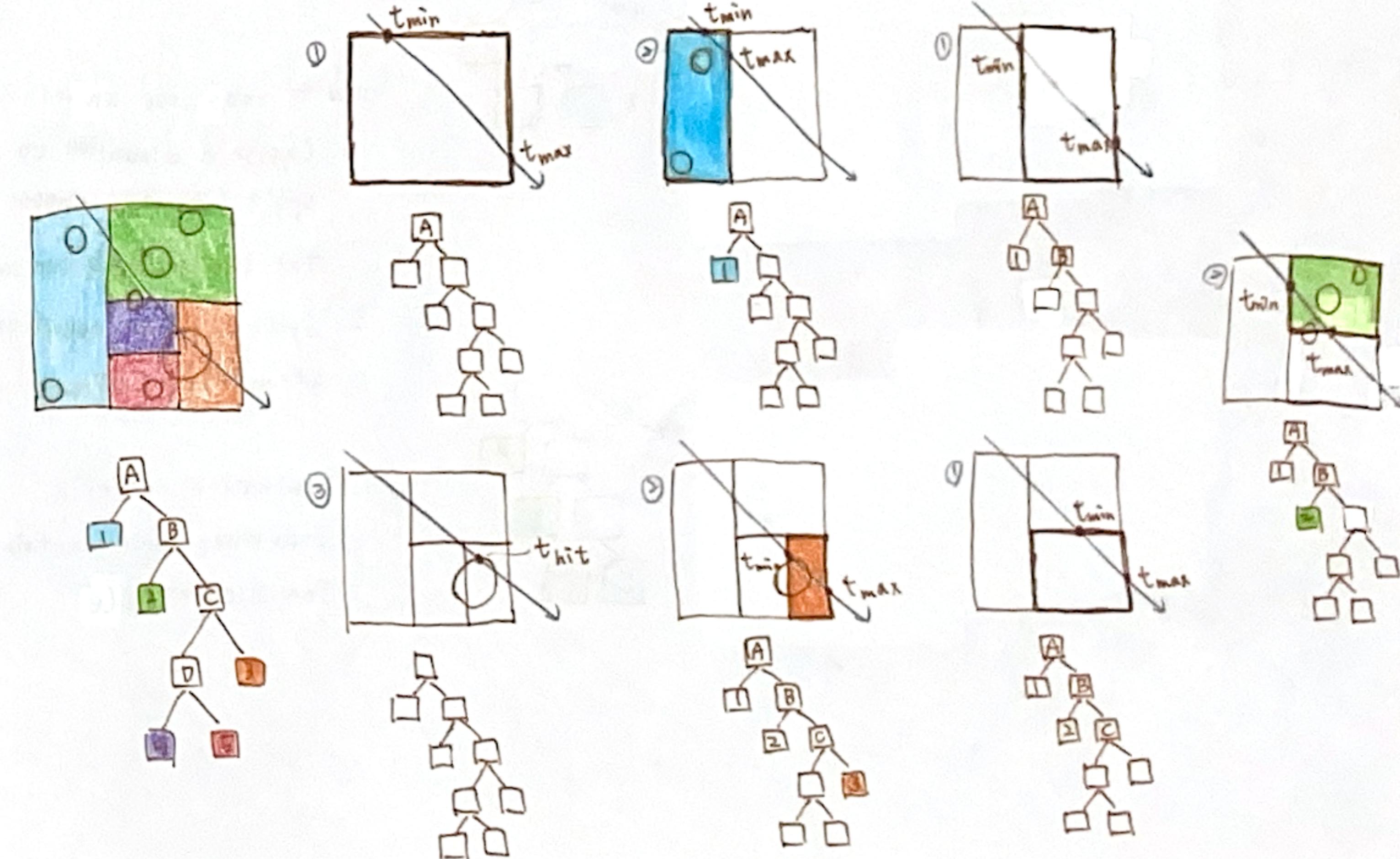
NOTICE!! No objects are stored in internal nodes.

#### 2. Leaf node store

a. list of objects.

### Step.2 Traversing a KD-Tree

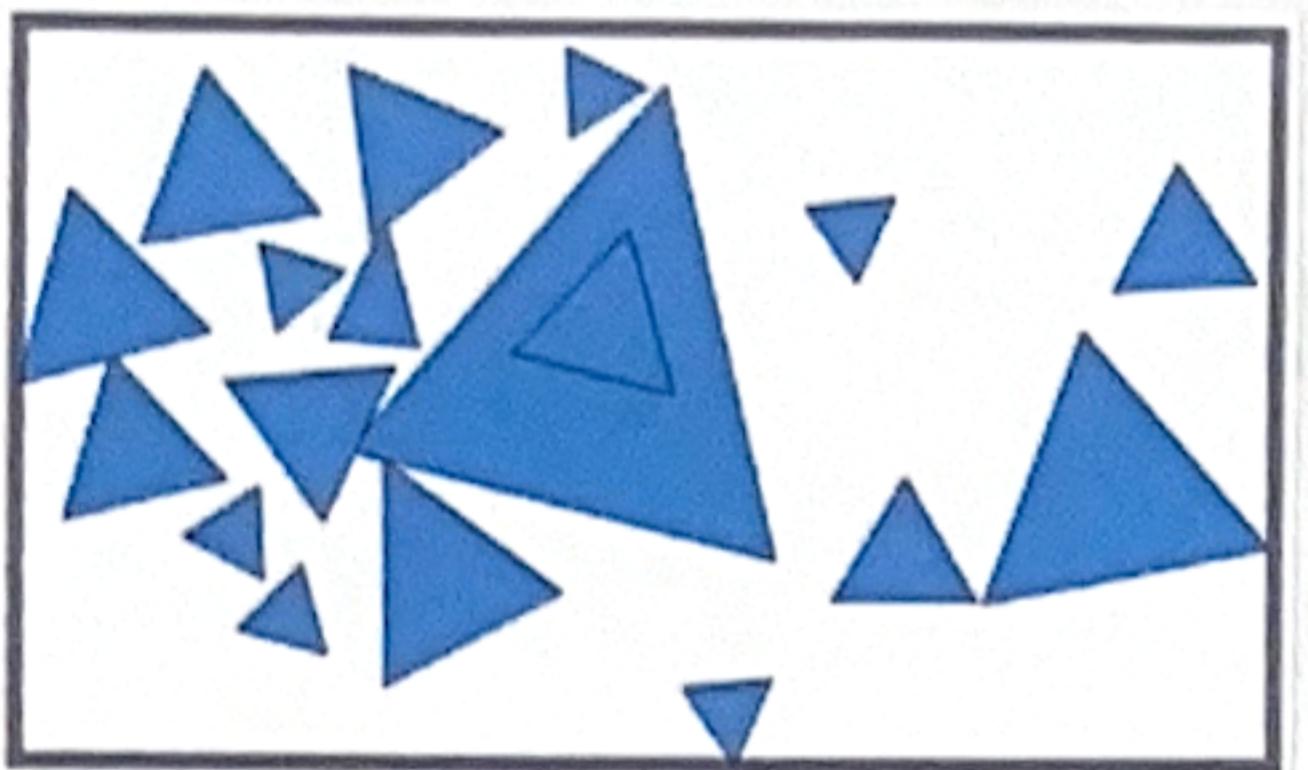
Internal node: split → ①  
Leaf node: intersect all objects → ②  
Intersection found!! → ③



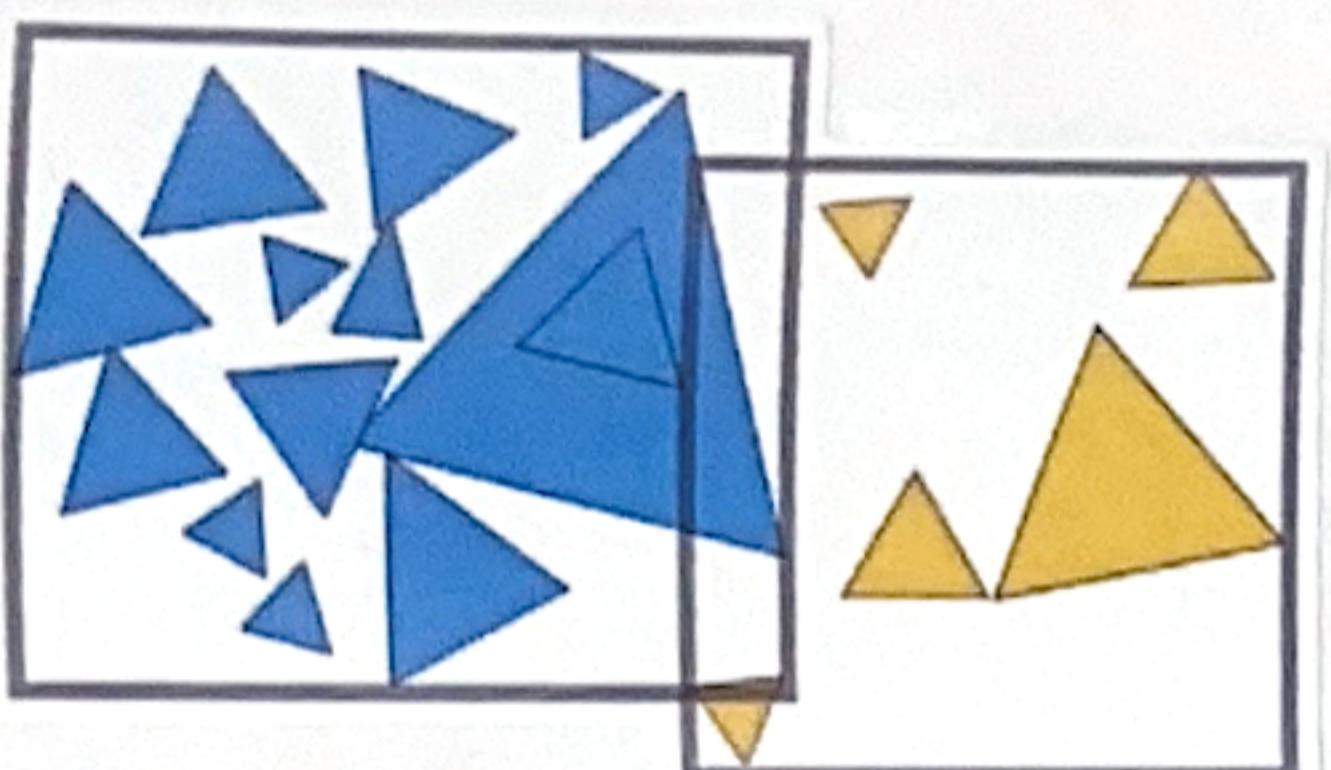
## KD-Trees problems:

1. An object may exist in multiple bounding boxes.
2. Must consider the intersection of triangles and bounding boxes.
3. Object partitions:

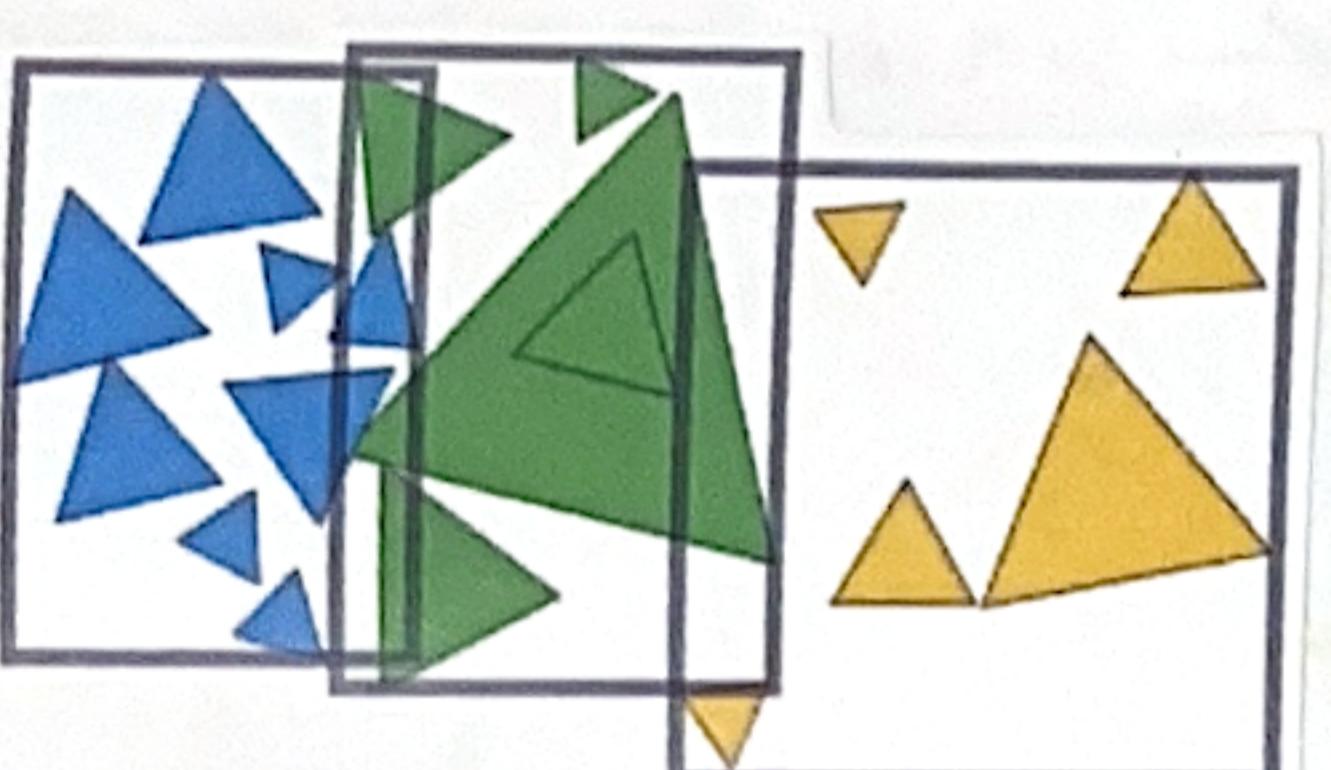
## \* Bounding Volume Hierarchy (BVH)



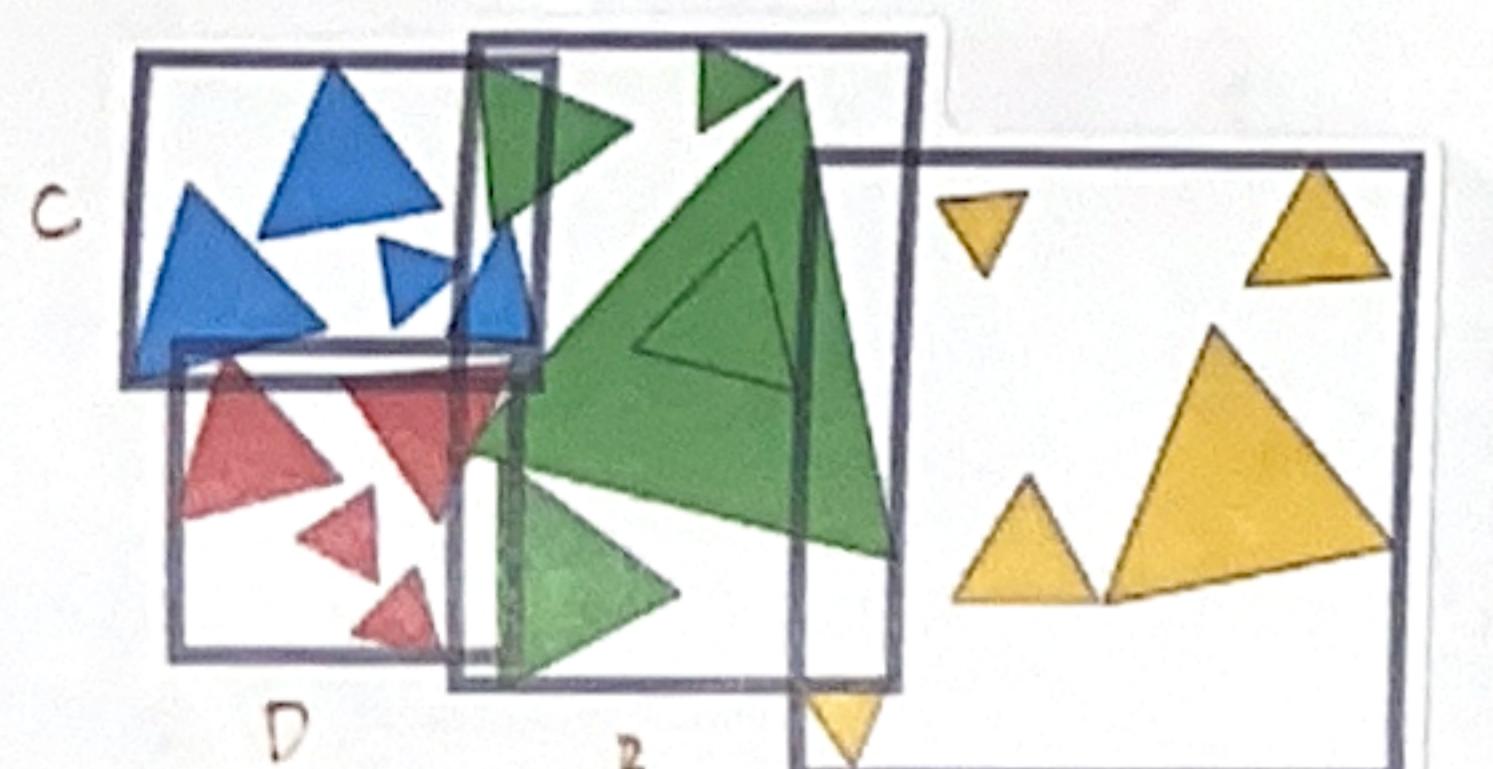
Root → [Blue Box]



Root → [Blue Box, Yellow Box]



Root → [White Box, Yellow Box, Green Box]



Root → [White Box, Yellow Box, Green Box, Red Box]

### Building BVHs:

1. Finding bounding box.
2. Recursively split set of objects in two subsets.
3. Recompute the bounding box of the subsets.
4. Stop when necessary
5. Store objects in each leaf node.

### How to subdivide a node?

1. Choose a dimension to split (Always choose the longest axis on node)
2. Split node at location of median object.

3. Termination criteria:  
stop when node contains few elements.

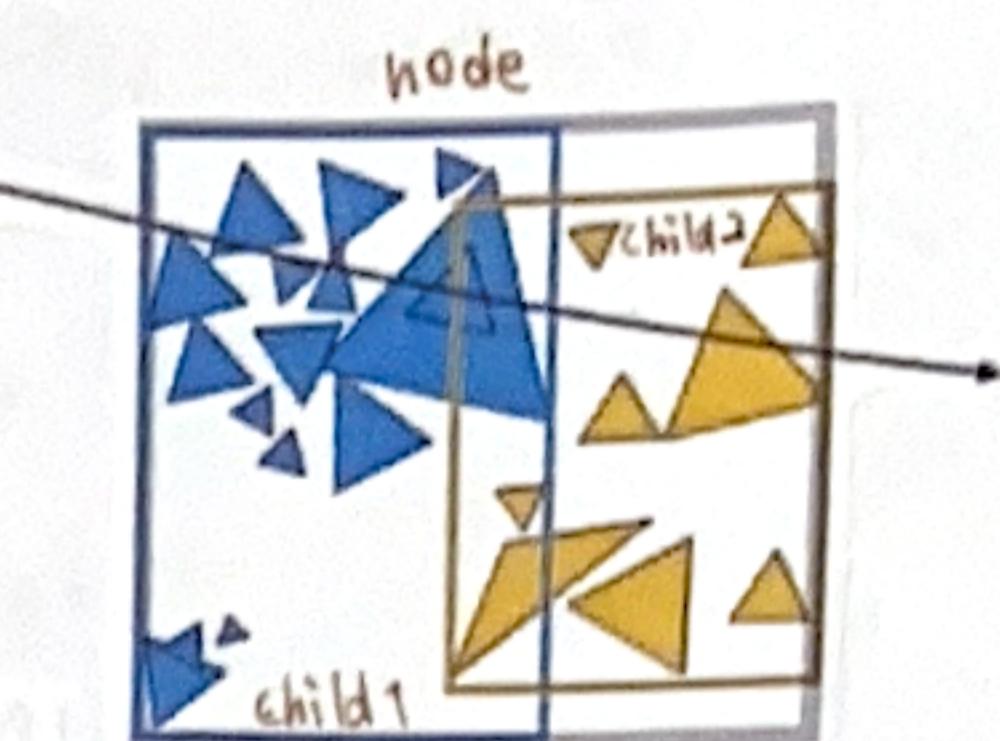
RT5

### Data structure for BVHs:

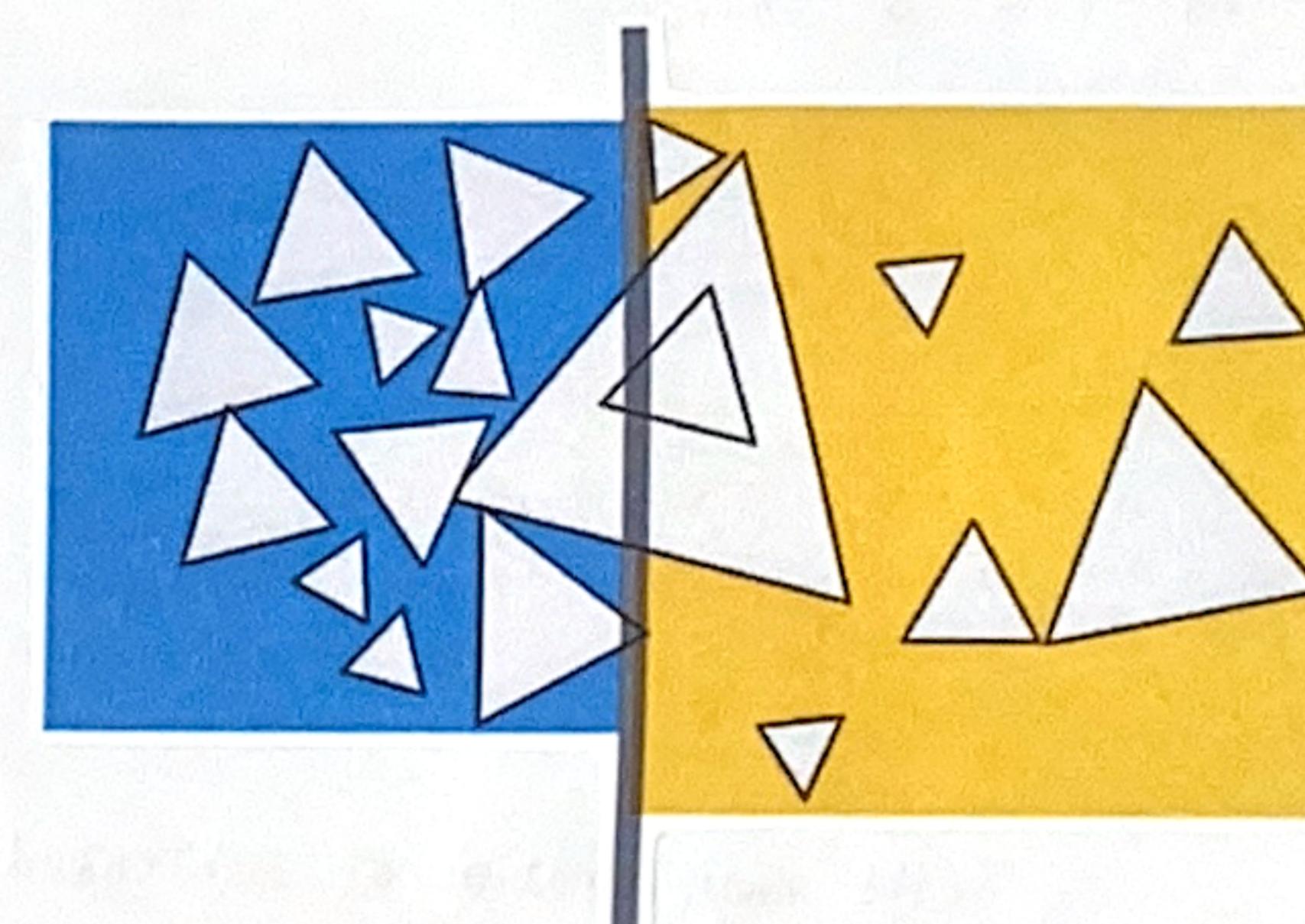
1. Internal nodes store { Bounding box  
children : pointers to child nodes }
2. Leaf nodes store { Bounding box  
List of objects }

### 3. coding (in C++):

```
Intersect (Ray ray, BVH node) {
    if (ray misses node.bbox) return;
    if (node is a leaf node)
        test intersection with all objs;
        return closest intersection;
    hit1 = Intersect (ray, node.child1);
    hit2 = Intersect (ray, node.child2);
    return the closer of hit1, hit2;
}
```

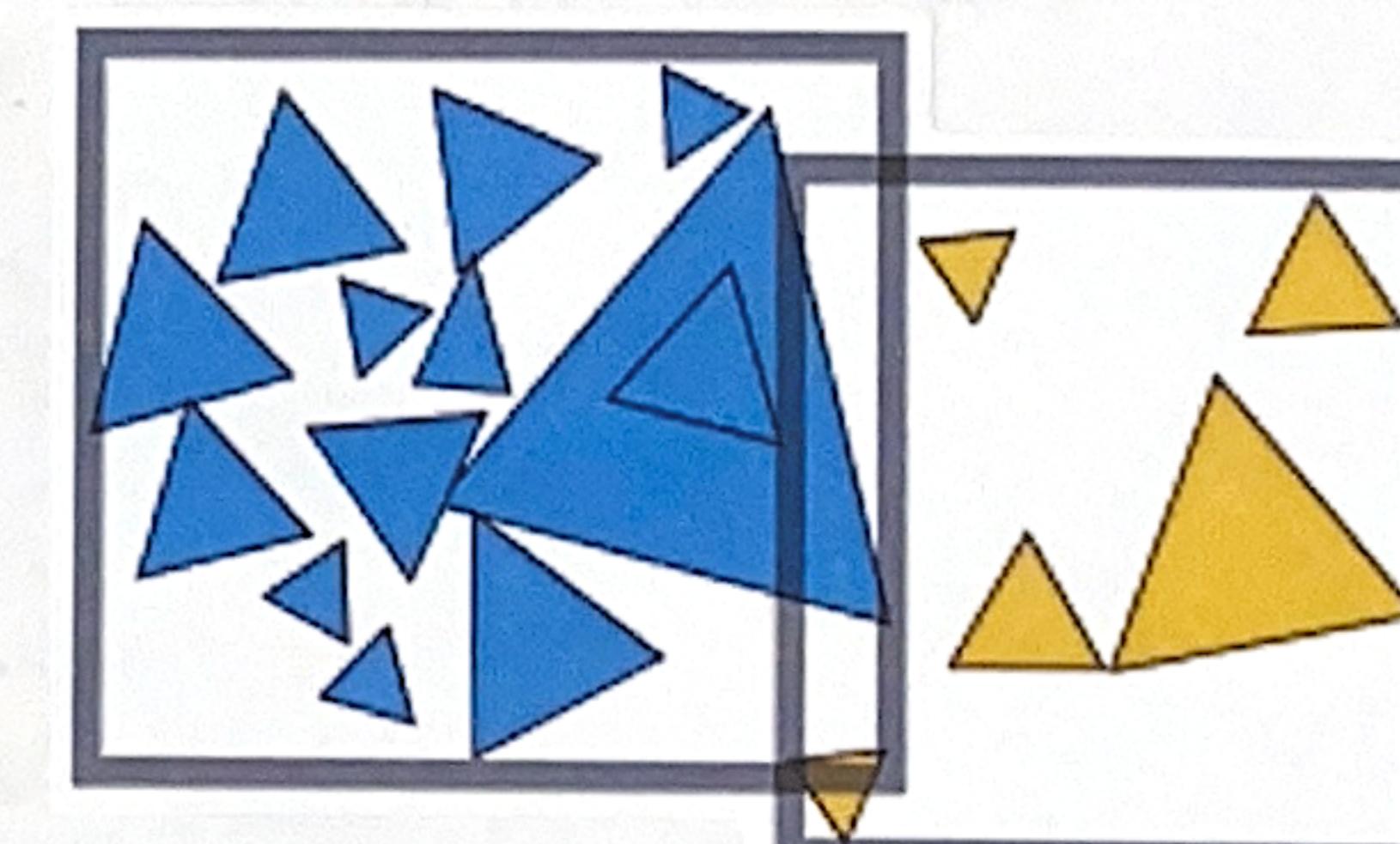


### To summary:



### Spatial partition (e.g. KD-tree):

1. Partition space into non-overlapping regions
2. An object can be contained in multiple regions.



### Object partition (e.g. BVH):

1. Partition set of objects into disjoint subsets.
2. Bounding boxes for each set may overlap in space.

## \* Radiometry: (辐射能量学)

Why? In Blinn-Phong model, light intensity  $I$  is 10, for example.

But what is 10?

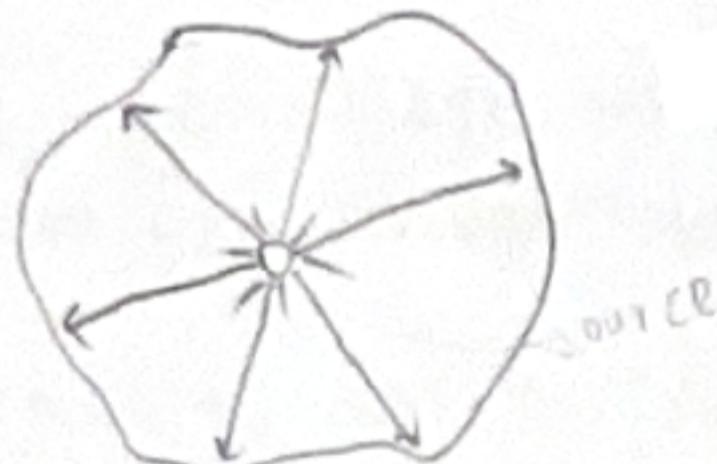
- ⇒
  - ① Measurement system and units for illumination.
  - ② Accurately measure the spatial properties of light.  
ex. Radiant flux, intensity, irradiance, radiance.
  - ③ Perform lighting calculation in physically correct manner.

### 1. Radiant energy and flux (Power):

Definition: Radient energy = the energy of electromagnetic radiation.  
 $\text{Q} \text{ (J=Joule)}$

Radiant flux (power): the energy emitted, reflected, transmitted or received, per unit time.  
 $\Phi = \frac{dQ}{dt}$  ( $\text{lm}=\text{lumen}$ ) ( $\text{W}=\text{Watt}$ )

### 2. Radiant intensity: Light emitted from a source.

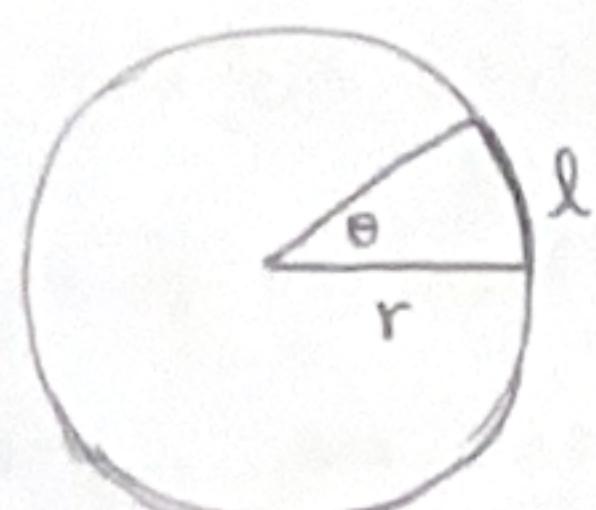


Definition: the power per unit solid angle emitted by a point light source.

$$I(w) = \frac{d\Phi}{d\omega} \quad (\frac{W}{sr} \text{ or } \frac{lm}{sr}) \quad (cd = candela)$$

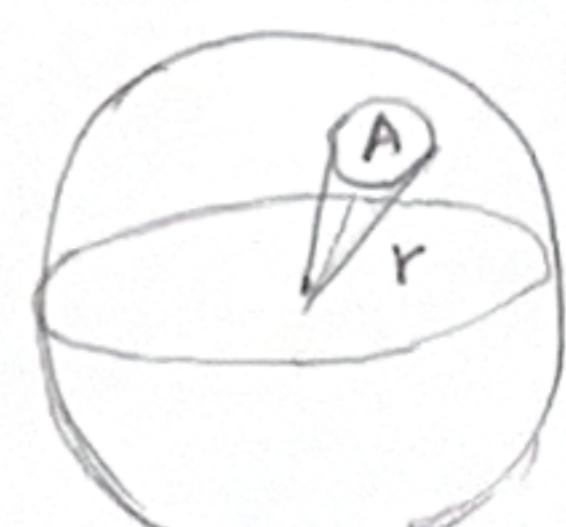
Angle: ratio of subtended arc length on circle to radius. (circle has  $2\pi$  radians)

$$\theta = \frac{l}{r}$$



Solid angle: ratio of subtended area on sphere to radius square.  
(sphere has  $4\pi$  steradians)

$$\Omega = \frac{A}{r^2}$$



RT6

### ▲ Differential of solid angle:

$$dA = (r d\theta)(r \sin\theta d\phi) = r^2 \sin\theta d\theta d\phi$$

$$d\omega = \frac{dA}{r^2} = \sin\theta d\theta d\phi$$

對日來說不均勻

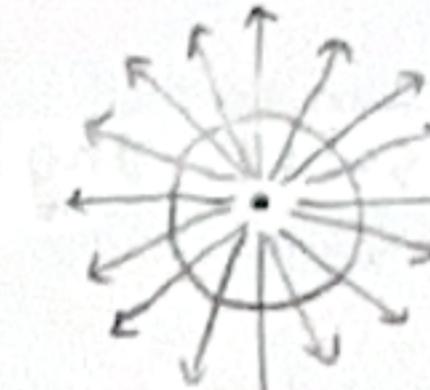
$$\text{Sphere: } S^2 \rightarrow \Omega = \int_{S^2} d\omega$$

$$= \int_0^{2\pi} \int_0^\pi \sin\theta d\theta d\phi$$

$$= 4\pi$$

use  $w$  to denote a direction vector (unit length)

### ▲ Radiant intensity example: Isotropic point source.



Radiant flux on the whole sphere:

$$\bar{\Phi} = \int_{S^2} \Phi d\omega^2 = 4\pi \Phi$$

$$\Rightarrow I(w) = \frac{\bar{\Phi}}{4\pi}$$

### 3. Irradiance: Light falling on a surface

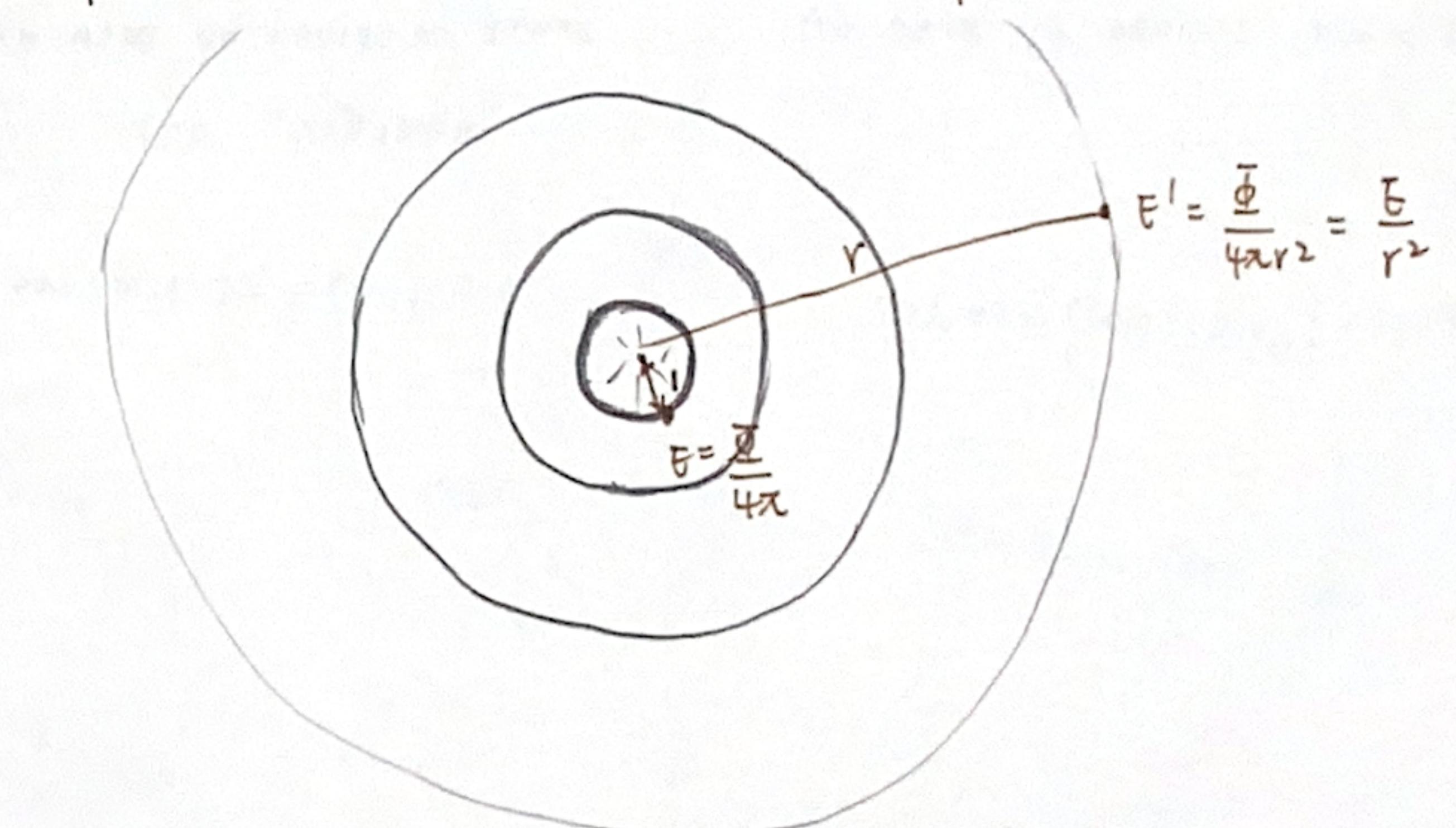
Definition: the power per unit area incident on a surface point.



$$E(x) = \frac{d\bar{\Phi}(x)}{dA} \quad (\frac{W}{m^2} \text{ or } \frac{lm}{m^2} = lux)$$

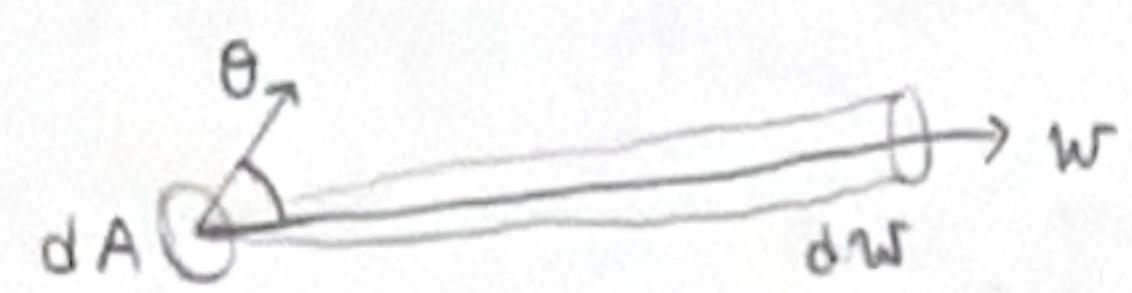
### ▲ Irradiance example:

Assume light is emitting power  $\bar{\Phi}$  in a uniform angular distribution, compare irradiance at surface of two spheres:



4. Radiance: light traveling along a ray

↳ Rendering is all about computing radiance



Definition: the power emitted, reflected, transmitted or received by a surface, per unit solid angle, per projected unit area.

$$L(p, w) = \frac{d^2 \Phi(p, w)}{dw dA \cos \theta} \left( \frac{w}{sr m^2} \text{ or } \frac{cd}{m^2} = \frac{lm}{sr m^2} = hit \right)$$

accounts for projected surface area

a. Incident Radiance: the irradiance per unit solid angle arriving at the surface.



$$L(p, w) = \frac{dE(p)}{dw \cos \theta}$$

↳ the light arriving at the surface along a given ray (point on surface and incident direction)

b. Exiting Radiance: the intensity per unit projected area leaving the surface.



$$L(p, w) = \frac{dI(p, w)}{dA \cos \theta}$$

↳ For an area light it is the light emitted along a given ray (point on surface and exit direction)

Irradiance

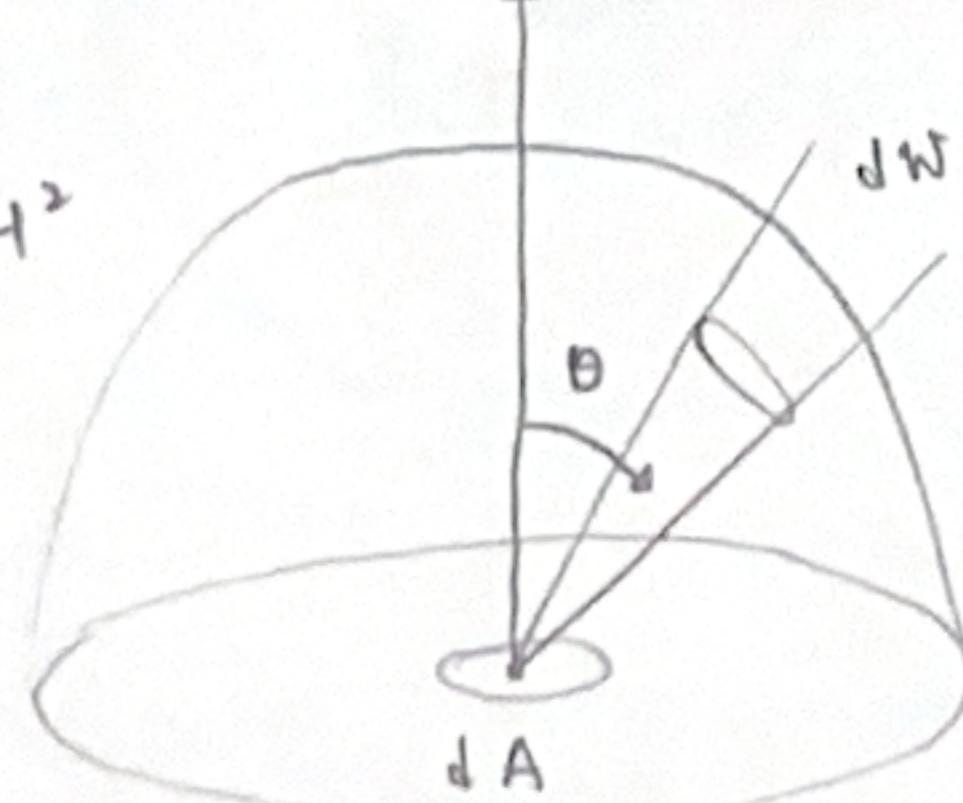
total power received by area dA

Radiance

power received by area dA from "direction" dw

$$E(p) = \int_{H^2} L_i(p, w) \cos \theta dw$$

Unit hemisphere =  $H^2$

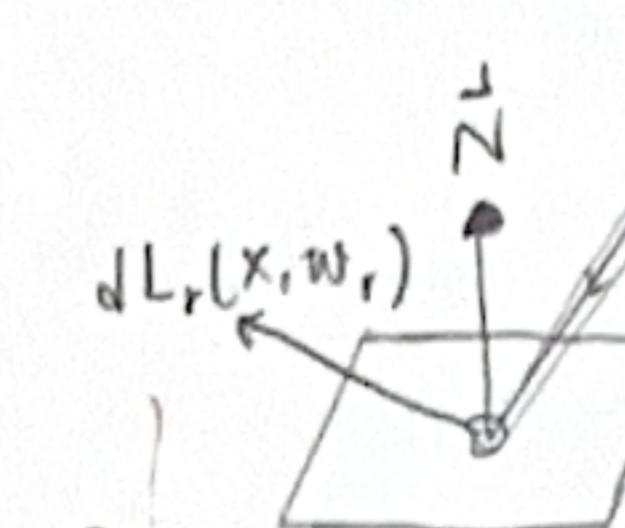


$$dE(p, w) = L_i(p, w) \cos \theta dw$$

\* Bidirectional reflectance distribution function: (BRDF)

双向反射分布函数

Reflection at a point:



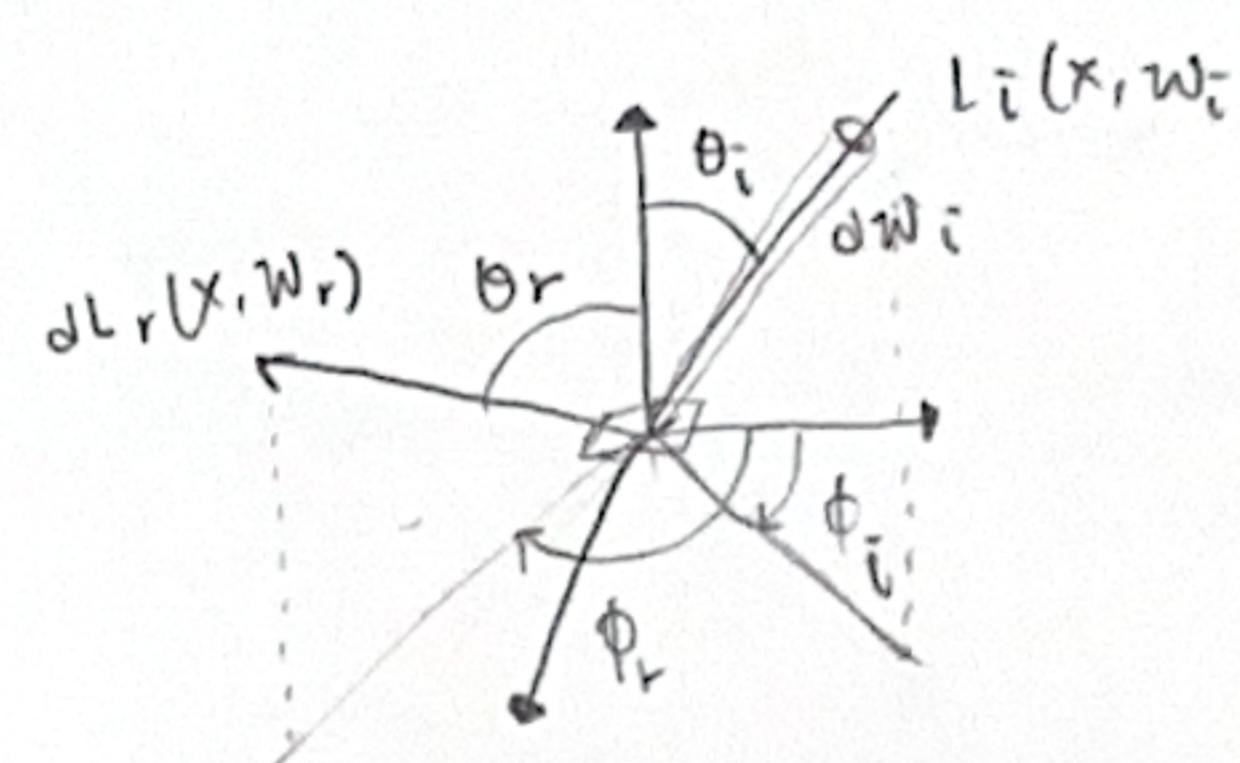
Differential irradiance incoming: Radiance from direction  $w_i$  turns into the power  $E$  that  $dA$  receives.

$$dE(w_i) = L(w_i) \cos \theta_i dw_i$$

Differential radiance exiting (due to  $dE(w_i)$ ): the power  $E$  will become the radiance to any other direction  $w_o$ .

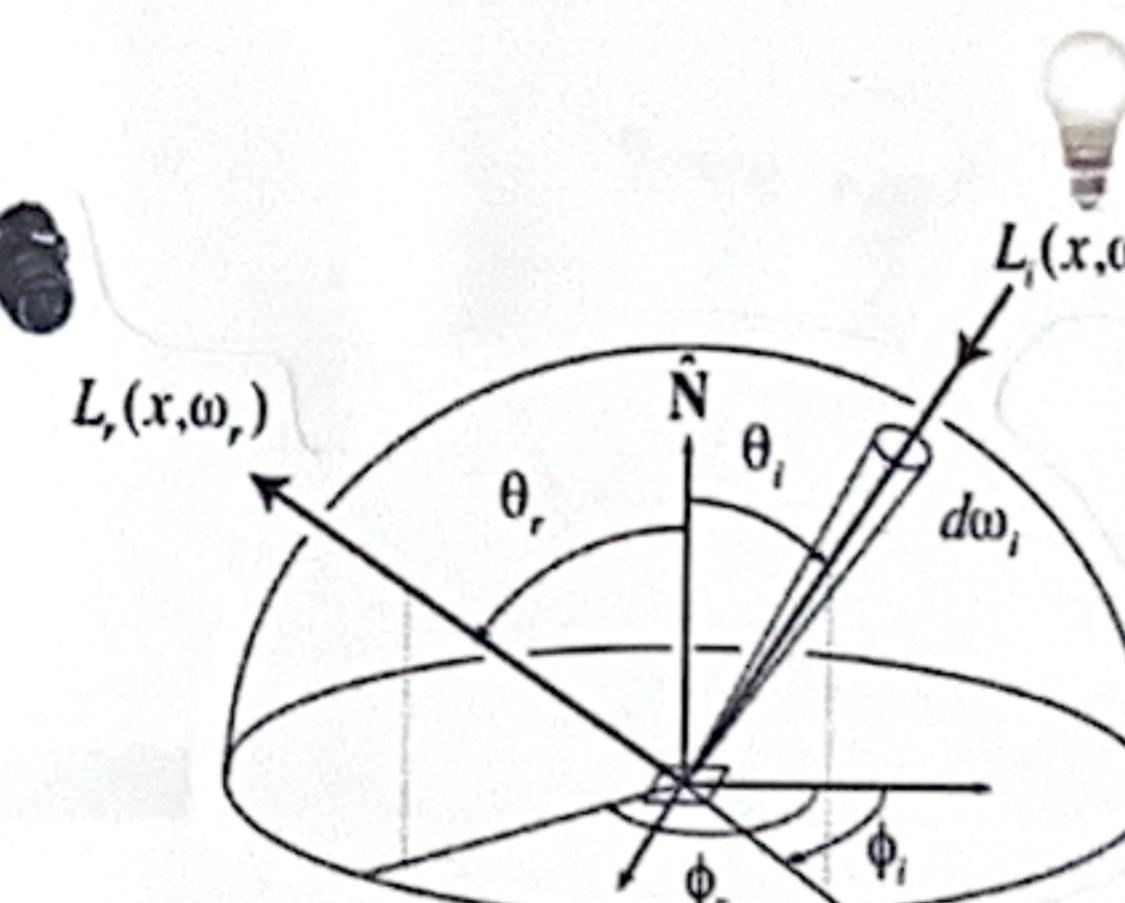
$$dL_r(w_r)$$

⇒ BRDF represents how much light is reflected into each outgoing direction  $w_r$  from each incoming direction.



$$f_r(w_i \rightarrow w_r) = \frac{dL_r(w_r)}{dE_i(w_i)} = \frac{dL_r(w_r)}{L_i(w_i) \cos \theta_i dw_i} \left( \frac{1}{sr} \right)$$

⇒ Therefore, the reflection equation is



$$L_r(p, w_r) = \int_{H^2} f_r(p, w_i \rightarrow w_r) L_i(p, w_i) \cos \theta_i dw_i$$

1. Reflected radiance depends on incoming radiance
2. Incoming radiance depends on reflected radiance (at another point in the scene)

⇒ Last, adding emission term, it become rendering equation

$$L_o(p, w_o) = L_o(p, w_o) + \int_{\Omega^+} L_i(p, w_i) f_r(p, w_i, w_o) (n \cdot w_i) dw_i$$

\* NOTE!! From now, we assume that all directions are pointing outwards.

## Solve rendering equation

$$\text{UNKNOWN } L_r(x_i, w_r) = L_e(x_i, w_r) + \int_{\Omega} L_r(x'_i, -w_i) f(x_i, w_i, w_r) \cos \theta_i d\omega_i$$

reflected light Emission      Reflected light BRDF  
(Output image)

$$l(u) = e(u) + \int l(v) K(u, v) dv$$

kernal of equation  
Light transport operator

$$L = E + KL$$

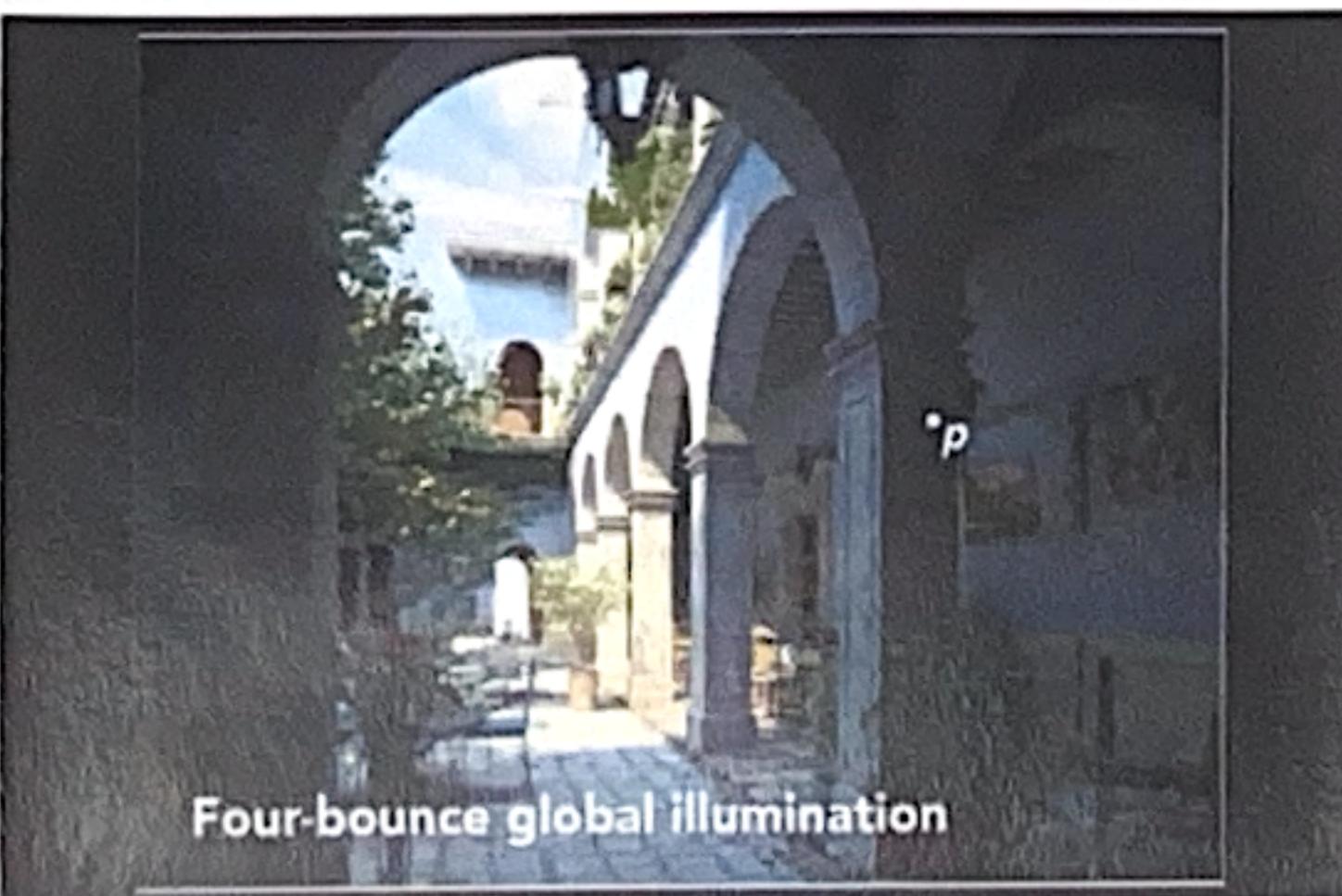
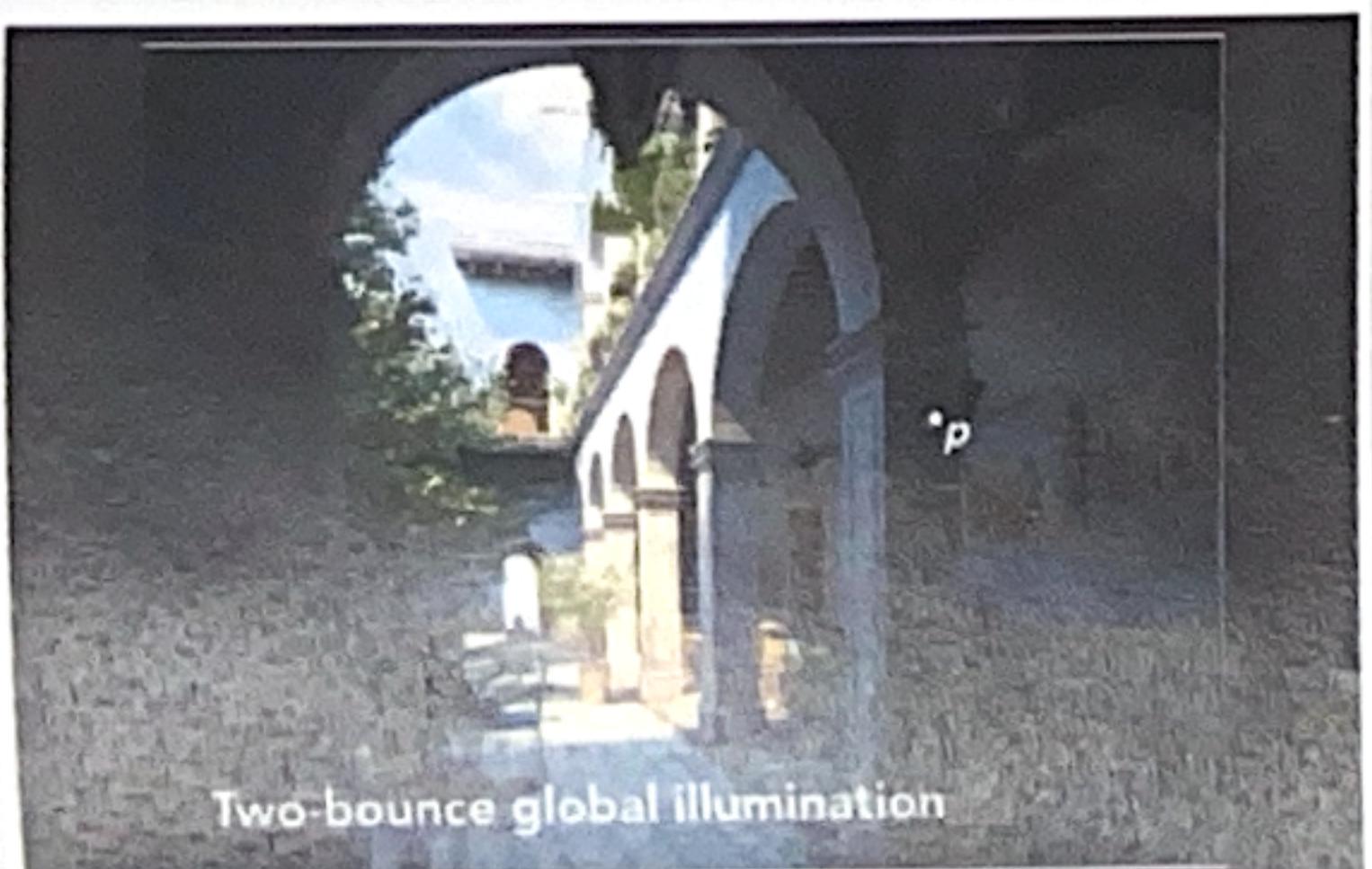
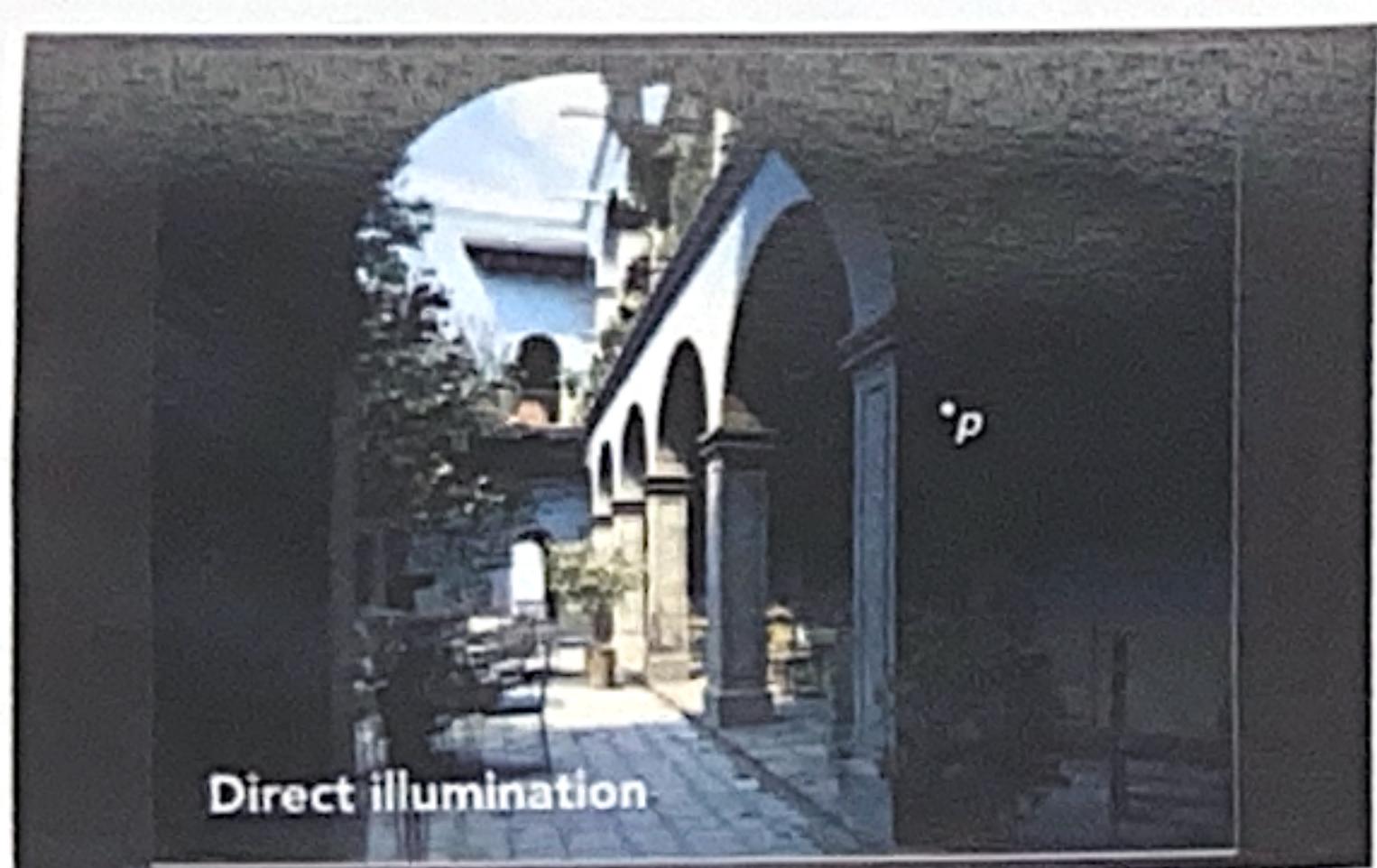
Matrix equation  
L, E: matrix  
K: light transport matrix

$$L = (I - K)^{-1} E$$

$$L = (I + K + K^2 + K^3 + \dots) E = E + KE + K^2E + K^3E + \dots$$

Shading in rasterization

$E$ : Emission directly from light sources
$KE$ : Direct illumination on surfaces
$K^2E$ : Indirect illumination (One bounce indirect)
$K^3E$ : (Two bounce indirect illumination)

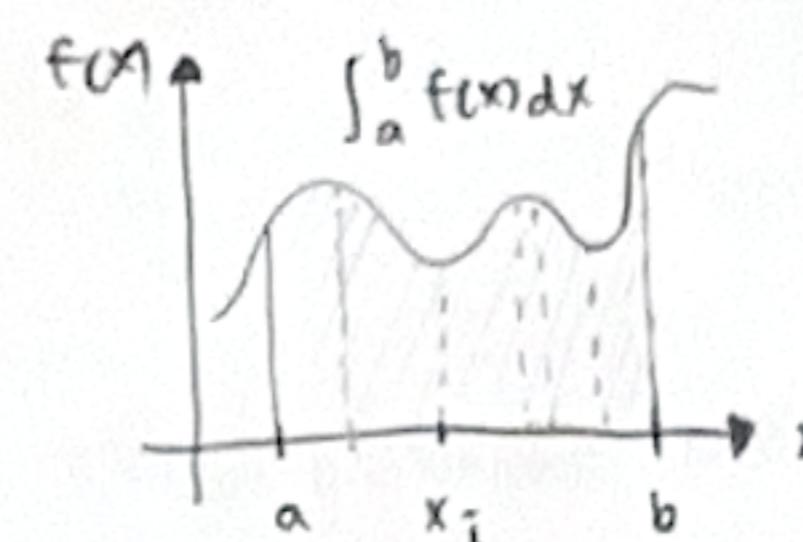


RT8

## Monte Carlo Integration

why? we want to solve an integral, but it can be too difficult to solve analytically.

How? estimate the integral of a function by averaging random samples of the function's value.



Define integral  $\int_a^b f(x) dx$

Random variable  $X_i \sim p(x)$

$$\Rightarrow \text{Monte Carlo estimator } F_N = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)}$$

Example: Uniform Monte Carlo estimator

Define integral  $\int_a^b f(x) dx$

$$\text{Uniform random variable } X_i \sim p(x) = C = \frac{1}{b-a}$$

$$\Rightarrow \text{Basic Monte Carlo estimator } F_N = \frac{b-a}{N} \sum_{i=1}^N f(X_i)$$

$$\text{To sum up: } \int f(x) dx = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)}, \quad X_i \sim p(x)$$

① The more samples, the less variance.

② Sample on  $x$ , integrate on  $x$ .

## Path tracing:

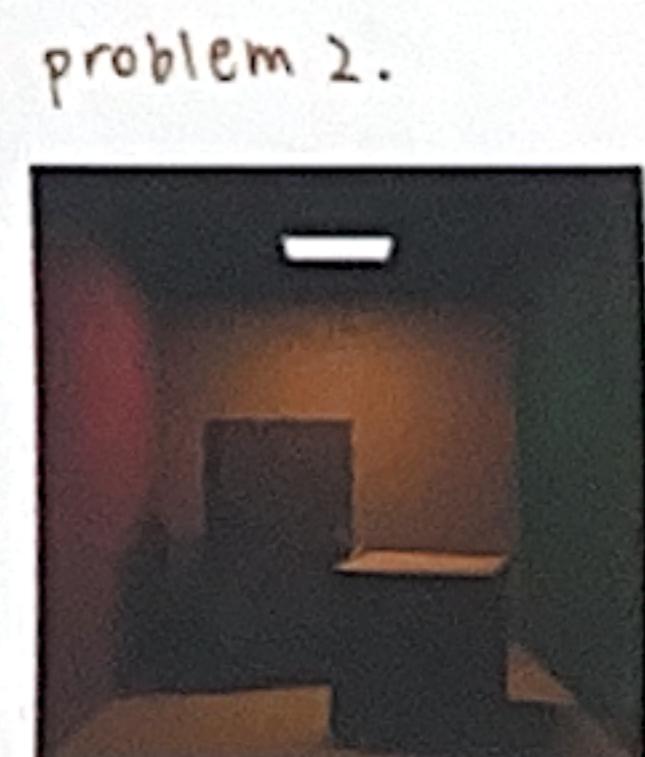
preview about Whitted-style ray tracing:

1. Always perform specular reflections / refractions.
2. Stop bouncing at diffuse surfaces.

problem 1.



Mirror reflection



problem 2.



Glossy reflection

Path traced:  
direct illumination

Path traced:  
global illumination

The Utah teapot

The Cornell box

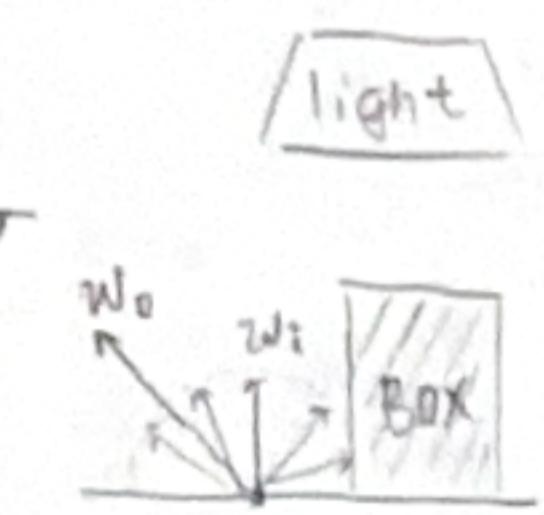
## Solving the path tracing coding step by step:

邊解來問題

Because the Whitted-style ray tracing is wrong, the rendering equation  $L_o(p, w_o) = L_e(p, w_o) + \int_{\Omega} L_i(p, w_i) f_r(p, w_i, w_o) (n \cdot w_i) d w_i$  is correct. It involves

- (1) Solving an integral over the hemisphere
- (2) Recursive execution.

Suppose we want to render one pixel (point) in the following scene for direct illumination only



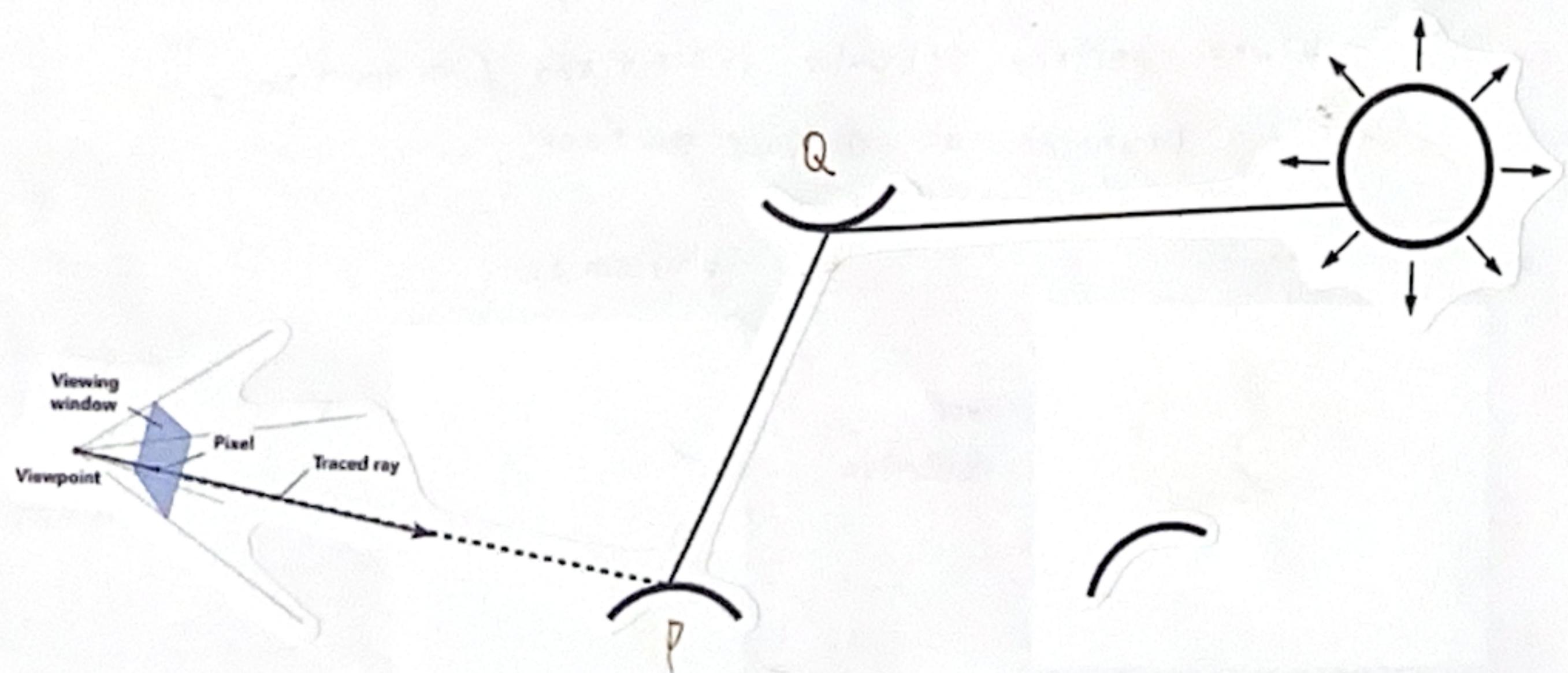
$$L_o(p, w_o) = \int_{\Omega} L_i(p, w_i) f_r(p, w_i, w_o) (n \cdot w_i) d w_i$$

- ①  $f(x)$  in Monte Carlo
- ② integration over directions  $\rightarrow$  Monte Carlo integration
- ③  $P(w_i) = \frac{1}{2\pi}$ , because of assuming uniformly sampling the hemisphere.

$$\approx \frac{1}{N} \sum_{i=1}^N \frac{L_i(p, w_i) f_r(p, w_i, w_o) (n \cdot w_i)}{P(w_i)}$$

(Code in next page without part A)

One more step forward global illumination



When the ray hit object Q directly, Q reflects light to P.

How much? The direct illumination at Q!

$\downarrow$   
Code in the next page  
with part A

RT9

Coding:

shape(p, w\_o)

Randomly choose N directions  $w_i \sim p_{df}$

$L_o = 0.0$

For each  $w_i$

Trace a ray  $r(p, w_i)$

If ray r hit the light

$L_o += (1/N) * L_i * f_r * cosine / pdf(w_i)$

part A Else If ray r hit an object at q

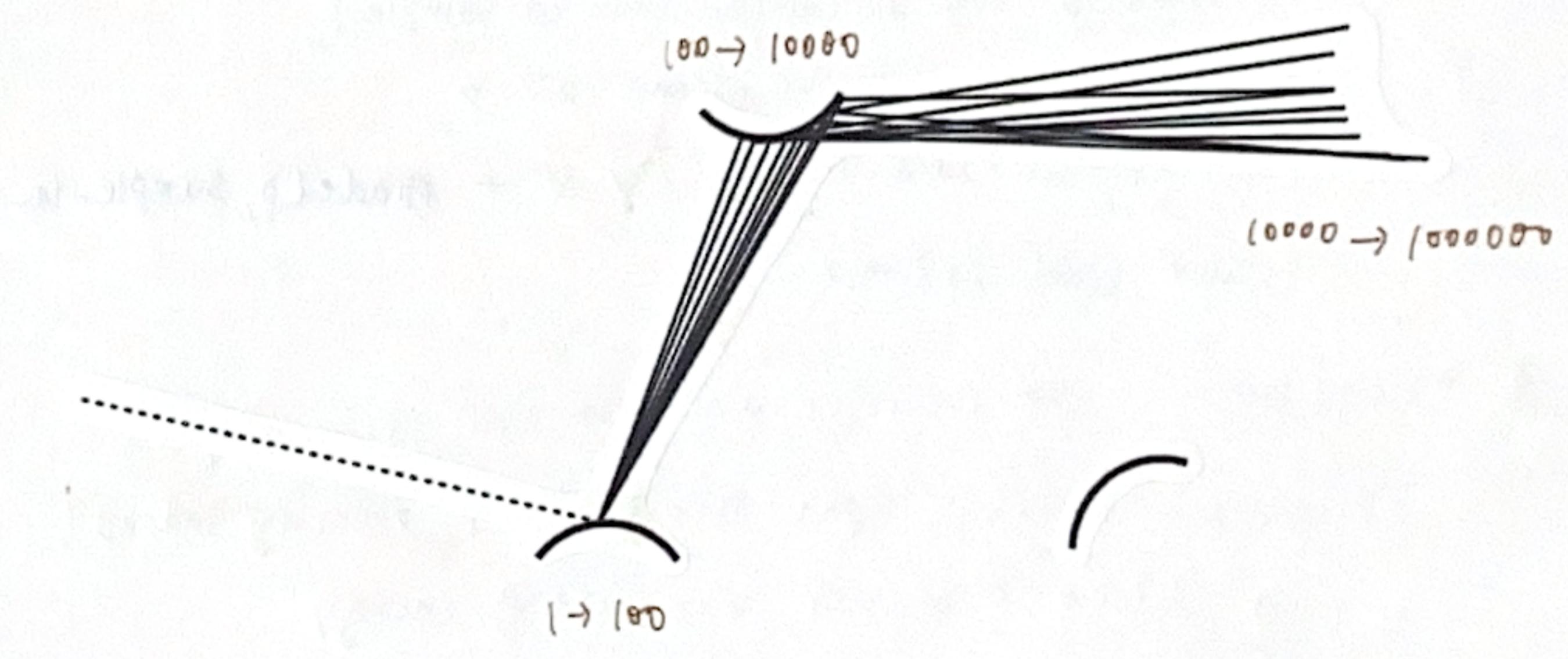
$L_o += (1/N) * shade(q, -w_i) + f_r * cosine / pdf(w_i)$

Return  $L_o$

NOTICE the direction!

PROBLEM 1: Explosion of #rays as #bounces go up

$\rightarrow$  #rays =  $N^{\#bounces}$



SOLUTION: assume that only one ray is traced at each shading point.

Coding:

shape(p, w\_o)

Randomly choose ONE direction  $w_i \sim p_{df}(w)$

Trace a ray  $r(p, w_i)$

If ray r hit the light

Return  $L_i * f_r * cosine / pdf(w_i)$

Else If ray r hit an object at q

Return  $shade(q, -w_i) * f_r * cosine / pdf(w_i)$

This is  
Path Tracing LL