



CSE251

Basics of Computer Graphics

Module: Ray Tracing

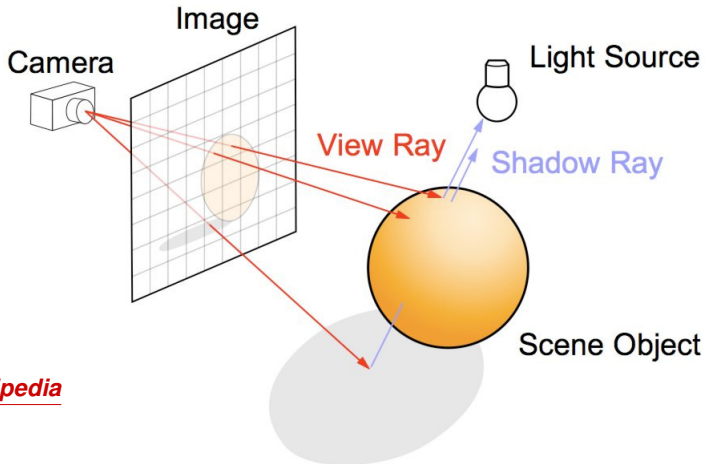
Avinash Sharma

Spring 2018

Image Generation

- ▶ Think of the image to be generated by graphics: Each pixel needs a colour. And there are $M \times N$ of them.
- ▶ Since we assume a pin-hole camera, the colour at each pixel is what the ray from the world point that falls on the pixel brings.
- ▶ Too many world points, but there are only MN pixels!
- ▶ Let us send a ray from the camera centre out through each pixel to the world. Let us see where that hits. Colour pixel accordingly!

Image Generation (cont.)



[Wikipedia](#)

Ray Tracing

- ▶ Send rays from CoP through each image pixel to the world and see what each finds
 - for each pixel in the image
 - Determine closest object in the direction of projector
 - Draw the pixel with appropriate colours
- ▶ Called **ray tracing** or **ray casting**.
- ▶ Equation of the ray is known. World objects are known. Need to intersect the ray with objects!

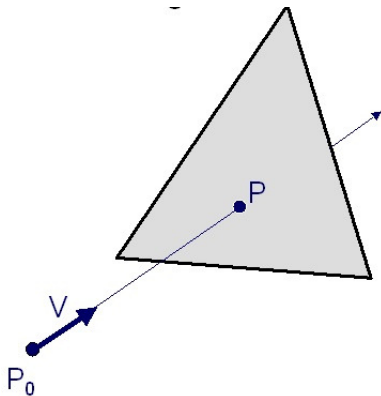
Ray Equation

- ▶ If the CoP \mathbf{P}_0 is (x_0, y_0, z_0) and pixel point \mathbf{P}_1 is (x_1, y_1, z_1) , the ray is given by $\mathbf{P} = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0)$ or

$$(x_0 + t\Delta x, y_0 + t\Delta y, z_0 + t\Delta z), \quad t > 0$$

- ▶ Negative values of t : behind CoP.
 $t = 1$: the pixel plane.
True region of interest: $t > 1$, in front of the camera
- ▶ Compute intersections with other objects. Closest object is the one with the smallest t value.

Intersection with a Polygon



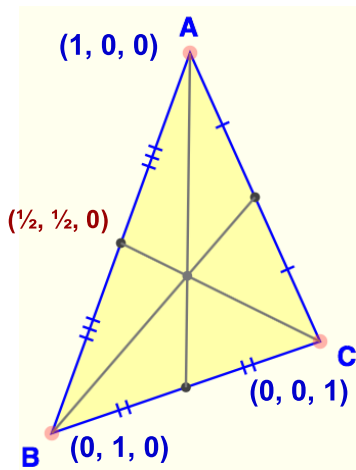
- Intersect the ray with the finite triangle: First intersect with the (infinite) plane.

Intersection with a Polygon

- ▶ Plane of the polygon is given by: $Ax + By + Cz + D = 0$
- ▶ Intersection point: $t = -\frac{Ax_0 + By_0 + Cz_0 + D}{A\Delta x + B\Delta y + C\Delta z}$
- ▶ Does it lie within the polygon?
- ▶ Project to a coordinate plane and check for 2D polygon containment.
- ▶ There are more efficient methods using **barycentric** coordinates.

Barycentric Coords on Triangles

- ▶ A point $P = \alpha A + \beta B + \gamma C$
- ▶ **Barycentric coords:** α, β, γ
- ▶ Also, $\alpha + \beta + \gamma = 1$
- ▶ Coordinates of mid-points of sides? Centroid?
- ▶ Properties of inside of the triangle? Outside?
- ▶ α, β, γ given **A, B, C** and **P**



Computing Barycentric Coords

- ▶ $\mathbf{P} = \alpha \mathbf{A} + \beta \mathbf{B} + (1 - \alpha - \beta) \mathbf{C}$
- ▶ $\begin{bmatrix} x - x_3 \\ y - y_3 \end{bmatrix} = \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \mathbf{T} \boldsymbol{\rho}$
- ▶ Barycentric coordinates: $\boldsymbol{\rho} = \mathbf{T}^{-1} (\mathbf{P} - \mathbf{C})$
- ▶ \mathbf{T} is invertible as $\mathbf{A}, \mathbf{B}, \mathbf{C}$ form a triangle. (Determinant of \mathbf{T} is related to area of the triangle ABC, which can not be zero, unless in case of degenerated scenarios).
 \mathbf{T} is fixed for the triangle and is precomputed.
- ▶ Works for point \mathbf{P} anywhere on the plane.
Evaluate $\boldsymbol{\rho}$ for the intersection point.
If any of $\alpha, \beta, \gamma \notin [0, 1]$, \mathbf{P} is outside triangle

An Example

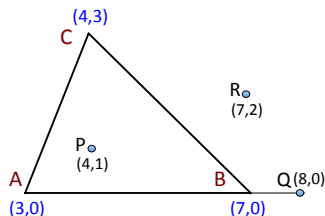
- Barycentric coords of P, Q, R?

- $\mathbf{T} : \begin{bmatrix} -1 & 3 \\ -3 & -3 \end{bmatrix}, \quad \mathbf{T}^{-1} : \frac{1}{12} \begin{bmatrix} -3 & -3 \\ 3 & -1 \end{bmatrix}$

- $\mathbf{P} : \mathbf{T}^{-1} \begin{bmatrix} 0 \\ -2 \end{bmatrix} \equiv (?, ?, ?)$

- $\mathbf{Q} : \mathbf{T}^{-1} \begin{bmatrix} 4 \\ -3 \end{bmatrix} \equiv (?, ?, ?)$

- $\mathbf{R} : \mathbf{T}^{-1} \begin{bmatrix} 3 \\ -1 \end{bmatrix} \equiv (?, ?, ?)$



An Example

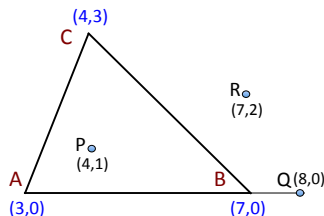
► Barycentric coords of P, Q, R?

► $\mathbf{T} : \begin{bmatrix} -1 & 3 \\ -3 & -3 \end{bmatrix}, \quad \mathbf{T}^{-1} : \frac{1}{12} \begin{bmatrix} -3 & -3 \\ 3 & -1 \end{bmatrix}$

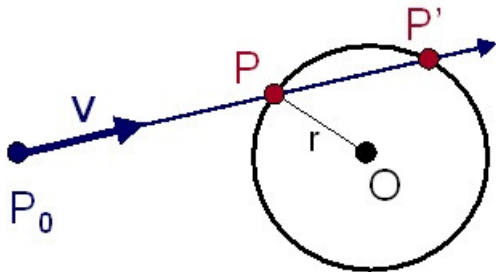
► $\mathbf{P} : \mathbf{T}^{-1} \begin{bmatrix} 0 \\ -2 \end{bmatrix} \equiv (\frac{1}{2}, \frac{1}{6}, \frac{1}{3})$

► $\mathbf{Q} : \mathbf{T}^{-1} \begin{bmatrix} 4 \\ -3 \end{bmatrix} \equiv (-\frac{1}{4}, \frac{5}{4}, 0)$

► $\mathbf{R} : \mathbf{T}^{-1} \begin{bmatrix} 3 \\ -1 \end{bmatrix} \equiv (-\frac{1}{2}, \frac{5}{6}, \frac{2}{3})$



Intersection with a Sphere



- ▶ Ray is given by $(x_0 + t\Delta x, y_0 + t\Delta y, z_0 + t\Delta z)$, $t > 0$
- ▶ Sphere is given by $(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2$.

Intersection with a Sphere

- ▶ Substituting ray into sphere: $(\Delta x^2 + \Delta y^2 + \Delta z^2) t^2 + 2[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] t + (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$
- ▶ A quadratic equation. Solve for t . Two real solutions or two imaginary solutions.
- ▶ Real solution with smaller positive t is the one of interest. When are both equal??
- ▶ If both imaginary, no intersection.
- ▶ Can normalize such that the coefficient of t^2 is 1, since we are interested only in the relative values of t .

Intersection with Other Primitives

- ▶ Need an analytical method to intersect a primitive
- ▶ Possible to compute **exact intersections** with objects defined analytically (no approximations)
- ▶ Intersections are not known or easy! Higher polynomial surfaces? Sinusoids? Others? Parametric surfaces?
- ▶ Result: Quadratic, Cubic, and Quartic polynomials can be solved analytically. Beyond that **only** iterative solutions!
- ▶ Otherwise: First **tessellate** or subdivide to triangles and then use the triangle algorithm.

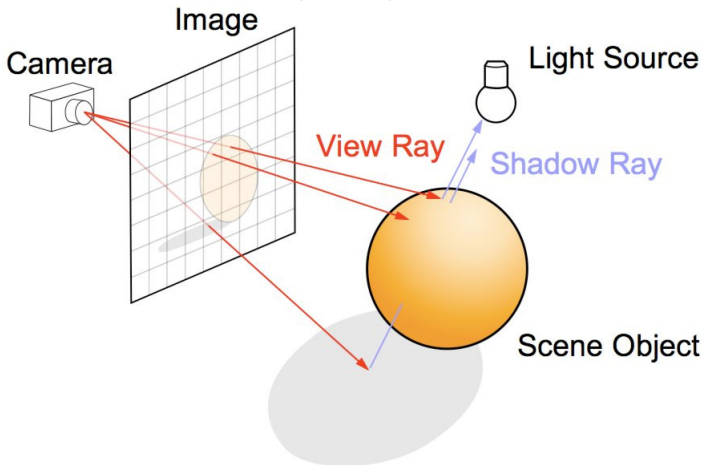
Scene with Multiple Primitives

- ▶ We now know how to intersect with **a single** primitive. Any typical scene has many primitives.
- ▶ Naive approach:
 - ▶ Check intersections with each primitive in a loop
 - ▶ Remember the primitive for which the t value is minimum positive.
- ▶ Very inefficient as each ray has to loop over each primitive!
- ▶ *We will be come back to this issue later!*

Can we do more?

- ▶ Let us look at the picture again

Can we do more? (cont.)



- We see: **light source, shadow, refractions!**

3D Scene with Recursive Ray Tracing



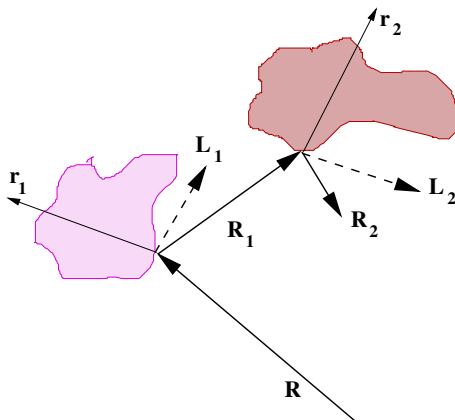
Wikipedia

Recursive Ray Tracing

- ▶ When a **primary ray** from CoP hits an object, it can
 - ▶ Reflect off the surface about the normal
 - ▶ Transmit into the object as per Snell's law of refraction
 - ▶ Collect light from all light sources by diffuse reflections.
Or be shadowed from one more light sources.
- ▶ These **secondary rays** can bring in a colour/intensity by recursively applying the above principle.
 - ▶ As the rays from a pixel ricochet (rebound, bound or skip off) through the scene, each successively interested surface is added to a binary **ray-tracing tree**.

Recursive Ray Tracing (cont.)

- ▶ Net appearance is a combination of the individual colours.



- ▶ New rays may be spawned from every point, and handled the same way!

RRTracing: Main Algorithm

- ▶ Recursive ray tracing stops when:
 - ▶ The ray intersect no surface.
 - ▶ The ray intersect a light source that is not a reflective surface.
 - ▶ The tree has been generated to it's maximum allowable depth.
- ▶ Call the recursive ray tracing routine for every pixel to compute its colour.

for each scan-line do

 for each pixel in scan line do

 determine the ray for the pixel

 pixelColour \leftarrow RT_Trace(ray, 1)

Procedure RayTrace

- ▶ Intersect ray with closest object and compute colour using a shading routine

RT_Trace(ray, depth)

Find the closest object for the ray

if (object found)

 compute normal at intersection point

 return RT_Shade(obj, ray, intersect, normal, dpth)

else

 return BackgroundColour

Procedure RayShade

- Combine all effects to compute colour

```
RT_Shade(obj, ray, pnt, n, d)
  clr = ambient term
  for each light L do
    IRay = ray to light from pnt
    if IRay is blocked, shadow
      Compute the light reaching pnt from L
    else
      clr +=  $k_d$  * diffuse component due to L
  if (d >= maxDepth)
    return clr
  Onto recursive processing now
```

Procedure RayShade (cont.)

```
if (object is reflective)
    rRay = reflected ray from pnt
    rClr = RT_Trace(rRay, d + 1)
    clr +=  $k_s$  * rClr
if (object is transparent)
    tRay = refracted ray from pnt
    if (no total internal reflection)
        tClr = RT_Trace(tRay, d + 1)
        clr +=  $k_t$  * tClr
return clr
```


When do we Stop?

- ▶ Truth: *Recursion never stops in the real world!*
- ▶ However, the impact on original pixel due to a later *bounce* will diminish with the **depth** of the ray.
- ▶ Stop when further impact is negligible.
- ▶ In practice: Set a maximum limit on the depth.
- ▶ Add a step to `RT_Trace`:
if (depth > MAX_DEPTH) return 0;

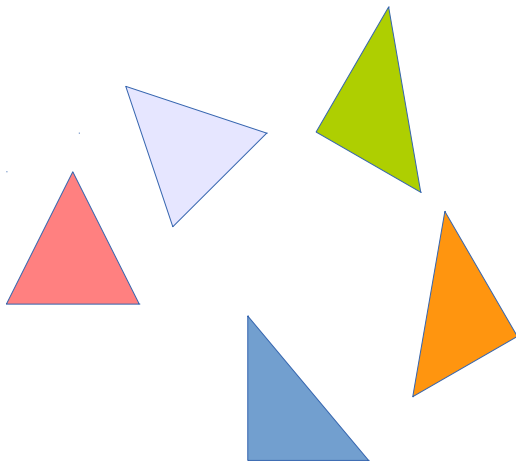
Acceleration Structure

- ▶ Use an **acceleration data structure** to quickly identify the subset of primitives the ray may intersect.
Or, eliminate primitives that the ***ray will not intersect***
- ▶ Shouldn't miss intersections, but may have a few extra.
- ▶ Procedure for each ray:
 - ▶ Traverse structure, eliminate primitives with no intersection
 - ▶ Recurse, remember the one with the minimum t
- ▶ Efficiency depends on the acceleration structure used.
- ▶ Popular spatial structures: Grids, Octree, kd-Tree, Bounding Volume Hierarchy (BVH), ...

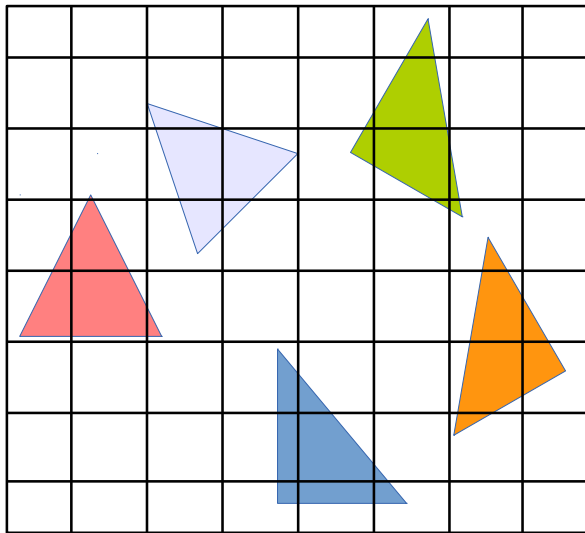
Identifying Primitives that Intersect

- ▶ Which primitive(s) does a ray intersect on its way?
- ▶ Organize primitives using a suitable data structure so that all primitives needn't be tested for each ray

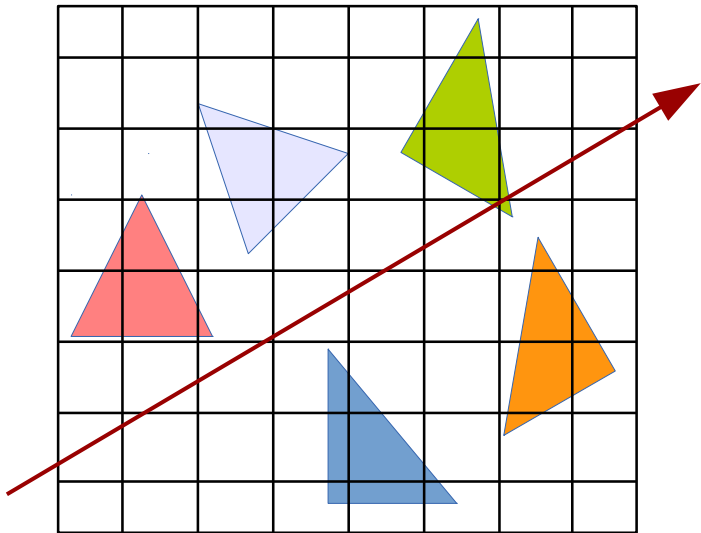
A Few Triangles



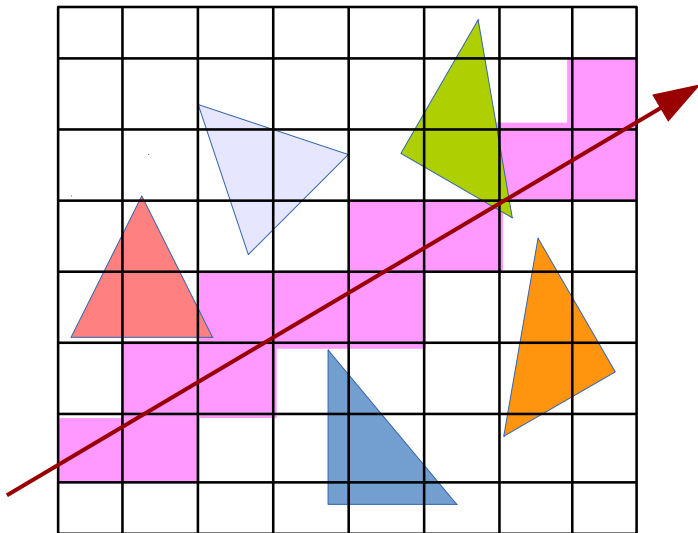
Triangles in a Grid



Triangles, Grid, and a Ray



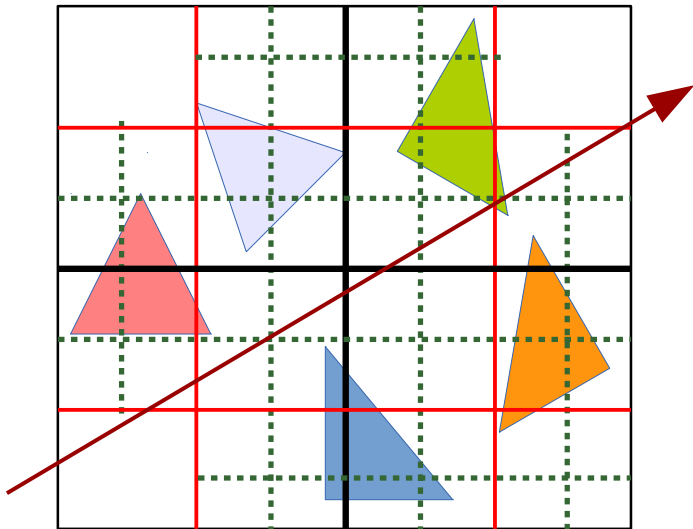
Evaluated Grid Cells



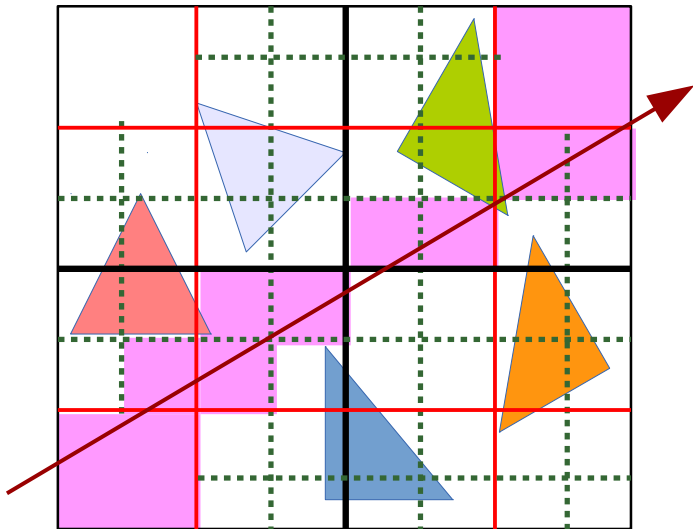
Teapot in a Stadium Situation

- ▶ Grids work well if distribution of primitives in the grid cells is nearly uniform
- ▶ What if there is a detailed **teapot** in a sparse **stadium**?
- ▶ Most parts of the stadium is empty, with triangles in a only a few cells
- ▶ The cell with the teapot has a large number of triangles and will take **a lot of time** to evaluate!
- ▶ Answer: non-uniform grid cells! Different ways to do this

Triangles and a Quad-Tree



Evaluated Quad-Tree Nodes



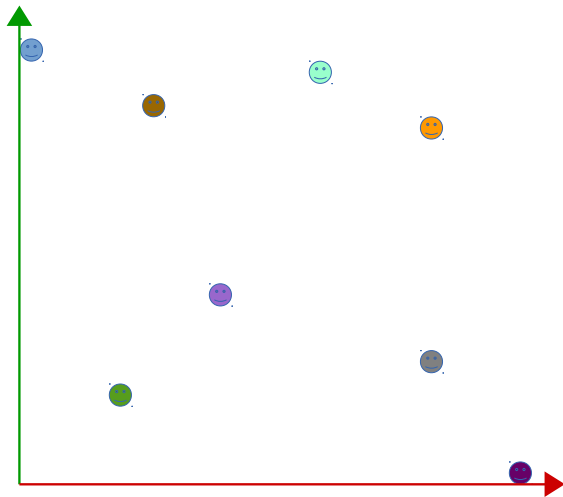
KD-Trees

- ▶ Grids, Quadtrees or Octrees, etc., divide the space using fixed planes, unmindful of the objects in it
- ▶ Objects may be split if the planes pass through them
- ▶ Can we minimize splits and be sensitive to the objects in the volume?
- ▶ Adaptive dividing plane selection: KD-Trees

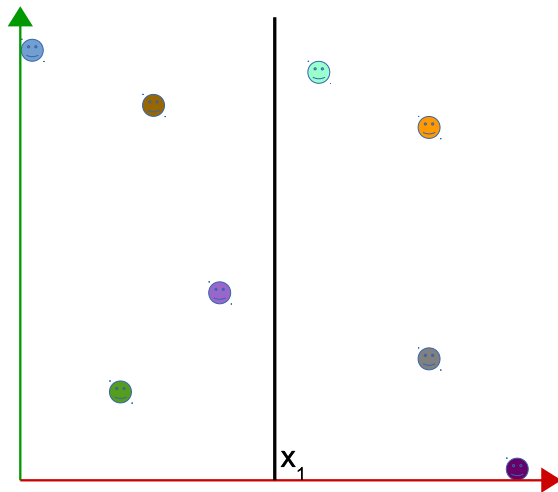
KD-Tree: Points



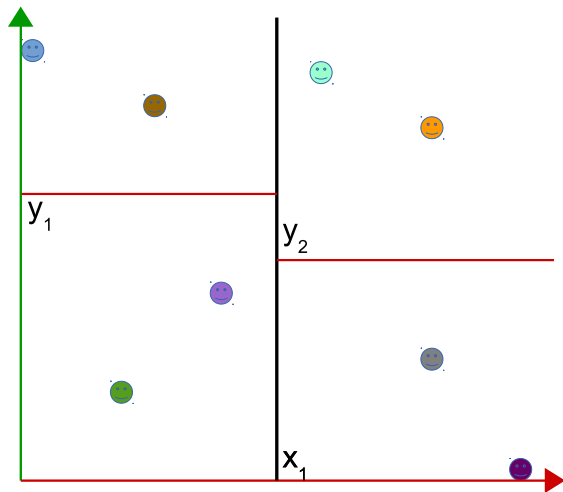
KD-Tree: Axes



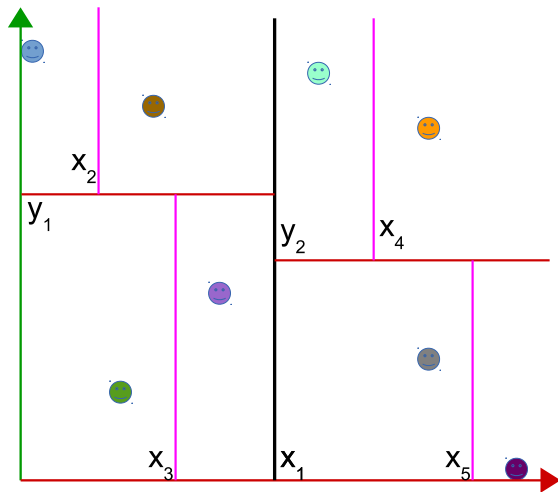
KD-Tree: First Division



KD-Tree: Second



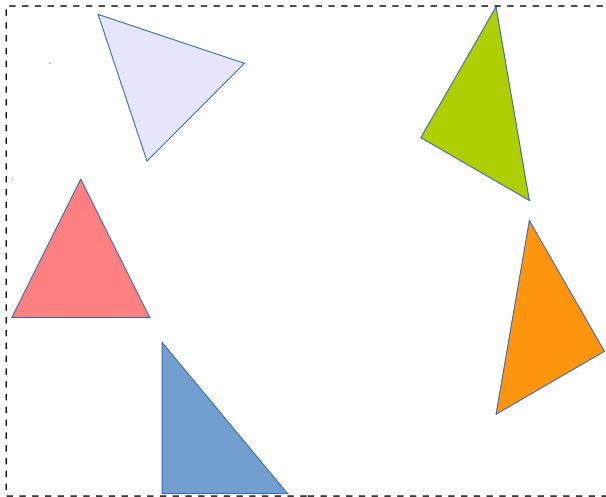
KD-Tree: Third



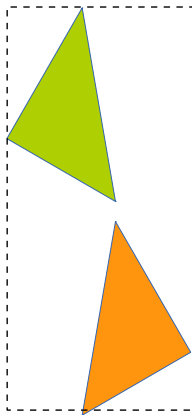
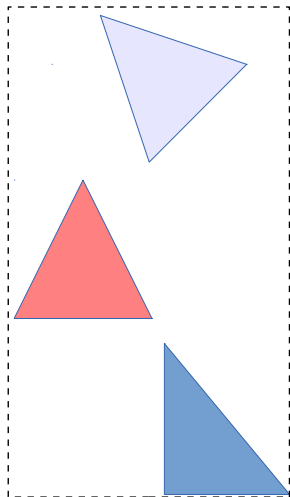
BVH Trees

- ▶ Spatial division can divide primitives.
- ▶ Can we keep primitives intact?
- ▶ Use bounding boxes to represent it and create a hierarchy
- ▶ Overlap may occur, but quite efficient
- ▶ BVH Tree: Bounding Volume Hierarchy Trees

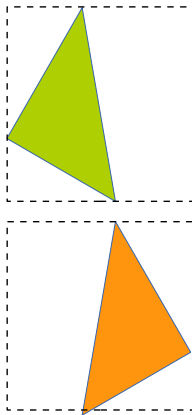
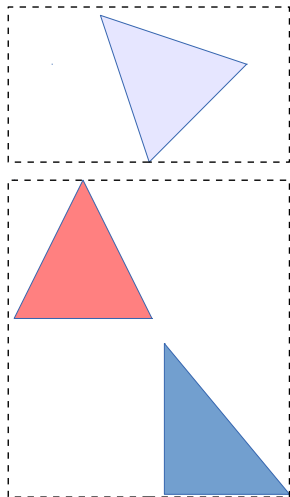
BVH Trees: Triangles



BVH Trees: Triangles



BVH Trees: Triangles



Ray Tracing: Discussion

- ▶ Ray tracing is for realistic rendering!
- ▶ Good acceleration structure critical to reduce the number of intersections computed.
- ▶ Coherence in image space enables us to trace a bunch of rays (**beams**) to be traced together.
- ▶ Very compute intensive as the ray tree can grow exponentially with spawning of new rays. Considered a **grand challenge** problem in parallel computing.
- ▶ Subject to numerical precision as small changes in secondary and tertiary rays can have large impact.
- ▶ Several simplifications: Trace a set of rays (beams, cones, pencils) to take advantage of coherence, stochastic sampling, etc.

Ray Tracing: Discussion (cont.)

- ▶ Used when really high quality rendered images are required at the expense of time.
- ▶ Powerful multicore CPUs and high-performance Graphics Processor Units (GPUs) have made it considerably fast today.

Thank you!