Computer Graphics

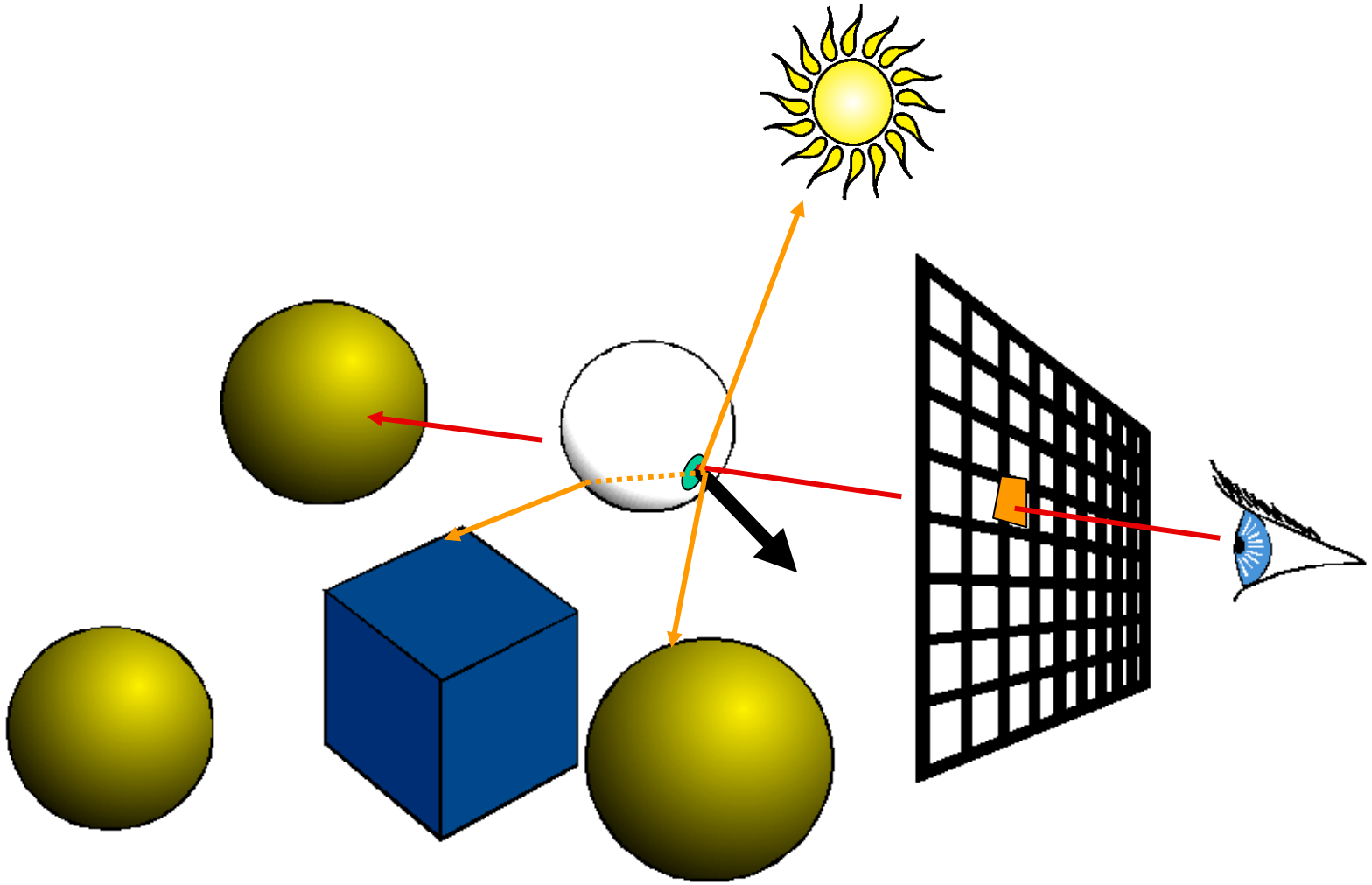
Shi-Min Hu

Tsinghua University

Ray Tracing Acceleration(光线跟踪加速)

- **Ray Tracing**
  - As we mentioned in previous course, ray tracing is one of the most important techniques in computer graphics. Using ray tracing techniques, we could generate impressive images including a lot of visual effects, such as hard/soft shadows, transparence(透明), translucence(半透明), reflection, refraction, texture and so on.

Recursive Ray Tracing

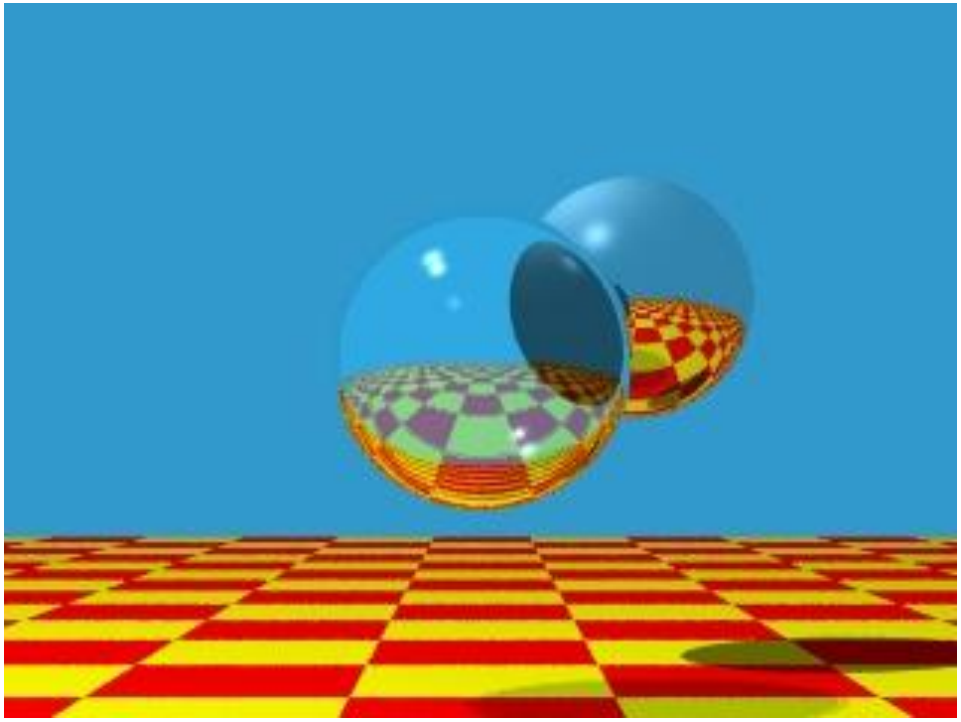


Recursive Ray Tracing

```
IntersectColor( vBeginPoint, vDirection)
{
   Determine IntersectPoint;
   Color = ambient color;
   for each light
        Color += local shading term;
   if(surface is reflective)
      color += reflect Coefficient *
           IntersectColor(IntersecPoint, Reflect Ray);
   else if ( surface is refractive)
     color +=  refract Coefficient *
           IntersectColor(IntersecPoint, Refract Ray);

   return color;
}
```

- Effects of ray tracing



Ray Tracing Acceleration

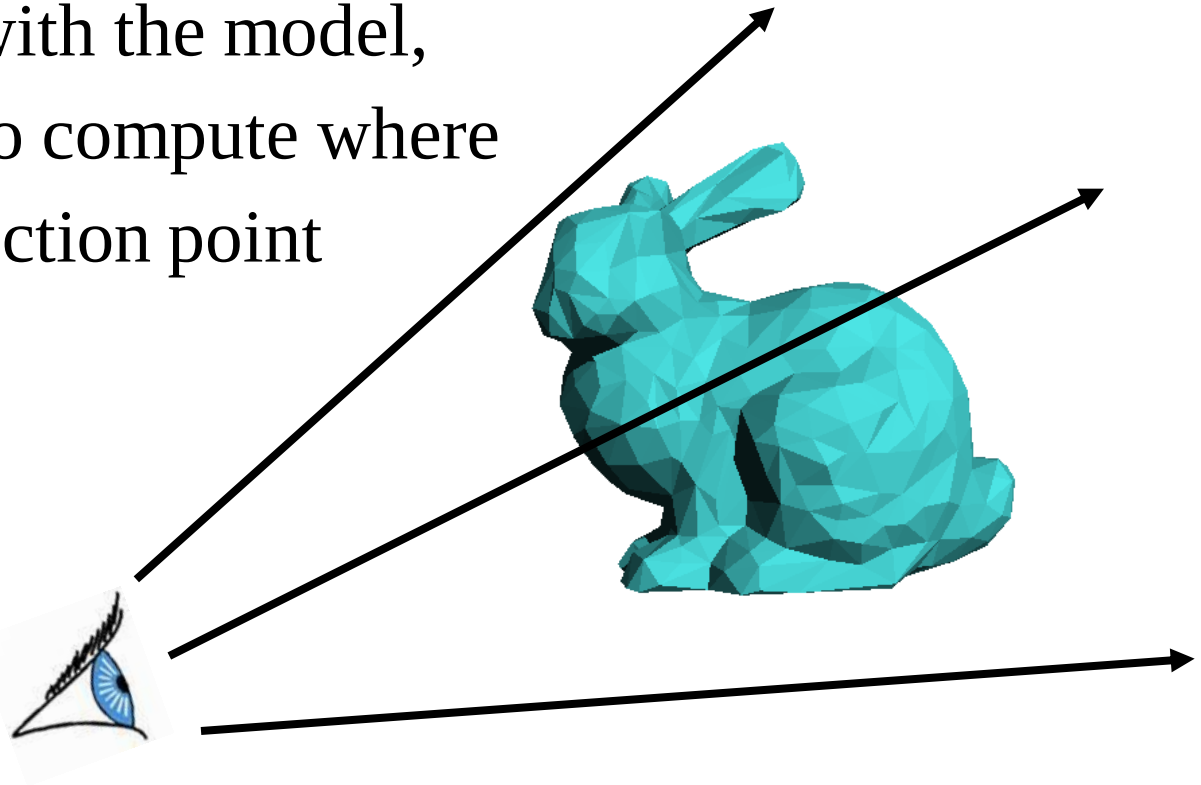
- **Motivation (Limitation of ray tracing)**
  - The common property of all ray tracing techniques is their high time and space complexity.（时空复杂性高）
  - They spend much time repeatedly for visibility computations, or doing ray object intersection testing（主要时间用于可见性计算和求交测试）

Ray Tracing Acceleration

- How to accelerate?
  - We will look at spatial(空间的) data structures
    - Hierarchical (层次性) Bounding Volumes (包围盒)
    - Uniform Grids (均匀格点)
    - Quadtree/Octree (四叉树/八叉树)
    - K-d tree / BSP tree (空间二分树)
  - Good spatial data structures could speed up ray tracing by 10-100 times

Ray Intersection

– Given a model, and a ray direction, as shown in the right figure, how to test whether the ray intersect with the model, and how to compute where the intersection point is?

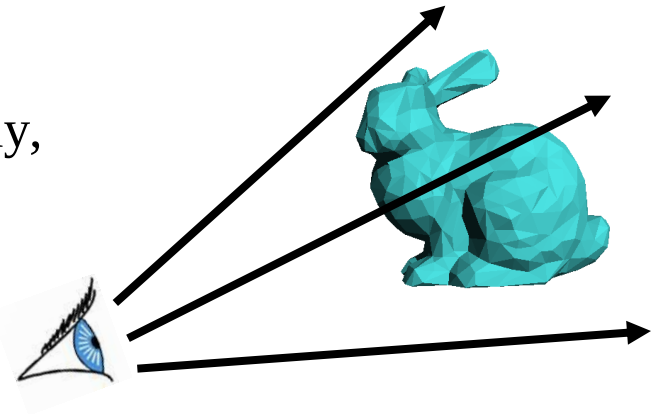


Ray Intersection

- Brute Force method

```
boolean IsIntersect(Ray r, Model m)
  For each triangle t in m
    if  IsIntersect(t,r)
        return true
  End For
  return false
```

  - However, this brute force method is costly, needs O(*n*) time complexity, *n* is the number of triangles

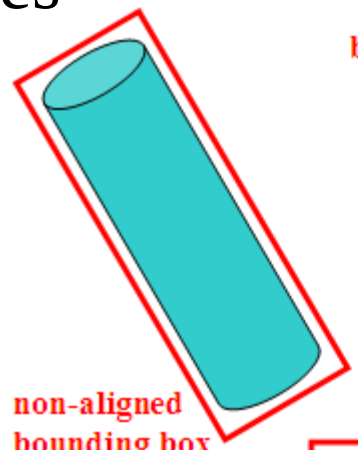


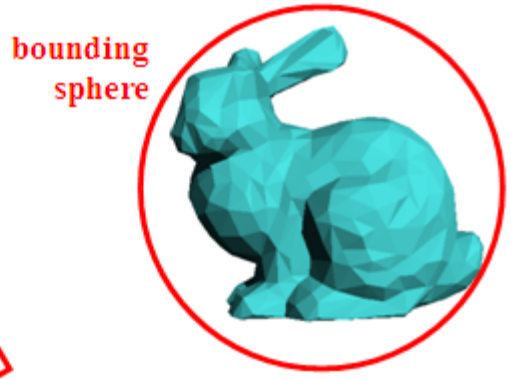
Bounding Volumes(包围盒)

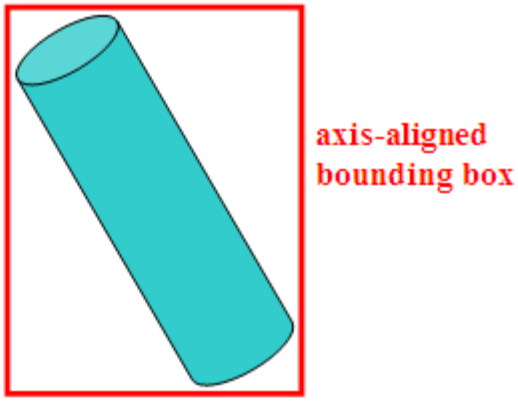
- Wrap(包住) things that are hard to test for intersection in things that are easy to test
  - Example: wrap a complicated triangle mesh in a box
  - Ray can't hit the real object unless it hits the box
  - Adds some overhead, but generally pays for itself
- Most common bounding volume types:
  - bounding box(包围盒) and bounding sphere（包围球）
  - bounding box could be axis-aligned(和坐标轴平行) or not

Bounding Volumes(包围盒)

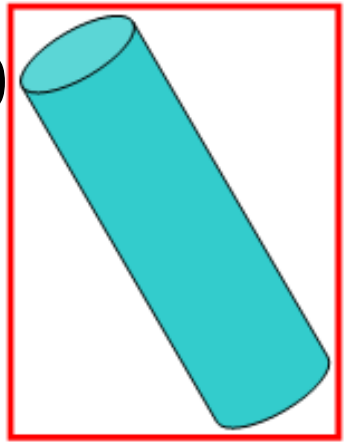
- Bounding Volume Examples
  - Before test ray/object intersect, First test the ray for an intersection with the bounding volume:
    - If the ray does not intersect with the bounding volume, the ray would not intersect with the object. (Easy reject)
    - If intersected, further test ray/object intersection





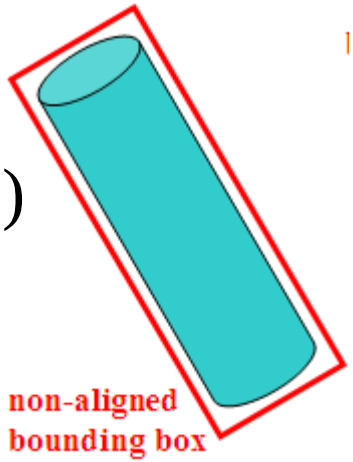


Axis-Aligned Bounding Box Creation



- Axis-Aligned bounding box(AABB)
  - the box is parallel to the x,y,z axis of the world coordinate system
- Creation
  - For each axis direction (x,y,z), calculate x_min, x_max, (y_min,y_max;z_min,z_max) of the object
  - The box (x_min,x_max,y_min,y_max,z_min, z_max) is the resulting axis-aligned bounding box

Non-Axis-Aligned Bounding Box



- Non-Axis-Aligned bounding box
  - Also called oriented bounding box (OBB)
  - an OBB is more optimal than an AABB. OBB “matches” more than AABB.
  - an optimal OBB is an approximation of the minimal bounding box, usually obtained by a complex optimization process.
- How to construct an OBB ?

Bounding Sphere

- bounding sphere
  - A bounding sphere only has two parameters: radius and center. When object rotates, bounding sphere does not need to rotate.
  - However, the ideal minimal-radius bounding sphere is hard to calculate.
  - Here, we introduce a method for finding a near optimal bounding sphere for a set of *n* points in 3D space. It is rather fast, O(*n*) time complexity. The generated sphere is about 5% larger than the ideal bounding sphere.

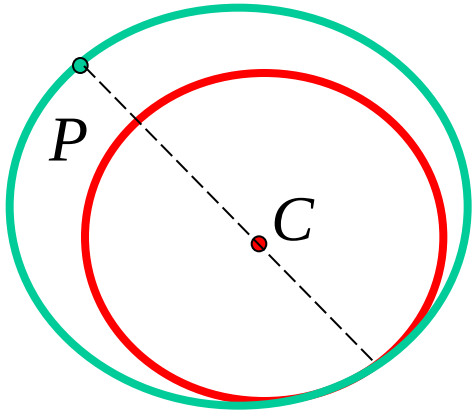
Bounding Sphere

- This algorithm is executed by 2 steps:
  - First step: Make one (quick) pass through the N points. Find these six points:
    - The point with minimum x, the point with maximum x,
      The point with minimum y, the point with maximum y,
      The point with minimum z, the point with maximum z.

    This gives three pairs of points. Each pair has the maximum span for its dimension. Pick the pair with the maximum point-to-point separation. Calculate the initial sphere, using this pair of points as a diameter.

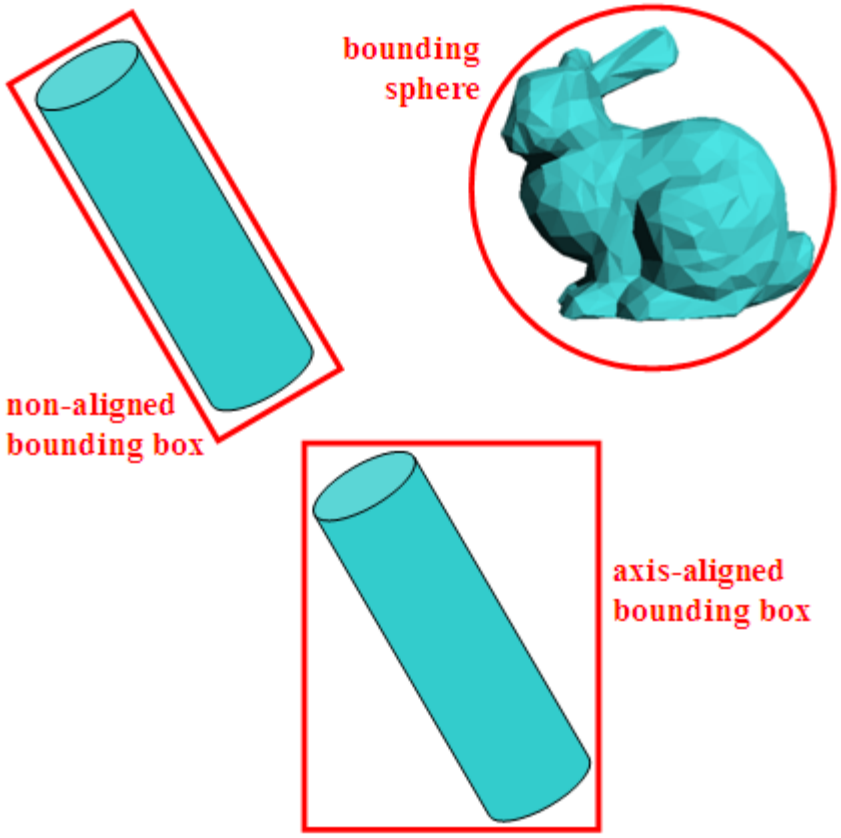
Bounding Sphere

– Make a second pass through the N points: for each point outside the current sphere, update the current sphere to the larger sphere passing through the point on one side, and the back side of the old sphere on the other side. Each new sphere will (barely) contain the old sphere, and the new point. As shown in the figure, old sphere (the red sphere)
is enlarged to the new sphere (The green sphere).



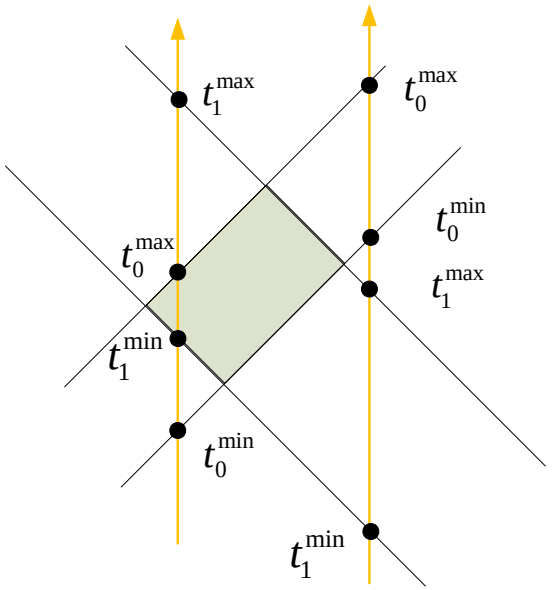
Bounding Volumes(包围盒)

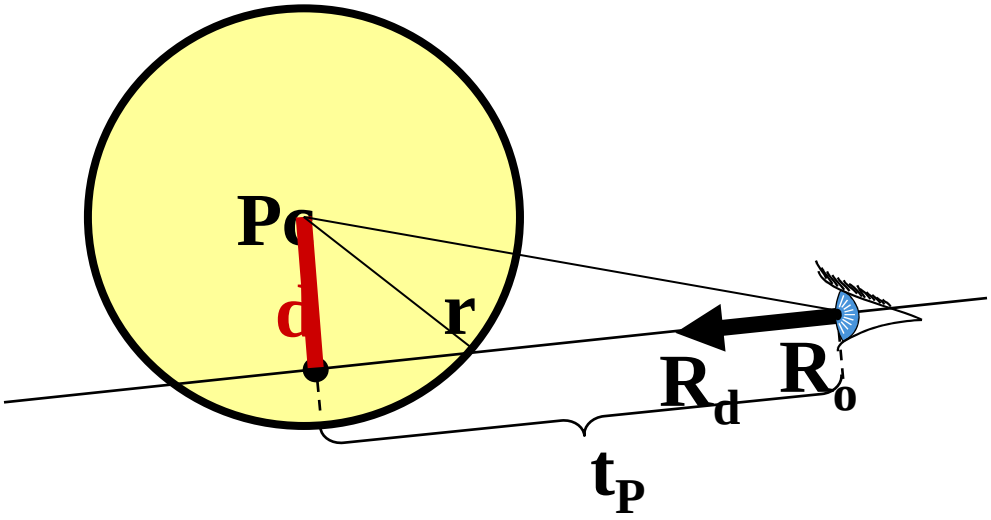
- Bounding Volumes
  - Besides acceleration for ray tracing. Bounding volumes could be used for other motivations, such as Hidden Surface Removal, Collision(碰撞) Detection



Bounding Volumes(包围盒)

- Ray Intersection with Bounding Volumes
  - Including Ray/Box intersection and Ray/Sphere Intersection. Both have been discussed in chapter 4 “Ray Tracing”.





Hierarchical Bounding Volume

- Hierarchical Bounding Volume (HBV:层次包围盒)
  - Limitation of bounding volume
    - Still need to test ray against every object, O(*n*) time complexity, where *n* is the number of objects.
  - A natural extension to bounding volumes is a hierarchical bounding volume (HBV).

- Given the bounding volumes of the objects in the scene, a tree data structure of bounding volumes is created with the bounding volumes of the objects at the leaves.
- Each interior node *v* of HBV corresponds to the bounding volumes that completely encloses the bounding volumes of all the children nodes of *v*.

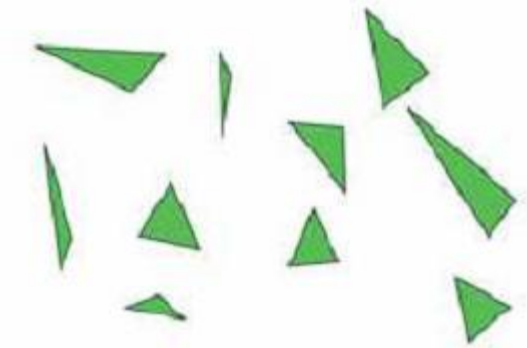
Hierarchical Bounding Volume

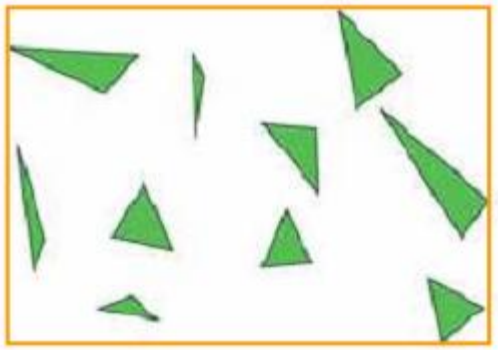
- Hierarchical Bounding Volume (HBV)
  - The method for testing a ray with the objects in the scene using HBV:
    - First test whether the ray intersect with the bounding volume at the root node. If not intersected, the ray cannot intersect any object.
    - Otherwise, recursively test the hierarchy for those nodes of HBV whose bounding volumes are intersected by the ray.

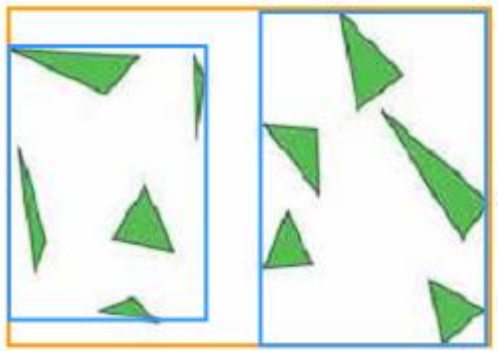
- An interesting property of HBV is: although a bounding volume of a node always completely includes its children bounding volumes, these children bounding volumes could probably intersect with each other.

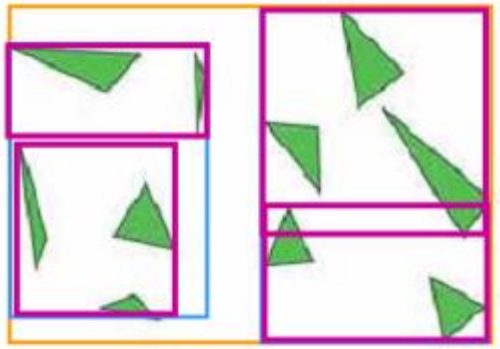
Hierarchical Bounding Volume

- An example
  - Several triangles (right figure)
  - Hierarchical bounding box (figure below)



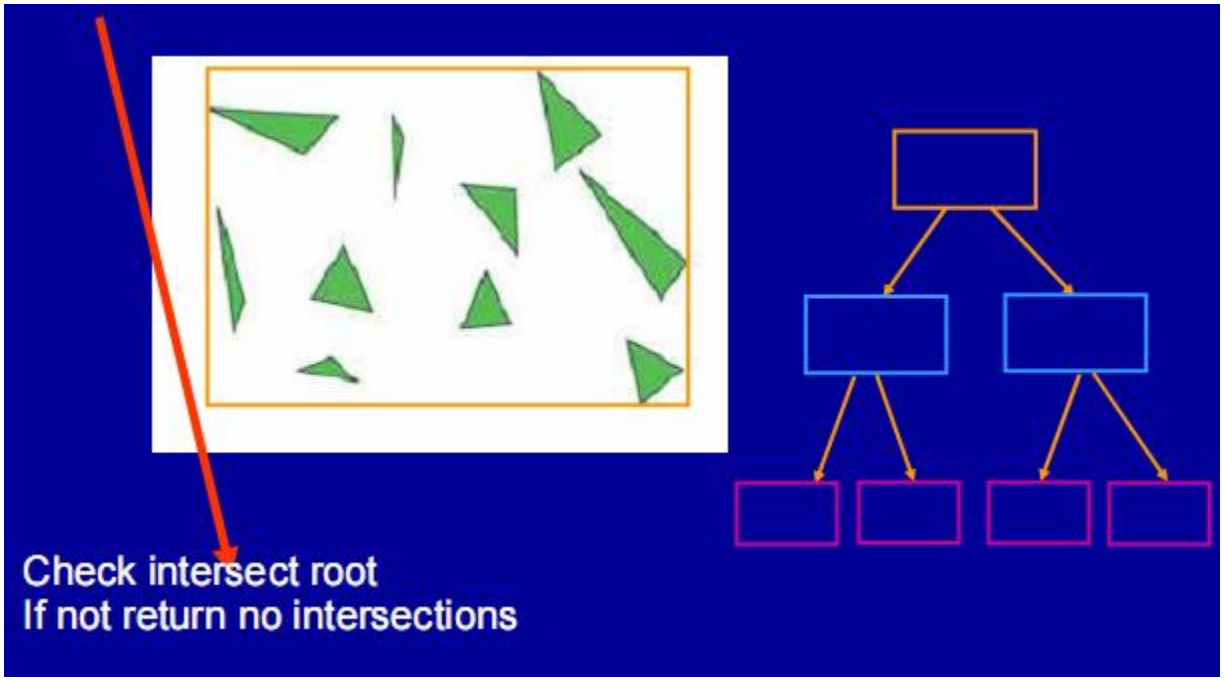






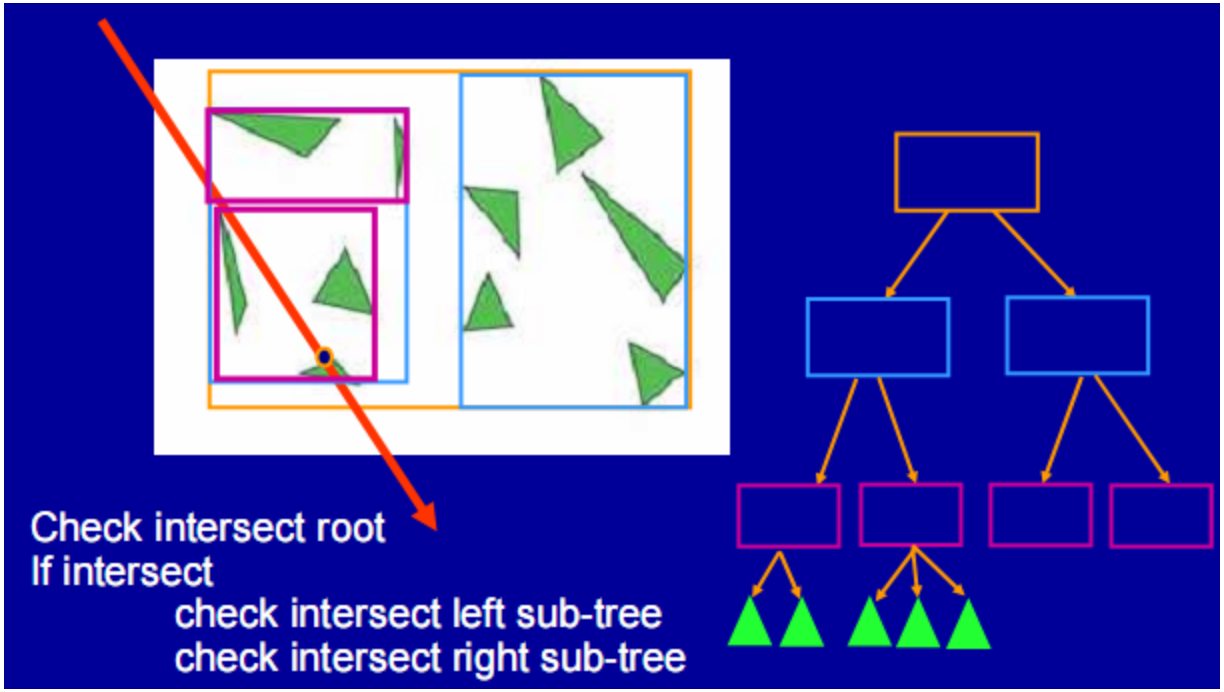
Hierarchical Bounding Volume

- An example



Hierarchical Bounding Volume

- An example

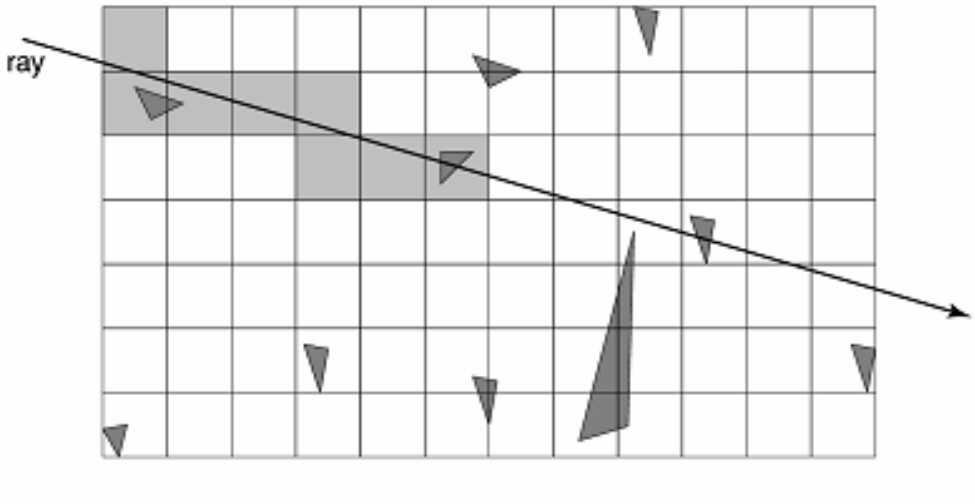


Hierarchical Bounding Volume

- Hierarchical Bounding Volume (HBV)
  - Works well if you use good (appropriate) bounding volume and hierarchy
  - Should give O(log*n*) time complexity rather than O(*n*) complexity (*n* = number of objects)
  - Can have multiple classes (box, sphere) of bounding volumes and pick the best appropriate one for each enclosed object（混用不同的包围盒）

Uniform Grids(均匀格点)

- Data Structure: a 3D array of cells that tile space
  - Each cell lists all primitives which intersect with that cell（每一个格点有所含面片的索引）

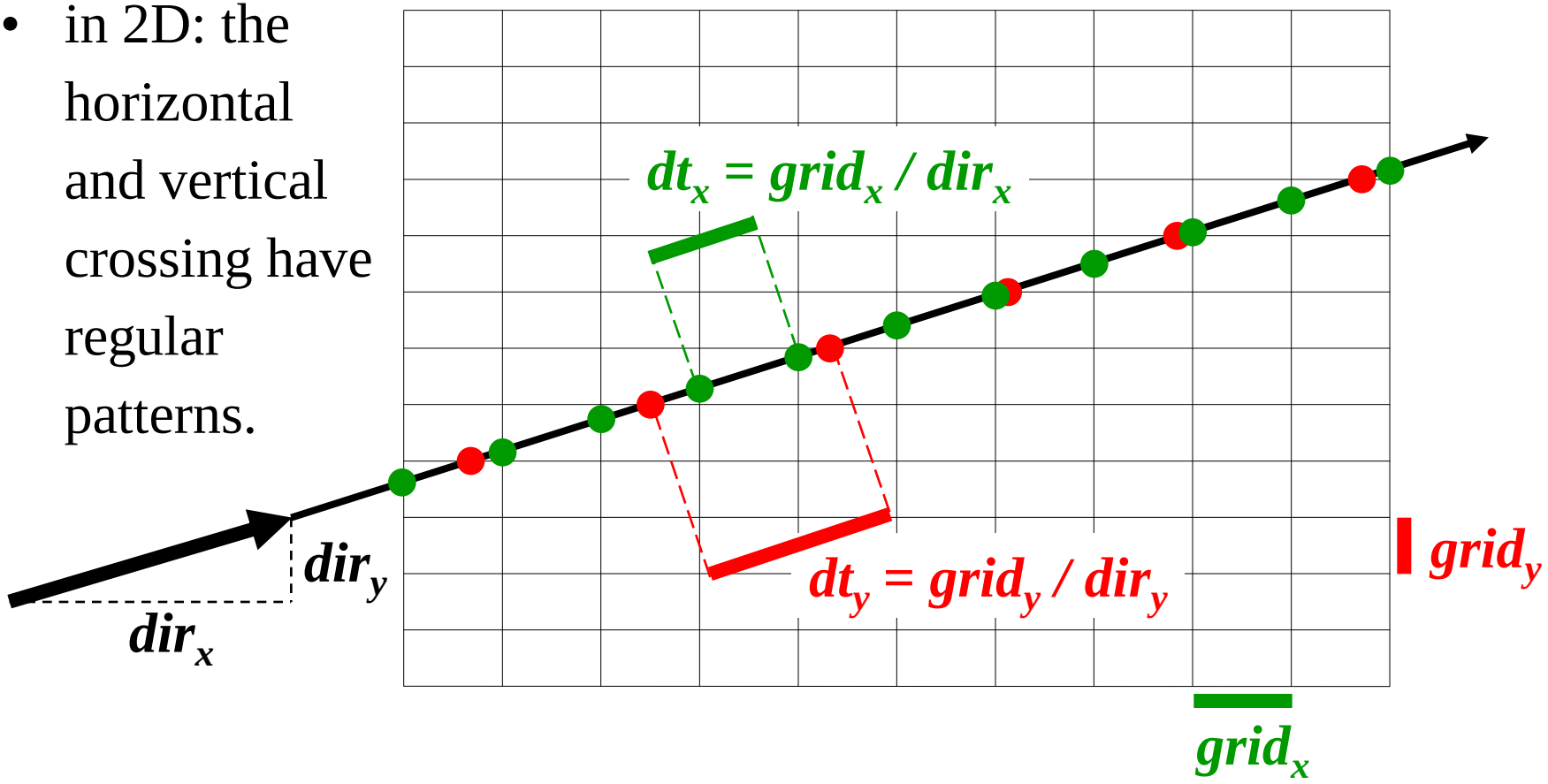


Uniform Grids(均匀格点)

- Intersection testing:
  - Start tracing at cell where ray begins
  - Step from cell to cell, searching for the first intersection cell
  - At each cell, test for intersection with all primitives intersected with that cell
  - If there is an intersection, return the closest one

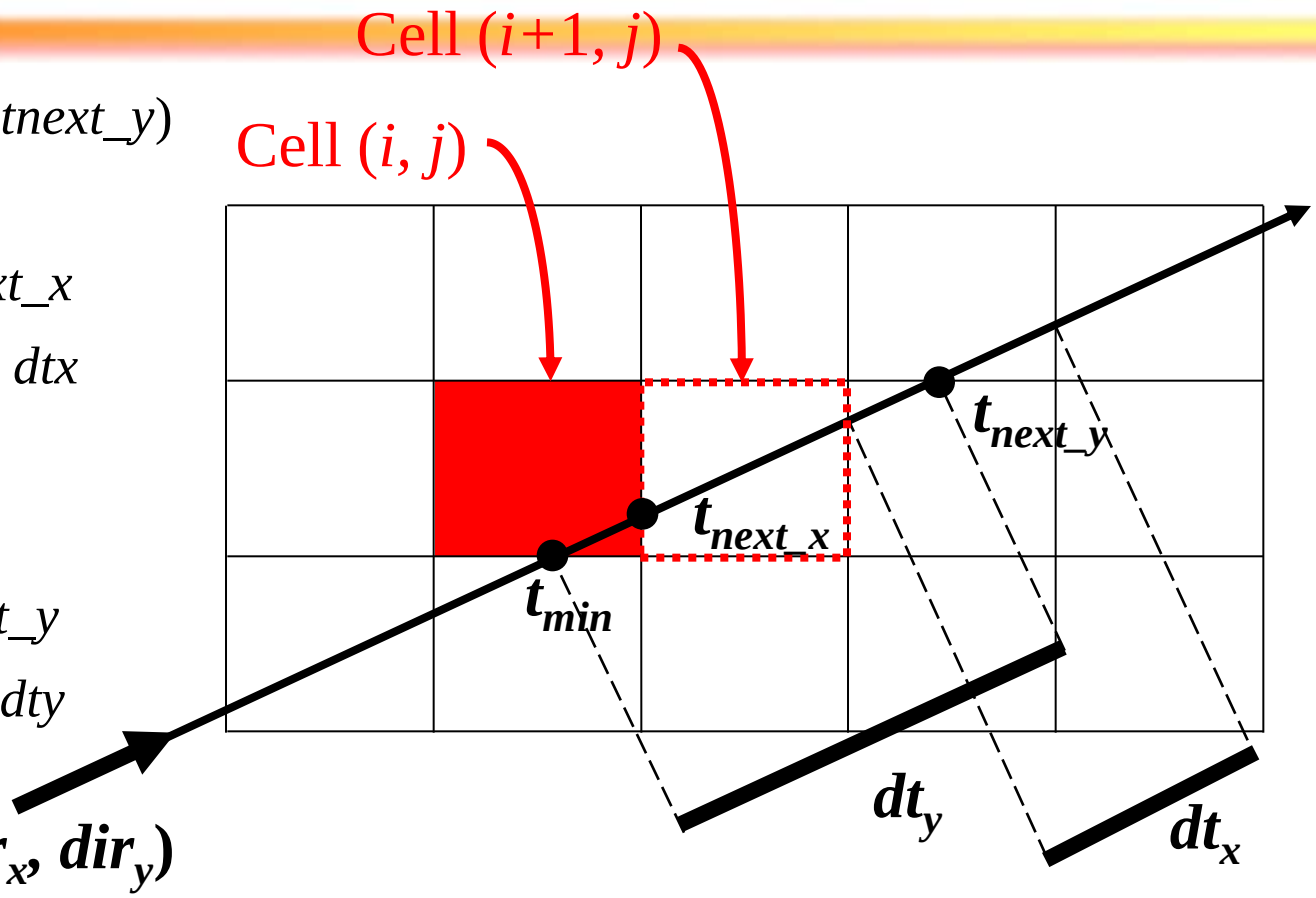
Uniform Grids(均匀格点)

- How to traverse the cells?
- in 2D: the horizontal and vertical crossing have regular patterns.



What is the next cell?

```
if  ( tnext_x  < tnext_y)
    i  +=  signx
    tmin  =  tnext_x
    tnext_x  +=  dtx
else
    j  +=  signy
    tmin  =  tnext_y
    tnext_y  +=  dty
```



if $(dir_x > 0)$ $sign_x = 1$ *else* $sign_x = -1$

if $(dir_y > 0)$ $sign_y = 1$ *else* $sign_y = -1$

What is the next cell?

- In 3D, it is similar
  - suppose current cell is (i,j,k)

if $(dir_x > 0)$ $sign_x = 1$ ***else*** $sign_x = -1$

if $(dir_y > 0)$ $sign_y = 1$ ***else*** $sign_y = -1$

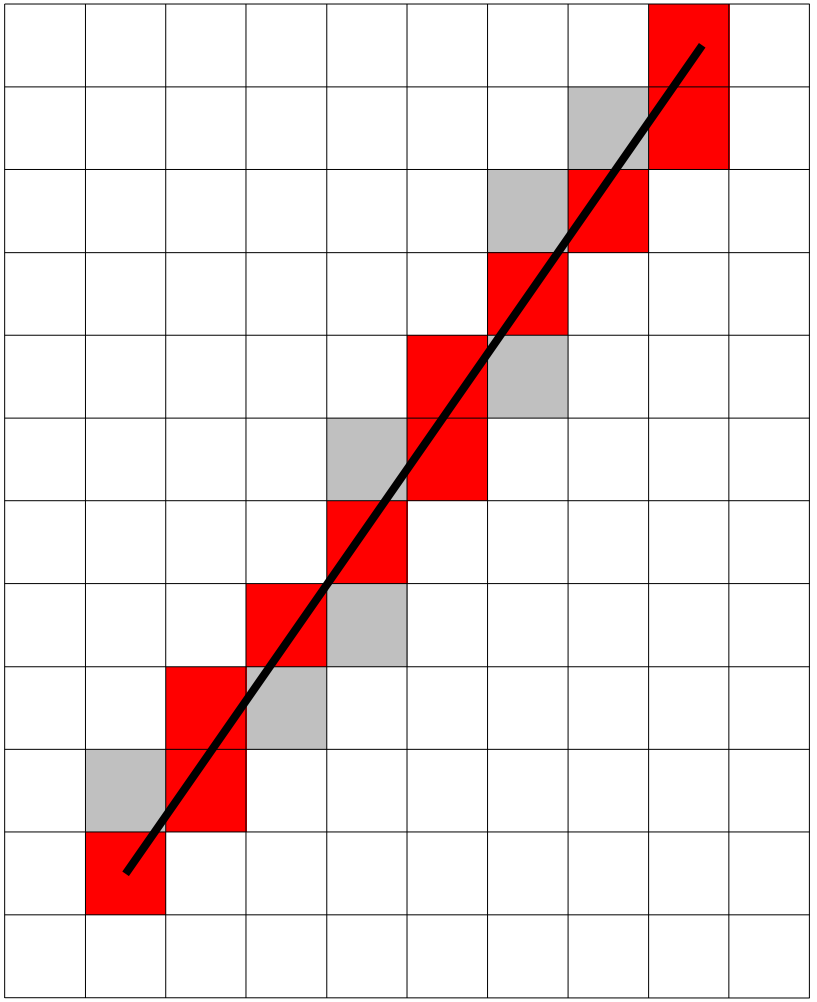
if $(dir_z > 0)$ $sign_z = 1$ ***else*** $sign_z = -1$

```
if   ( tnext_x  < tnext_y and tnext_x  < tnext_z )
     i  +=  signx; tmin  =  tnext_x; tnext_x  +=  dtx;
else if (tnext_y  < tnext_x and tnext_y  < tnext_z )
     j  +=  signy; tmin  =  tnext_y; tnext_y  +=  dty;
else // tnext_z is minimum
     k  +=  signz; tmin  =  tnext_z;  tnext_z  +=  dtz;
```

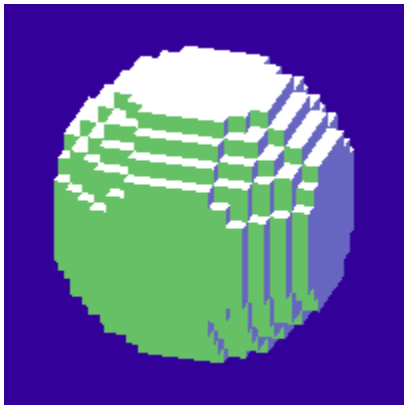
What is the next cell?

- 3DDDA – Three Dimensional Digital Difference Analyzer

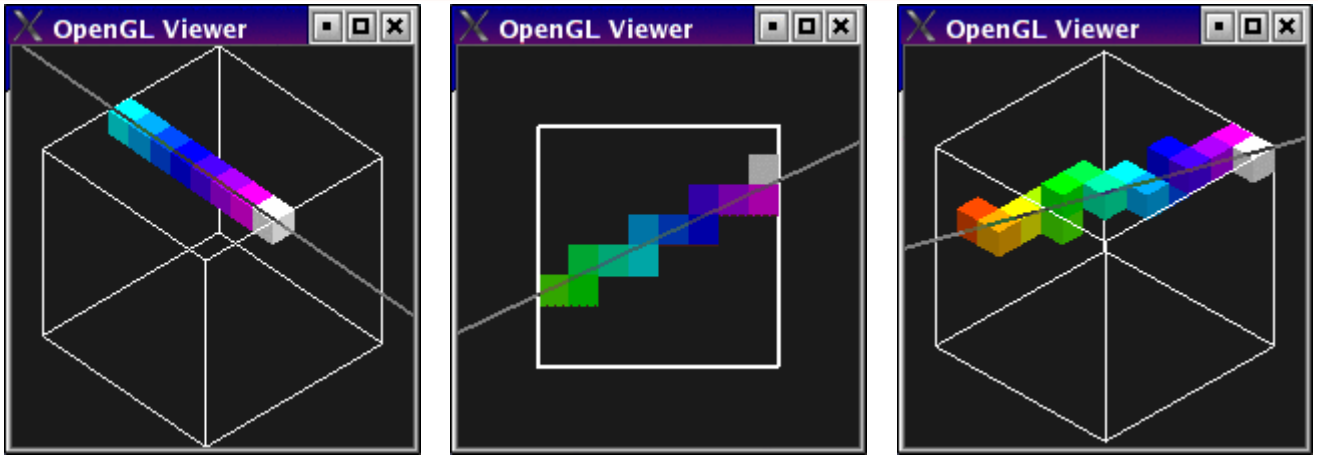
- Similar to Line Rasterization



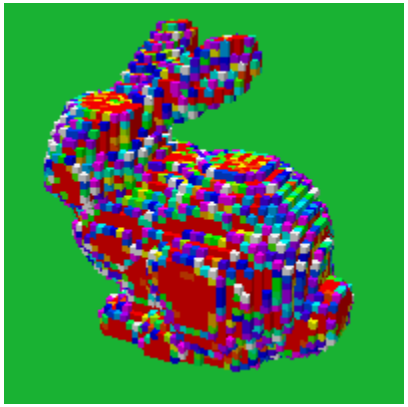
Ray Marching Visualization



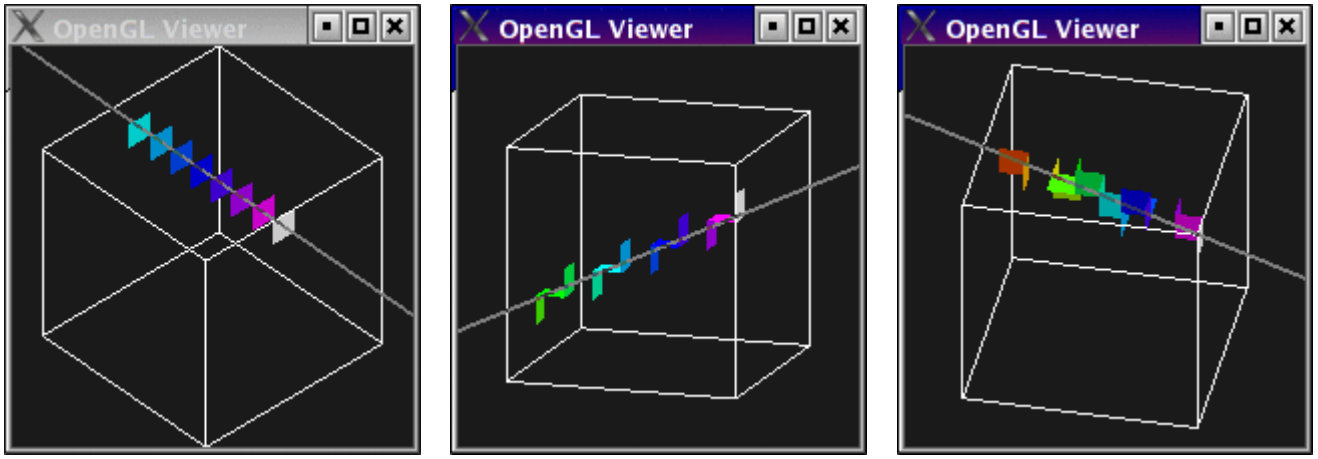
sphere voxelization



cells traversed



primitive density

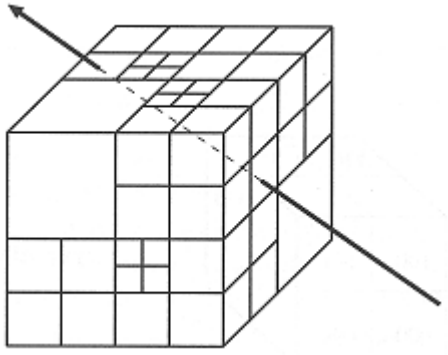


entered faces

Uniform Grids(均匀格点)

- Advantages
  - easy to construct and easy to traverse
- Disadvantages
  - Uniform grids are a poor choice if the world is non-homogeneous(不均匀). Many polygons will cluster in a small space
  - How many cells to use?
    - too few → many objects in a cell → slow
    - too many → many empty cells to traverse → slow and large storage
  - Non-uniform spatial division is better!

Octree (八叉树)



- QuadTree (四叉树)
  - The 2D generalization of binary tree (二叉树)
  - node is a square
  - recursively split a square into four equal sub-squares
  - stop when leaves get “simple enough”
- Octree
  - 3D generalization of quadtree
  - node is a cube, recursively split into 8 equal sub-cubes
  - Octree is more expensive to traverse than uniform grids, but octree adapts better to non-homogeneous scenes

Octree (八叉树)

- Construction of Octree
  - First, enclose the entire scene in a minimal axis-aligned box.
  - Built recursively in a top-down fashion, and stops recursion when a stopping criterion is fulfilled.
  - These criteria can include: a maximum number of recursion level has been reached, or that there is fewer than a threshold number of primitives in a node.（终止递归条件：已达最大层次、或包围盒内物体已很少）

Octree (八叉树)

- Construction of Octree
  - If the recursion is stopped, the algorithm binds the primitives to the node and terminates the recursion.
  - Otherwise (does not reach the depth threshold and the number of primitives bind to the node is larger than the threshold), it subdivides the node along its main three axes using three planes, thereby forming eight equal-sized sub-nodes. Each sub-node is tested and possibly subdivided again into 2*2*2 smaller ones.

Octree (八叉树)

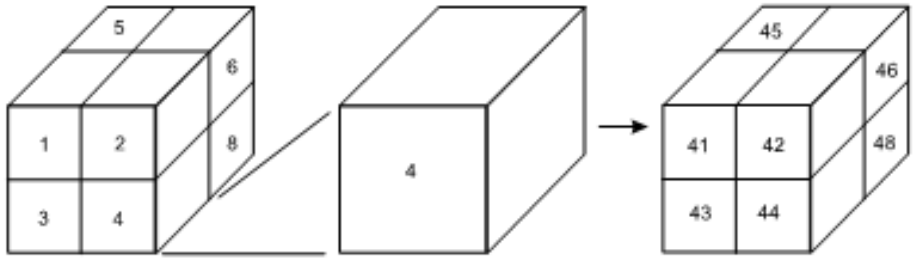
- Construction of Octree
  - Note that, primitives are always stored in leaf nodes; and therefore, certain primitives have to be stored in more than one leaf node (for example, a primitive which happens to locate at the center of the octree)
  - But this is not efficient, since a tiny object may be stored in many nodes. One solution is to split the primitives, but that inroduces more primitives.
    - A variant of octree called octree-R to solve this problem

Octree (八叉树)

- Octree-R
  - Octree-R results in arbitrary positioning of the splitting planes inside the interior nodes. The difference between the octree and octree-R is: the splitting planes of octree are always in the center, while octree-R not.
  - Using smart heuristic algorithm for positioning the splitting planes inside the octree interior node for all three axes.
  - Octree-R could speed-up octree by 4% - 47%, depending on the distribution of the objects in the scene.

Octree (八叉树)

- addressing of child nodes in octree
  - There are two methods for accessing the interior octree node
    - Direct pointing
      the interior node contains eight pointers to its descendants
    - The address of the child nodes is usually formed by postfixing or prefixing the parent address by digits from 1 to 8, as in the figure below



Octree (八叉树)

- Traversal in octree
  - Ray traversal algorithm for octree is more complicated than uniform grid, and BSP tree (we will introduce it later). Each ray intersecting an octree node can visit at most its four of eight descendants, and the computation of their order to be visited along the ray path is more involved than for BSP tree.
  - the ray traversal algorithm，could refer to a survey about it .

V. Havran. A summary of octree ray traversal algorithms. Ray Tracing News, Dec. 1999. Available from http://www.acm.org/tog/resources/RTNews/html/rtnv12n2.html

BSP (Binary Space Partition) tree

- BSP (Binary Space Partitioning) Tree
  - BSP tree is a spatial subdivision that can be used to solve a variety of geometrical problems. It was initially developed as a means of solving the hidden surface problem in computer graphics.
  - It is a higher dimensional analogy to the binary search tree.
  - BSP tree has two major types, *axis-aligned* and *polygon-aligned*.

BSP (Binary Space Partition) tree

- Polygon-aligned BSP tree
  - Choose a plane underlying the polygon as the splitting entity that subdivides the spatial region into two parts. (So the splitting plane could be in arbitrary position)
  - The scene is typically required to contain only polygons, which is too restrictive for ray tracing applications.
  - So we do not discuss this form of BSP tree in class.

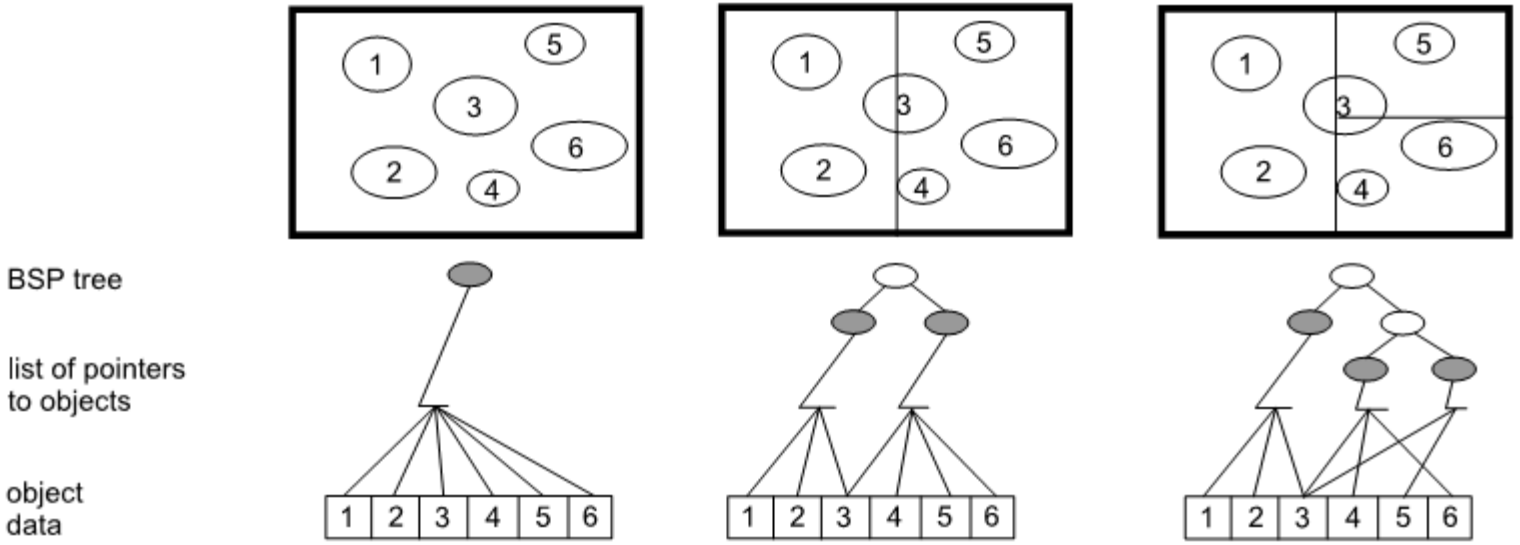
BSP (Binary Space Partition) tree

- Axis-aligned BSP tree
  - The splitting plane is always perpendicular to one of the coordinate axes. And the splitting plane always lies at the mid-point of the current node resulting in two children nodes with equal size (However, in some other literature, the axis-aligned form of BSP tree can have arbitrary positioning of the splitting planes)

- The orthogonality of splitting planes significantly simplifies the intersection test between a ray and the splitting plane. The cost of computation in intersection between a ray and the axis-aligned splitting plane is about 3 times lower than for an arbitrary positioned plane.

BSP (Binary Space Partition) tree

- An example of axis-aligned BSP tree



BSP (Binary Space Partition) tree

- Construction of axis-aligned BSP tree
  - Similar to octree, BSP tree is constructed hierarchically in top-down fashion.
    - A splitting plane (perpendicular to axis x or y or z) is selected to subdivide a current leaf node into two equal-size sub-nodes. Then the leaf becomes an interior node, with two new descendant leaves.
    - The process is repeated recursively until certain *termination criteria* are reached.
    - Similar to octree, commonly used *termination criteria* are the maximum leaf depth and the number of primitives associated with the leaf.

BSP (Binary Space Partition) tree

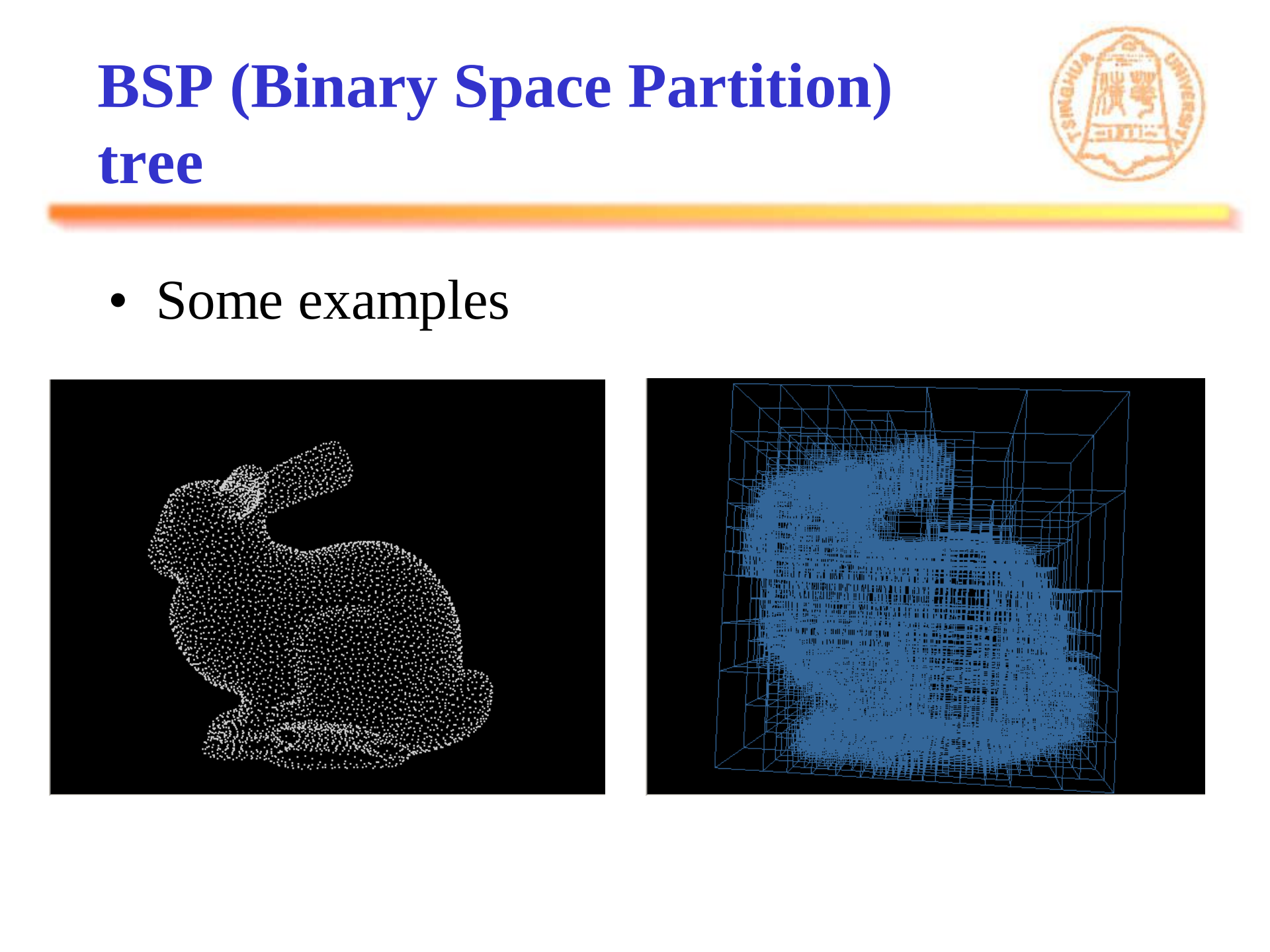
Pseudo-code for constructing a BSP-tree

```
procedure Subdivide(CurrentNode, CurrentTreeDepth, CurrentSubdividingAxis)
if ( (CurrentNode contains too many objects)
and (CurrentTreeDepth is not too high) ) then
   Children of CurrentNode ← CurrentNode's Bounding Volume
   {Note that child[0].max.DividingAxis and child[1].min.DividingAxis are always equal.}
   if CurrentSubdividingAxis = X-Axis then
      child[1].min.x ← mid-point of CurrentNode's X-Bound
      child[0].max.x ← mid-point of CurrentNode's X-Bound
      NextSubdividingAxis ← Y-Axis
   else if CurrentSubdividingAxis = Y-Axis then
      child[1].min.y ← mid-point of CurrentNode's Y-Bound
      child[0].max.y ← mid-point of CurrentNode's Y-Bound
      NextSubdividingAxis ← Z-Axis
   else if CurrentSubdividingAxis = Z-Axis then
      child[1].min.z ← mid-point of CurrentNode's Z-Bound
      child[0].max.z ← mid-point of CurrentNode's Z-Bound
      NextSubdividingAxis ← X-Axis
   end if
   for all objects referenced in CurrentNode do
      if the object is within children's bounding volume then
         add the object to the children's object list
      end if
   end for
   Subdivide(child[0], CurrentTreeDepth + 1, NextSubdividingAxis)
   Subdivide(child[1], CurrentTreeDepth + 1, NextSubdividingAxis)
end if
```

BSP (Binary Space Partition) tree

- Construction of axis-aligned BSP tree
  - As shown in the pseudo-code in last page, traditionally, the splitting plane is positioned at the mid-point of the chosen axis, and the order of axes (x,y,z) is regularly changed on successive levels of the hierarchy (for example, first x, next y, next z, next x, …). This makes the hierarchical structure more regular.

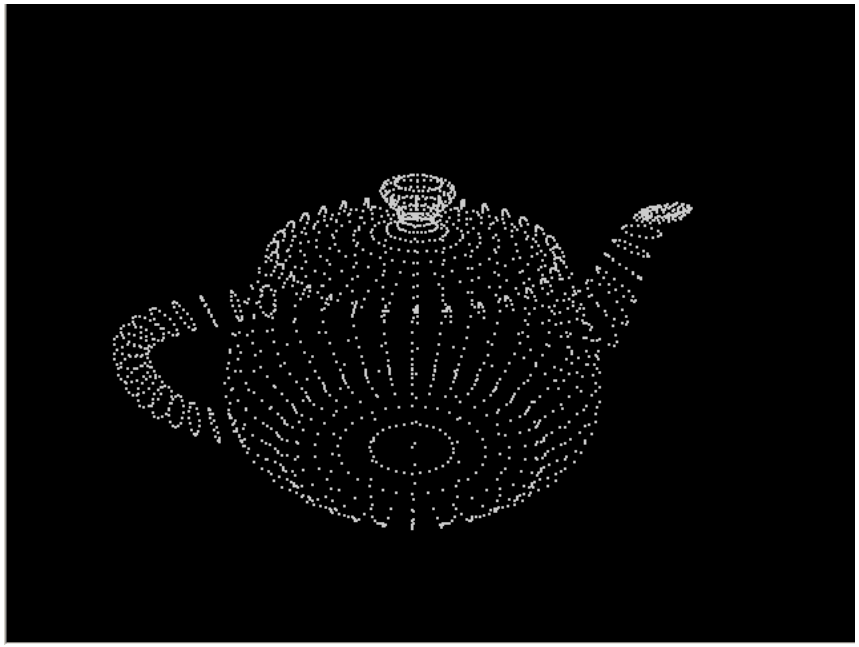
BSP (Binary Space Partition) tree

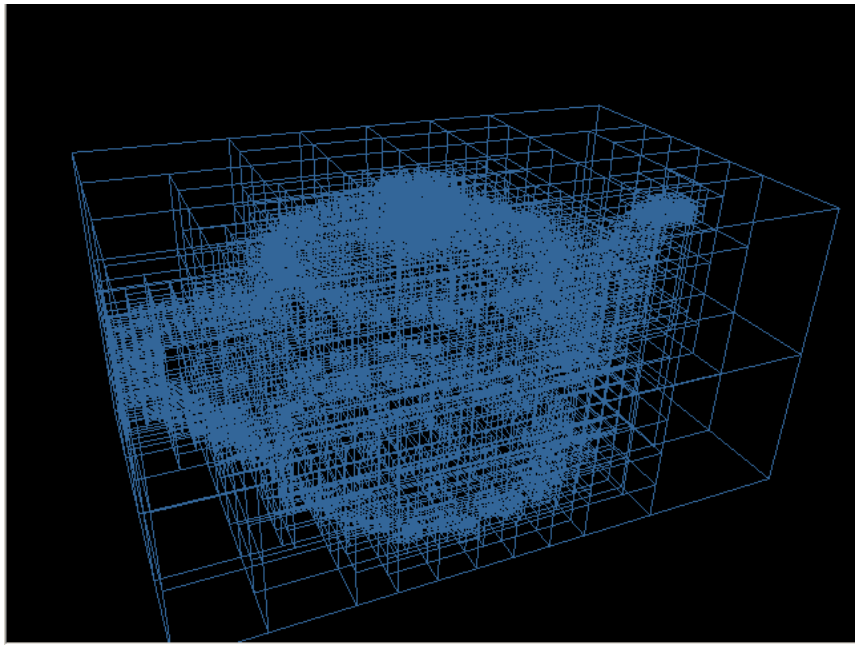


- Some examples

BSP (Binary Space Partition) tree

- Some examples





BSP (Binary Space Partition) tree

- Difference between kd-tree and BSP tree
  - Distinguished by the positioning of the splitting plane. Conceptually, BSP tree and kd-tree are equivalent. The only difference is: in BSP-tree, the splitting plane always lies at the mid-point of the current node, resulting in two children nodes with equal size; the kd-tree can have arbitrarily positioning of the splitting planes.
  - Thus, any BSP tree is a kd-tree, but not vice versa.

BSP (Binary Space Partition) tree

- Ray traversal in BSP tree

```
RayTreeIntersect (Ray, Node, min, max)
    if (Node is NIL) then return ["no intersect"]
    if (Node is a leaf) then begin
        intersect Ray with each primitive in the object link list
            discarding those farther away than "max"
        return ["object with closest intersection point"]
    end
    dist ← signed distance along Ray to the cutting plane of the Node
    near ← child of Node for half-space containing the origin of Ray
    far ← the "other" child of Node—i.e. not equal to near
    if ((dist > max) or (dist < 0)) then      /*Whole interval is on near side*/
        return [RayTreeIntersect (Ray, near, min, max)]
    else if (dist < min) then                 /*Whole interval is on far side*/
        return [RayTreeIntersect (Ray, far, min, max)]
    else begin                                /*the interual intersects the plane*/
        hit_data ← RayTreeIntersect (Ray, near, min, dist)    /*test near side*/
        if hit_data indicates that there was a hit then return [hit_data]
        return [RayTreeIntersect (Ray, near, dist, max)]      /*test far side*/
    end
```

BSP (Binary Space Partition) tree

- Ray traversal in BSP tree
  - In the pseudo-code given in the last page, recursion is used. Recursion could be avoided by maintaining an explicit stack.
  - When calling the function *RayTreeIntersect()* the first time, initial values of *min* and *max* should be distances (measured from the ray origin along the ray direction) to the two intersecting points between the ray and the bounding volume of the root of the BSP tree. Notice that if a ray originates from inside the BSP tree, then the intial value of *min* would be negative.

BSP (Binary Space Partition) tree

- Ray traversal in BSP tree
  - Generally, ray traversal in BSP tree is 10% faster than using octree.
  - Because BSP tree always split at mid-points, if an object spans multiple tree nodes, then the intersection calculation between this object and a given ray may need to be carried out many times, once in each node traversed by the ray.
    Similar to using Octree-R instead of octree, kd-tree could be used to solve this problem.

Other techniques

- Distributed Ray tracing（分布式光线跟踪）
  - also called stochastic ray tracing, is a refinement of ray tracing that allows for the rendering of "soft" phenomena.
  - Conventional ray tracing uses single rays to sample many different domains. For example, when the color of an object is calculated, ray tracing might send a single ray to each light source in the scene. This leads to sharp shadows. Conventional ray tracing also typically generate one reflection ray and one transmission ray per intersection. As a result, reflected and transmitted images are perfectly (and unrealistically) sharp.

Distributed Ray Tracing

- Distributed Ray tracing
  - It generates various kinds of effects:
    - Illumination: extended light sources, soft shadows;
    - Pixel: anti-aliasing;
    - Lens: depth-of-field;
    - BRDF: glossy-reflection;（模糊镜面反射）
    - Time: motion-blur

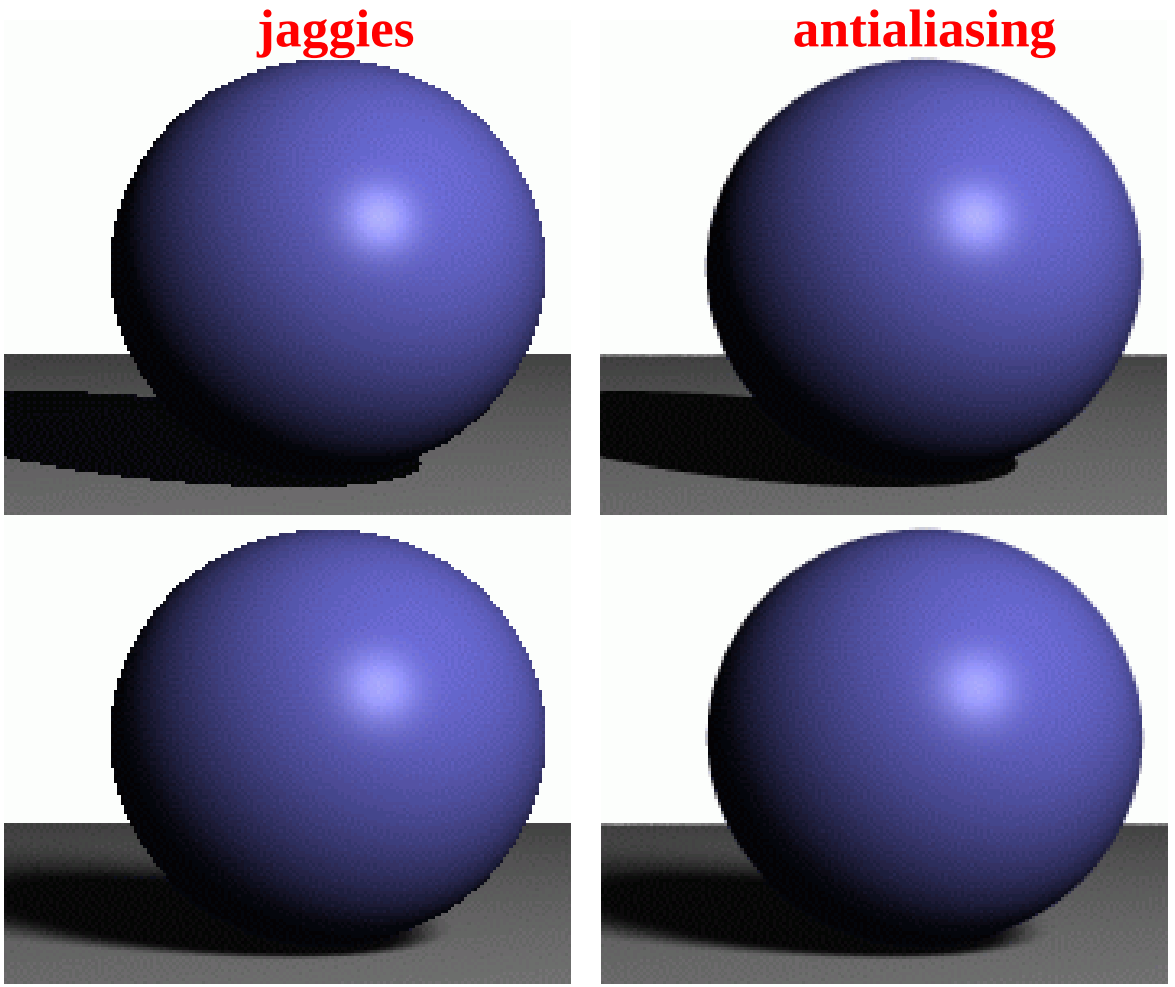
Distributed Ray Tracing

- Effects

  Soft shadow

  &

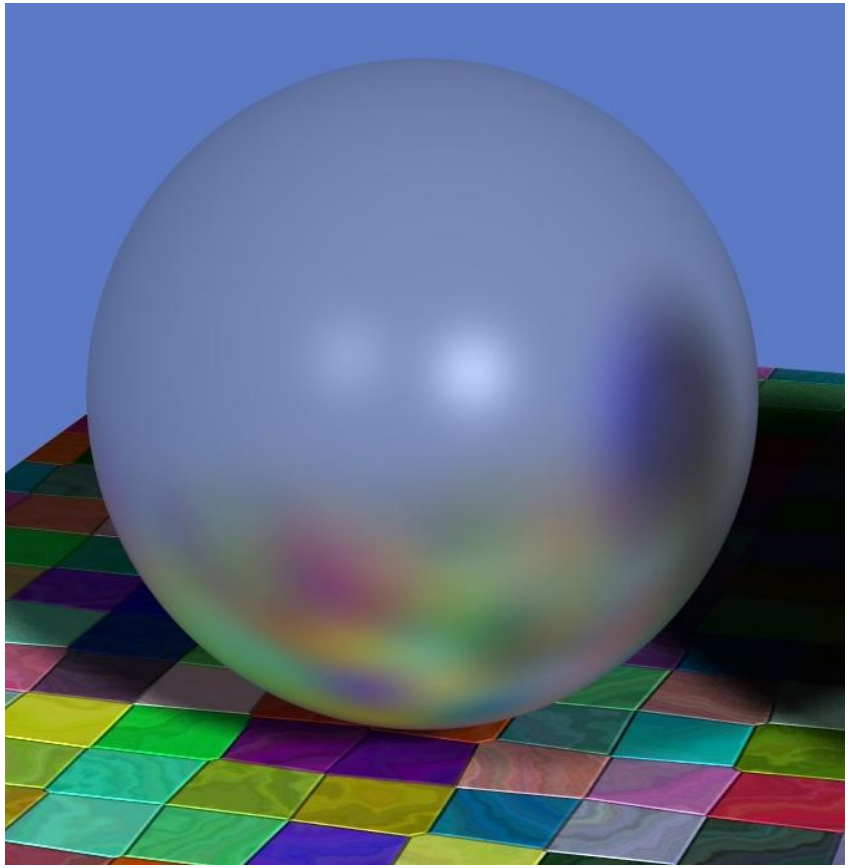
  Anti-Aliasing



Distributed Ray Tracing

- Effects

  glossy reflection



Distributed Ray Tracing

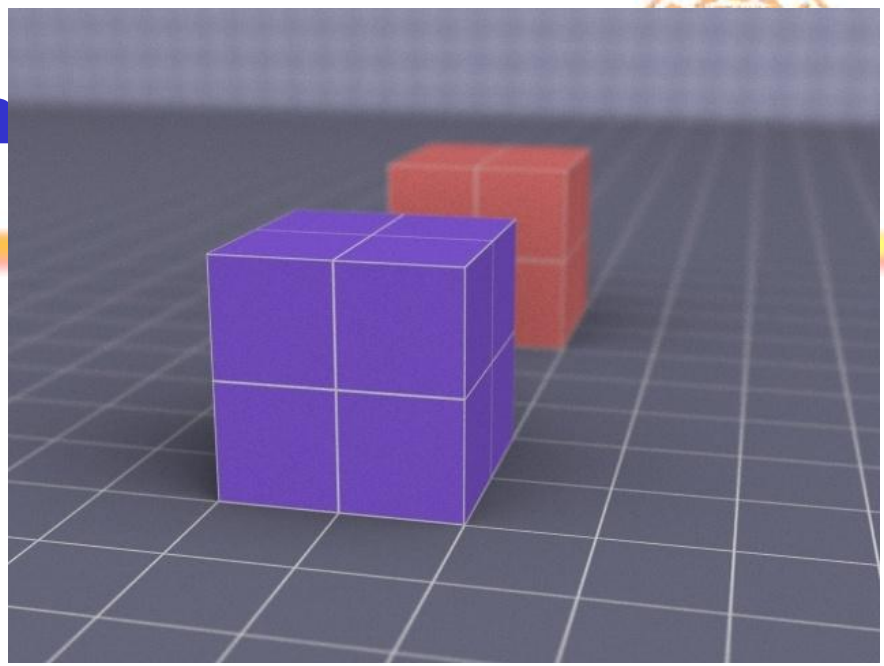
- Effects

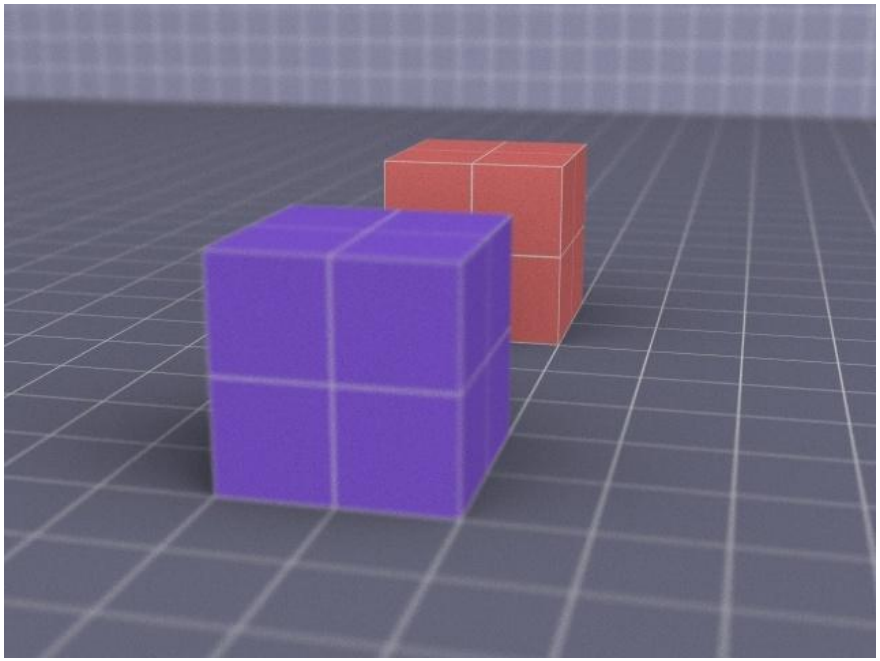
  motion blur

Distributed Ray Tra

- Effects

  depth of field



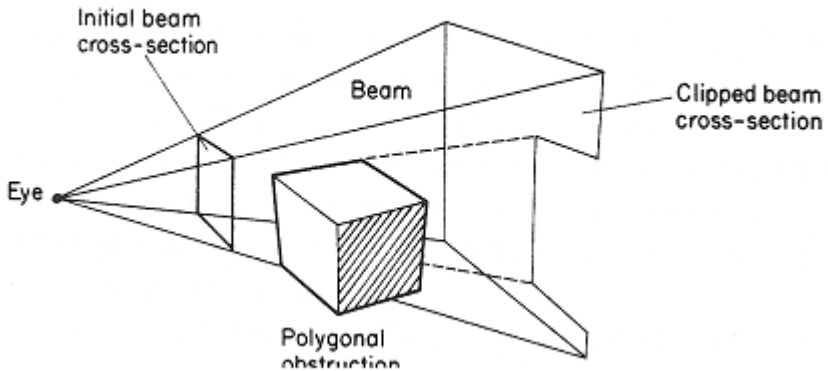


Other techniques

- Beam(光束) tracing
  - Beam tracing is a derivative of the ray tracing algorithm that replaces rays, which have no thickness, with beams. Beams are shaped like unbounded pyramids.
  - In beam tracing, a pyramidal beam is initially cast through the entire viewing frustum. This initial viewing beam is intersected with each polygon in the environment, from nearest to farthest. When a beam intersects with a reflective or refractive polygon, a new beam is created in a similar fashion to ray-tracing.

Beam Tracing

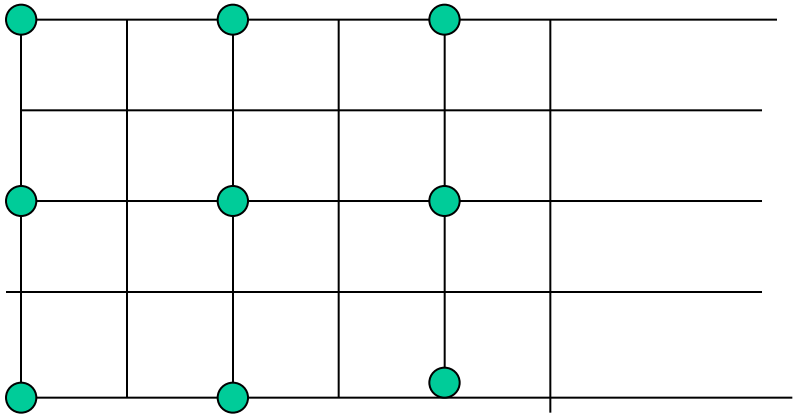
- General idea
  - utilize the continuity of rays to accelerate
  - For example, test intersecting



a beam with a convex polygon, if the 4 corner ray of the beam intersects with the polygon, then all rays inside the beam would intersect with the polygon

Other techniques

- Selected ray tracing and Interpolation
  - We can select pixel for ray racing, for example, we may trace pixel of corners of 2*2, 3*3or 4*4 block
  - Then interpolate colors



- Better pixel selection by Wavlet
  - a technique using wavelet for importance sampling.
  - (importance sampling is the term for "how to choose important sample points").
  - See Video（Wis-Video）

- Domo of ray tracing
  - RPU: a programmable Ray Processing Unit for real-time ray tracing

    rpu_video_2xfpga
  - The RPU is a fully programmable ray tracing hardware, with support for programmable material, geometry and lighting. (This is a Siggraph2005 paper)
  - See video

Thanks!