CS 6958
LECTURE 8
TRIANGLES, BVH

February 3, 2014

Last Time

- derived ray-triangle intersection
- clarification:
  - ray tracing inherently abstract in terms of object specification
  - we can use any object once we define an algorithm for intersecting it with a ray (and computing localized normal direction)

Ray Tracing Algorithm


```
foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects
      shade intersection point
```

Ray Tracing Algorithm


```
foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects
      shade intersection point
```

```
foreach object
  t_new = object.intersect(ray)
  t_closest = min(t_closest, t_new)
```

Ray Tracing Algorithm


```
/// Abstract Primitive class defining properties which are required for our ray tracer.
/// For now, it specifies just ray-object intersection routine, but can be extended to
/// support shadow rays, bounding volumes, etc
class Primitive {
public:
    virtual bool Intersect(const Ray &ray) const = 0;
}

/// Sphere primitive
class Sphere : public Primitive {
    bool Intersect(const Ray &ray) const;
}


// Triangle primitive
class Triangle : public Primitive {
    bool Intersect(const Ray &ray) const;
}
```

Ray Tracing Algorithm


```
/// Abstract Primitive class defining properties which are required for our ray tracer.
/// For now, it specifies just ray-object intersection routine, but can be extended to
/// support shadow rays, bounding volumes, etc
class Primitive {
public:
    virtual bool Intersect(const Ray &ray) const = 0;
}

/// Sphere primitive
class Sphere : public Primitive {
    bool Intersect(const Ray &ray) const;
}

// Triangle primitive
class Triangle : public Primitive {
    bool Intersect(const Ray &ray) const;
}
```

Others:

- Torus
- Cone / Cylinder
- Box / Rectangle
- Extrusions
- Surfaces of revolution
- Metaballs
- Iso-surface
- Spline surfaces
- Subdivision surfaces

Ray Tracing Algorithm

Note! We can’t use inheritance, hence we are restricted to a single primitive

Making Ray Tracing Faster

- faster rays
  - packets (less overhead per ray, cache coherence)
  - CPU optimizations
- fewer rays
  - adaptive super-sampling (less samples)

- faster ray-primitive intersection tests
- fewer ray-primitive intersection tests
  - acceleration structures

Which Operation Most Costly?

foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects
      shade intersection point

Acceleration Structures

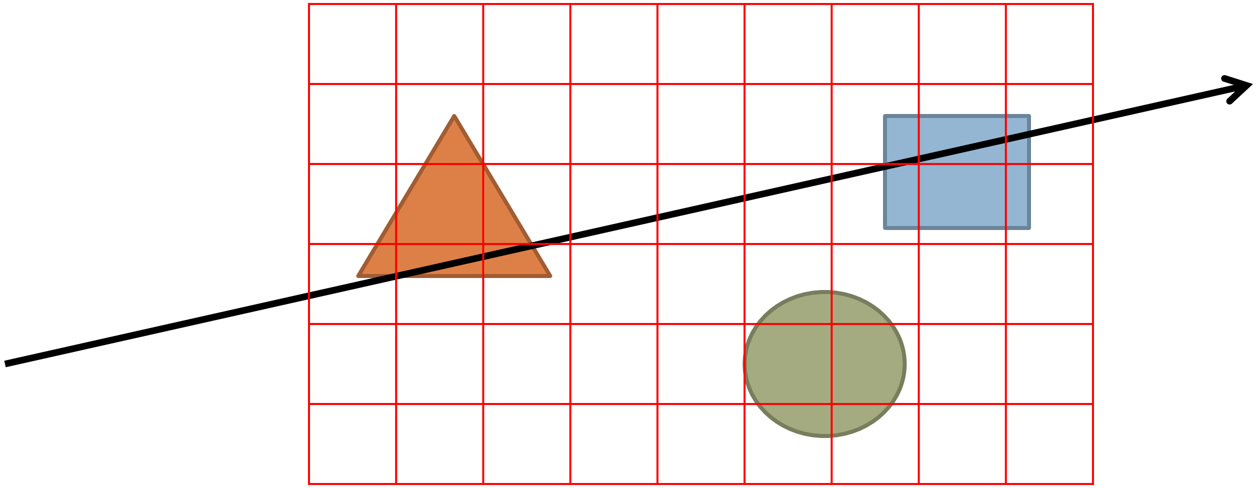

```
foreach frame
  foreach pixel
    foreach sample
      generate ray
      traverse ray through acceleration structure
      shade intersection point
```

- change O(n) to O(log n), n – objects in scene
- intersecting ray with structure primitive must be cheap

Acceleration Structures

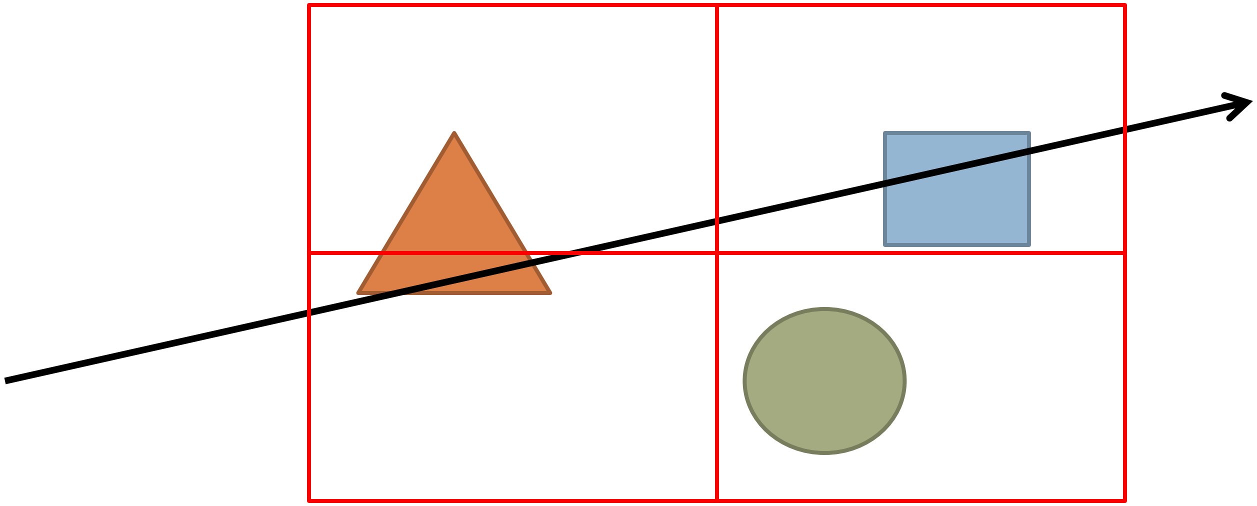
Acceleration Structures

- Grid



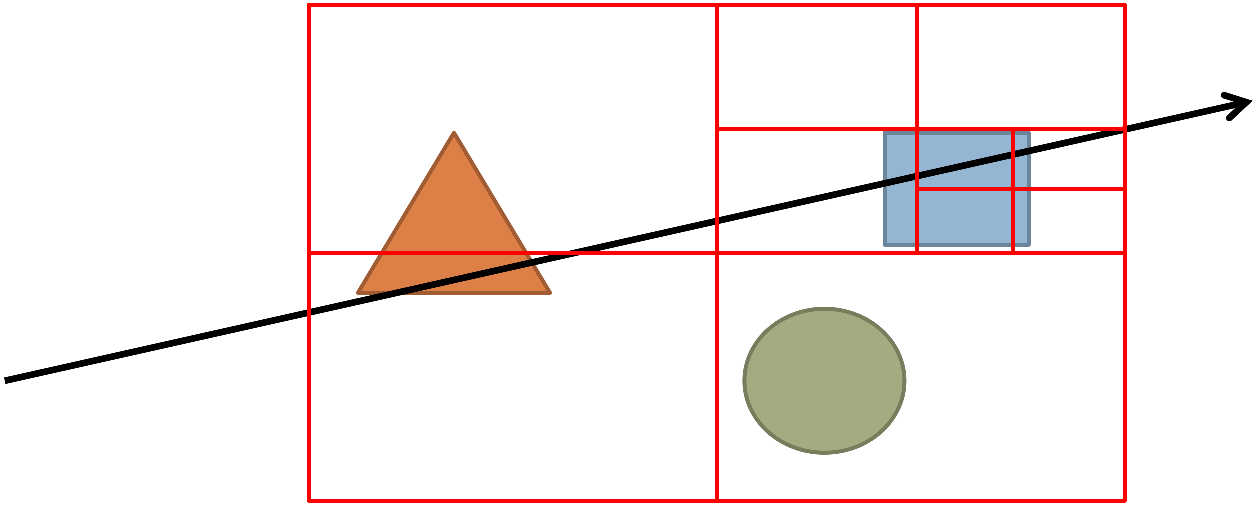
Acceleration Structures

- Grid
- Octree



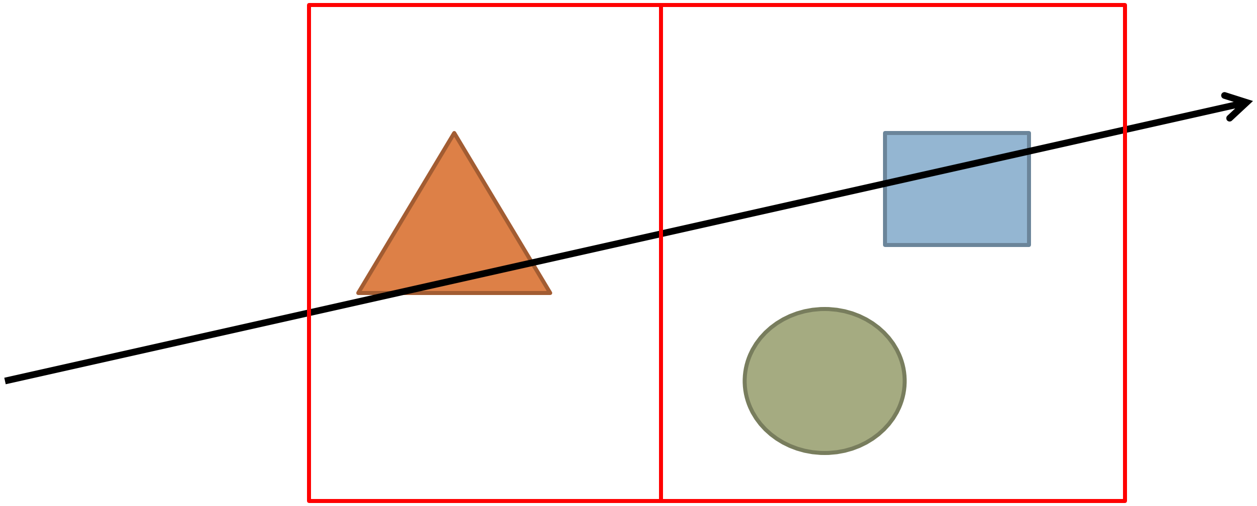
Acceleration Structures

- Grid
- Octree



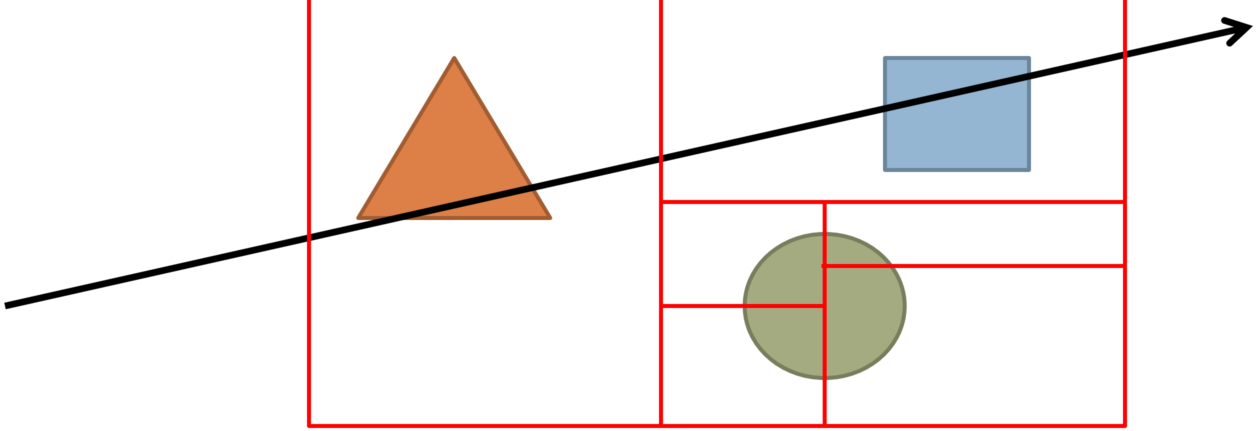
Acceleration Structures

- Grid
- Octree
- KD tree (K-dimensional)



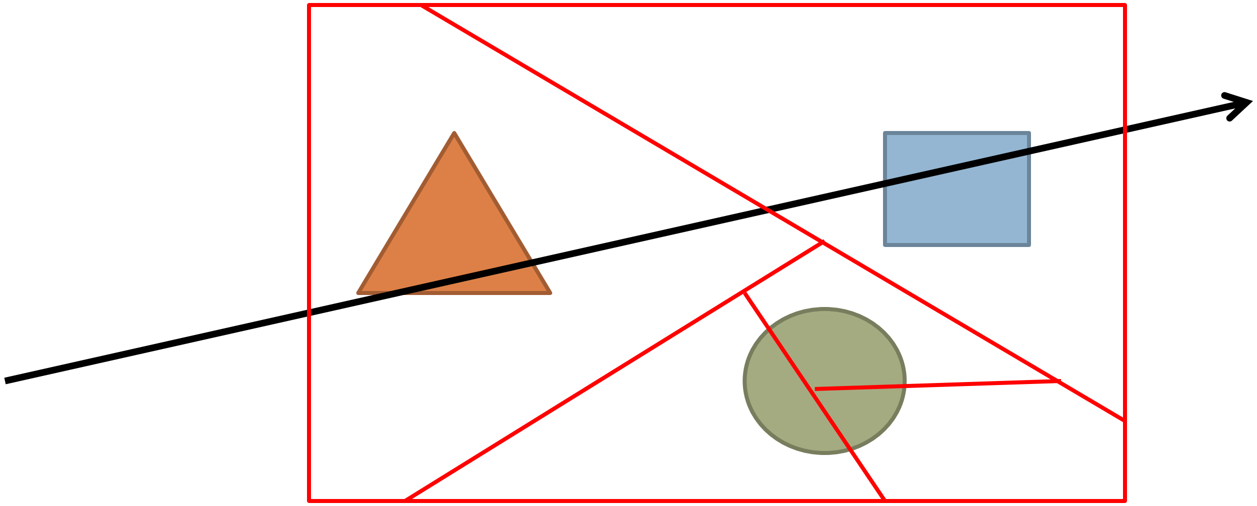
Acceleration Structures

- Grid
- Octree
- KD tree (K-dimensional)



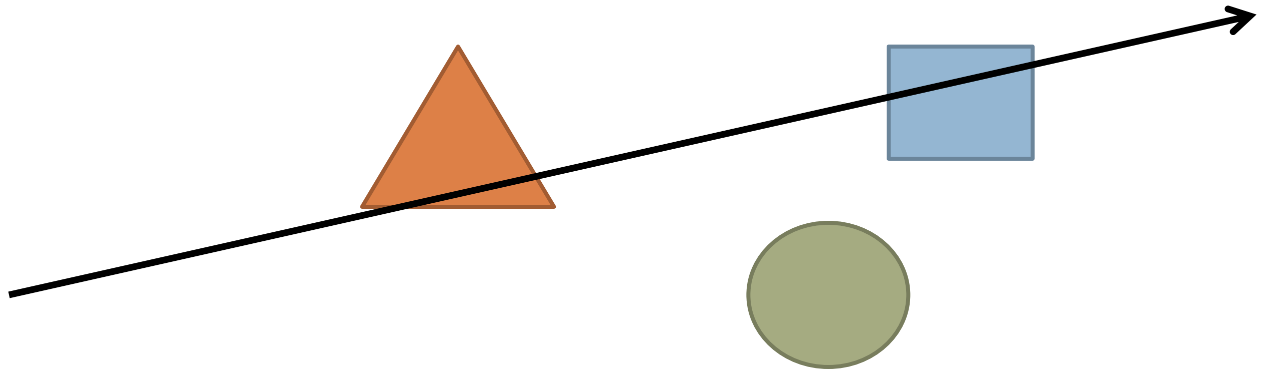
Acceleration Structures

- Grid
- Octree
- KD tree (K-dimensional)
- BSP tree (Binary Space Partitioning)



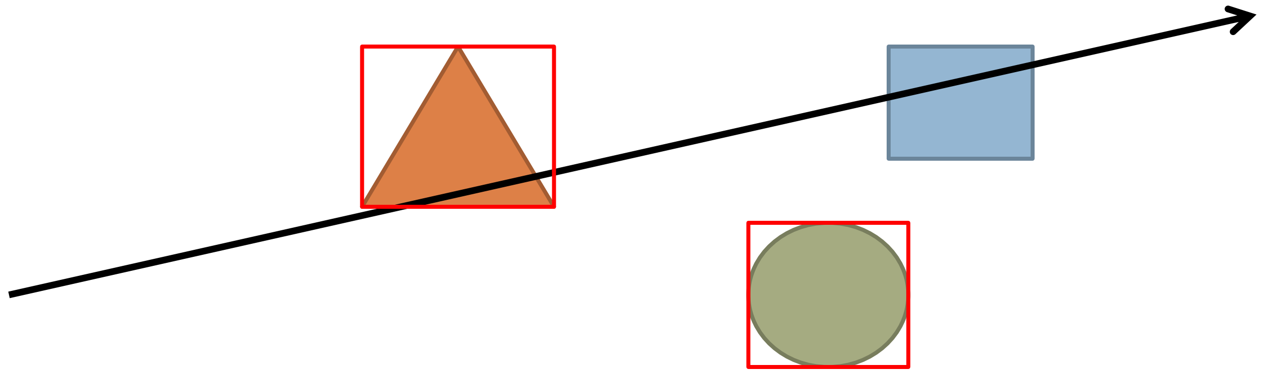
Acceleration Structures

- Grid
- Octree
- KD tree (K-dimensional)
- BSP tree (Binary Space Partitioning)
- BVH (Boundary Volume Hierarchy)



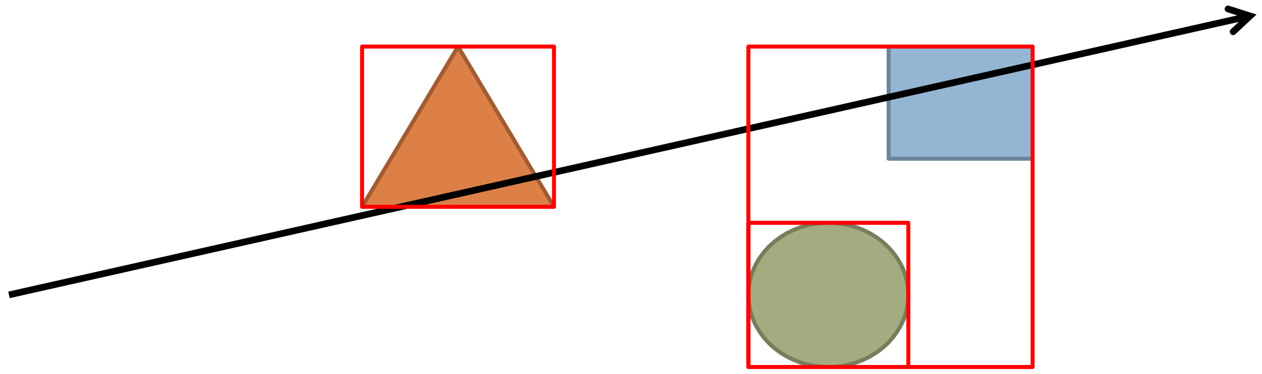
Acceleration Structures

- Grid
- Octree
- KD tree (K-dimensional)
- BSP tree (Binary Space Partitioning)
- BVH (Boundary Volume Hierarchy)



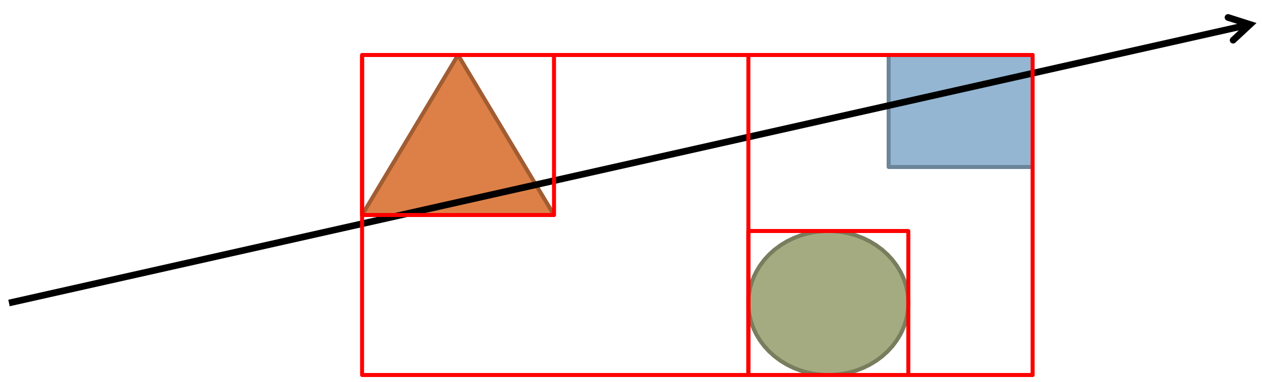
Acceleration Structures

- Grid
- Octree
- KD tree (K-dimensional)
- BSP tree (Binary Space Partitioning)
- BVH (Boundary Volume Hierarchy)

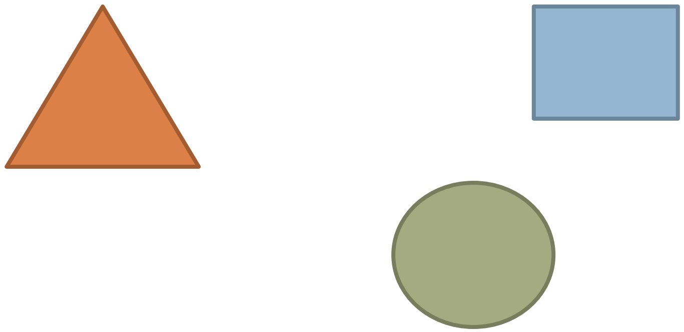


Acceleration Structures

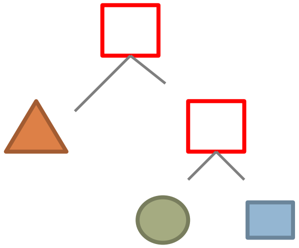
- Grid
- Octree
- KD tree (K-dimensional)
- BSP tree (Binary Space Partitioning)
- BVH (Boundary Volume Hierarchy)

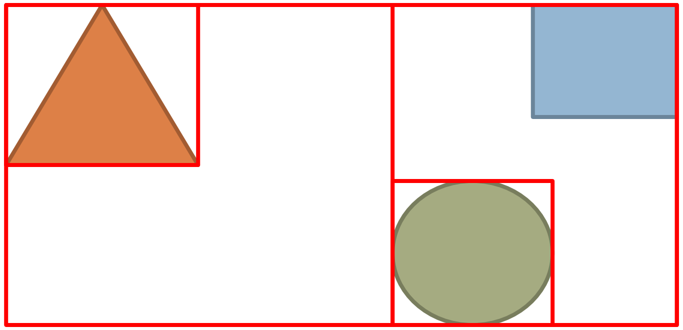


BVH Traversal - Idea

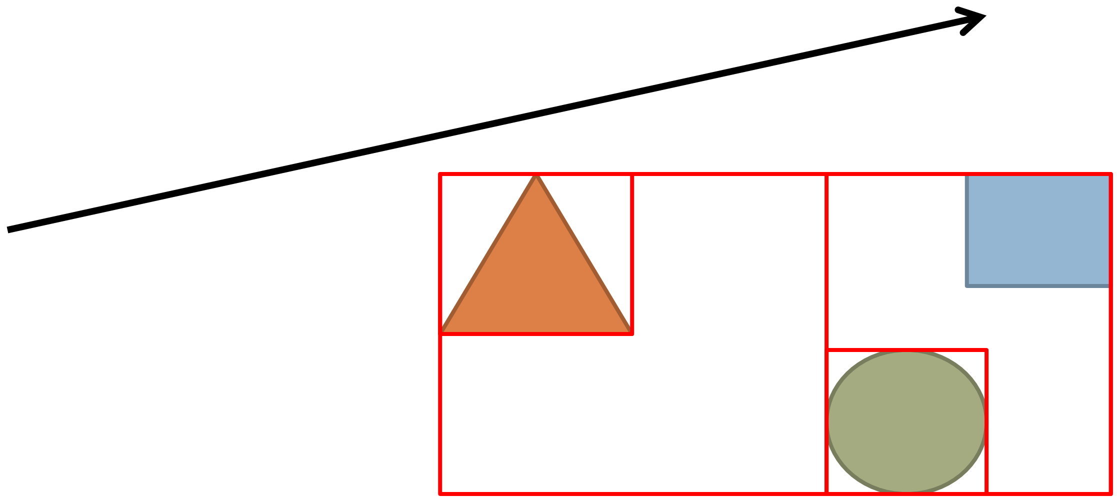


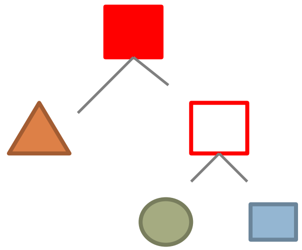
BVH Traversal - Idea



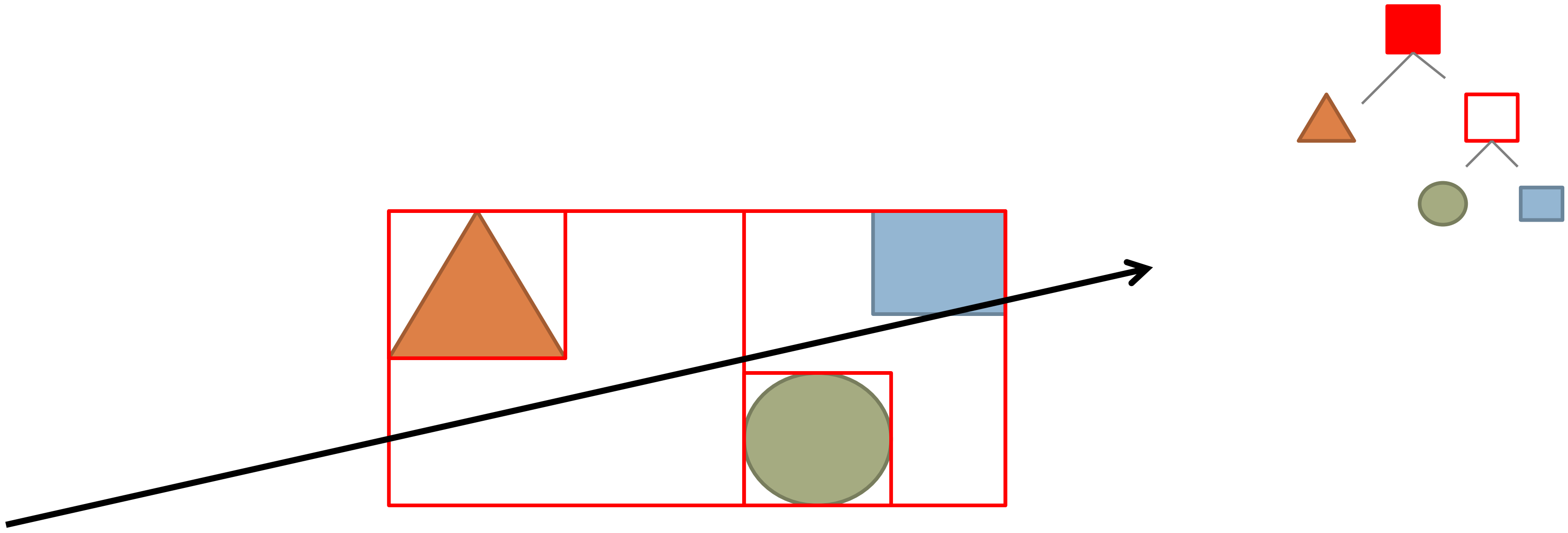


BVH Traversal - Idea

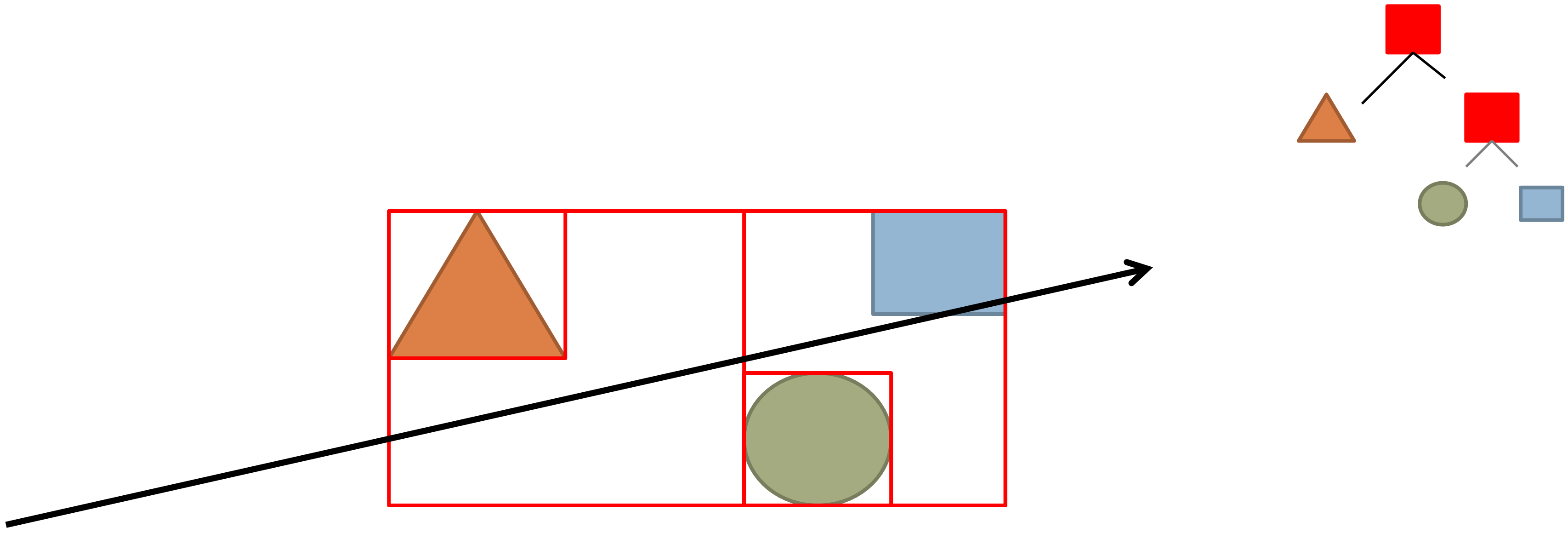




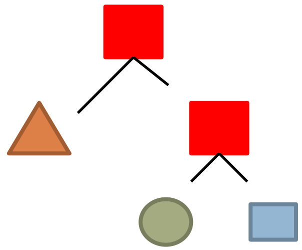
BVH Traversal - Idea

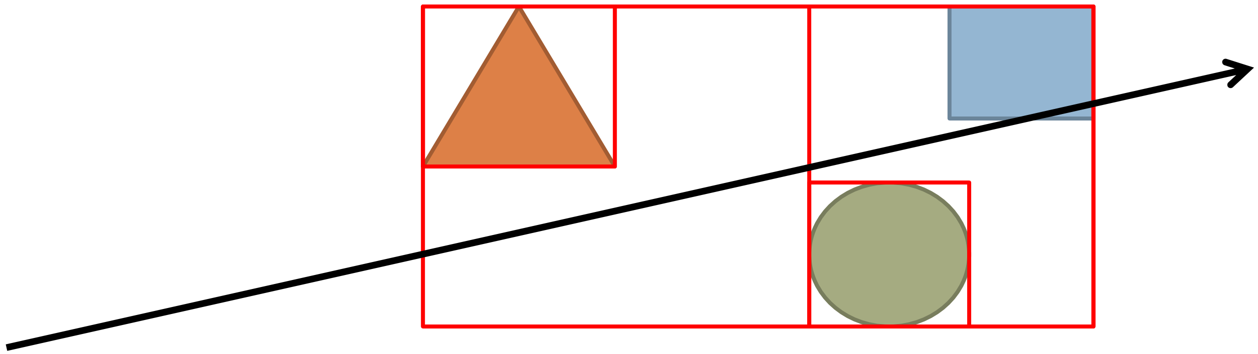


BVH Traversal - Idea

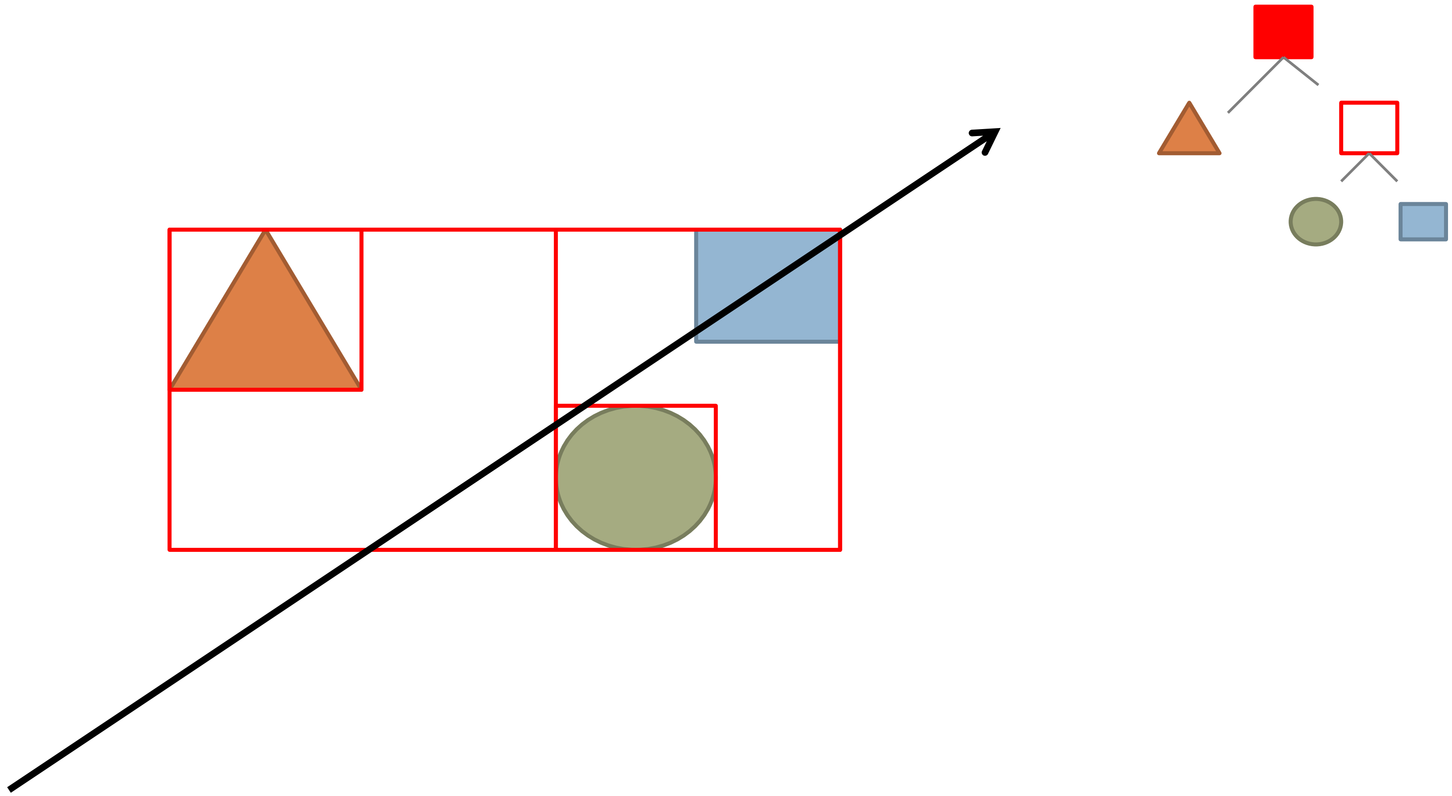


BVH Traversal - Idea

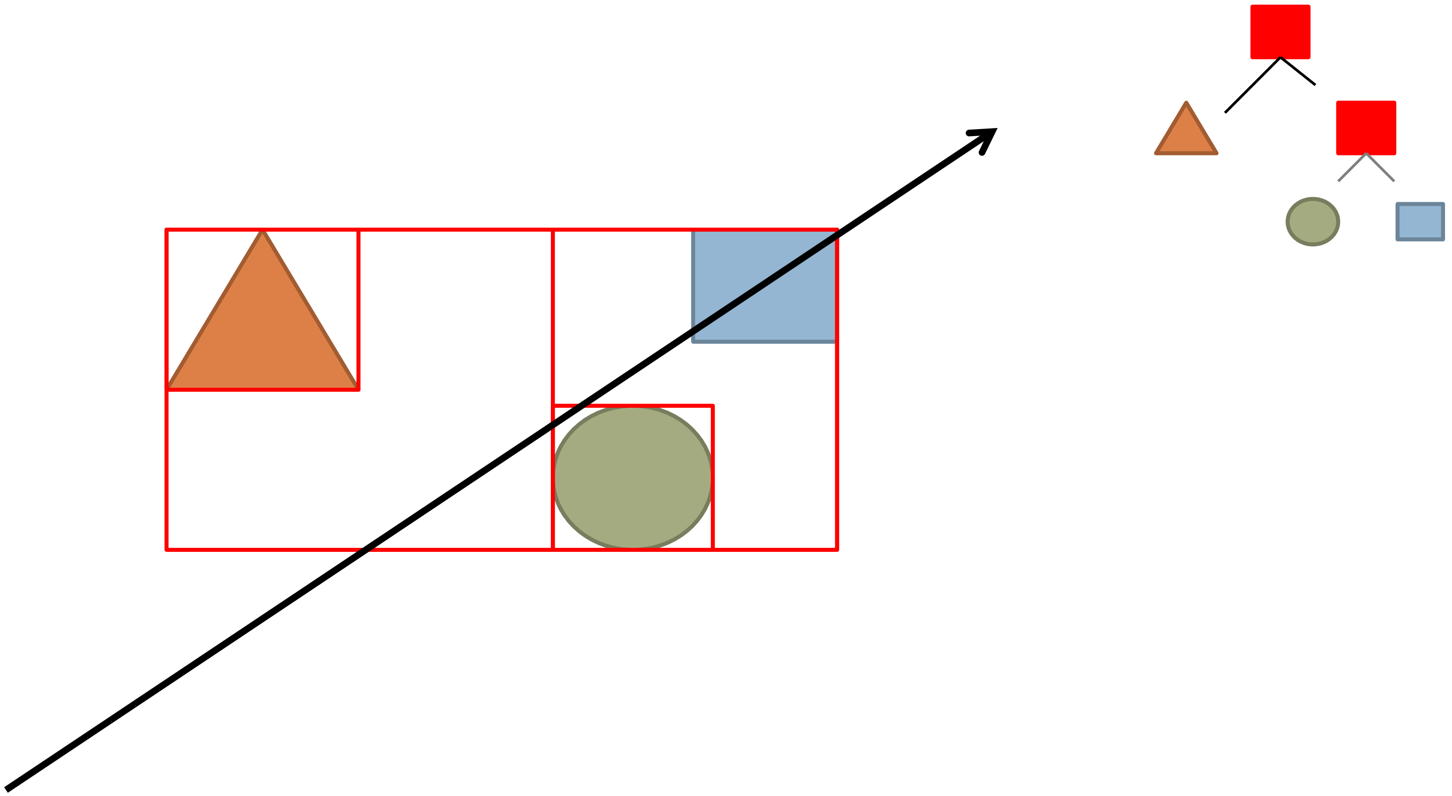




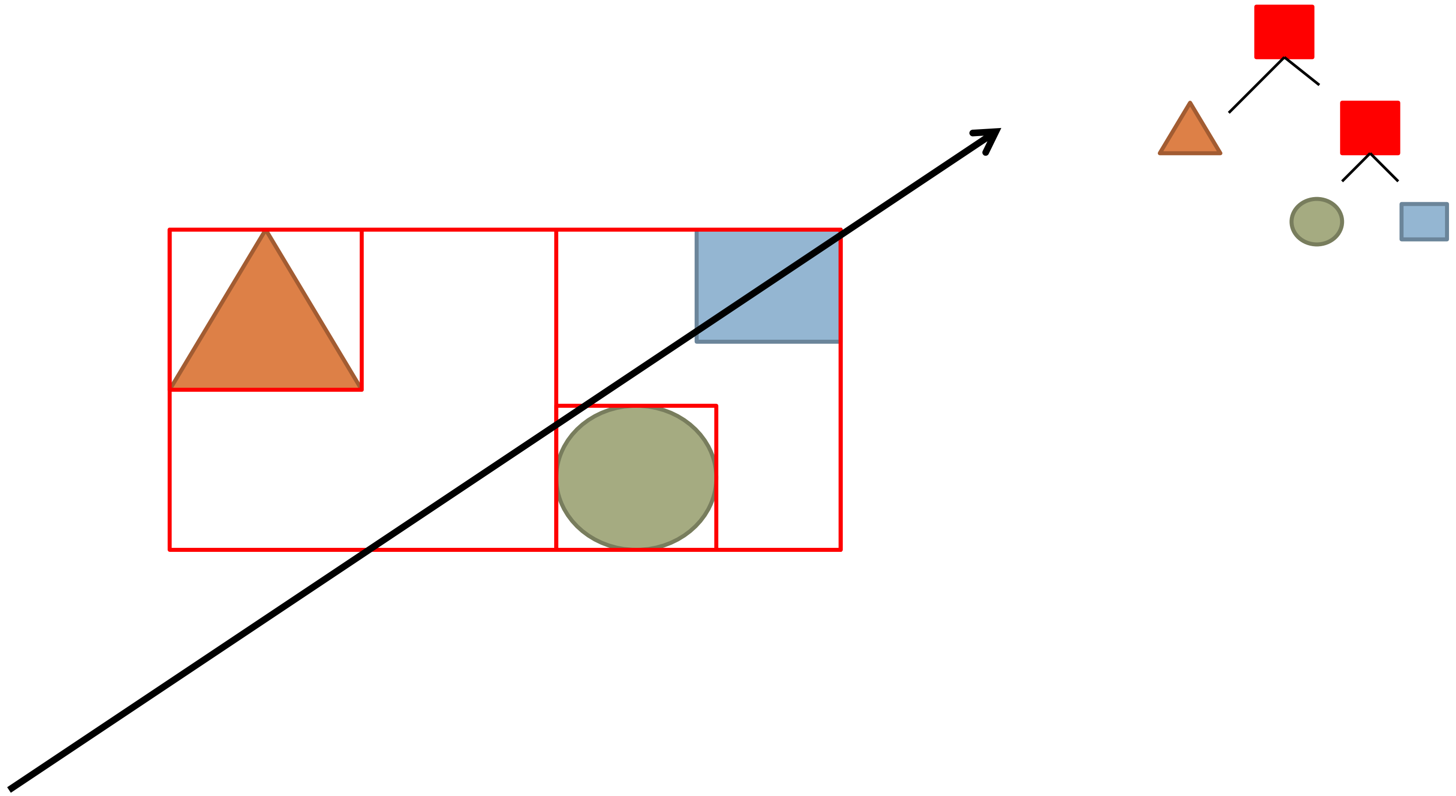
BVH Traversal - Idea



BVH Traversal - Idea



BVH Traversal - Idea



BVH Traversal - Pseudocode

- description is recursive, but
  - TPs have small stack memory, so manage it ourselves
  - code will run faster

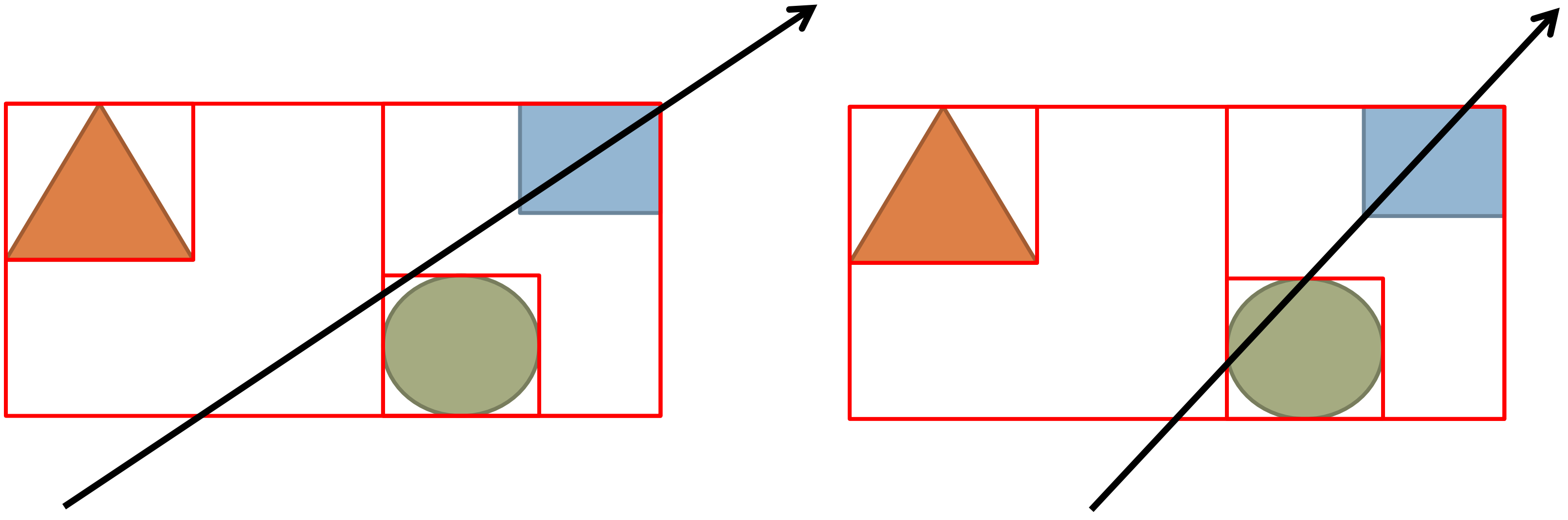
```
int stack[32];    // holds node IDs to traverse
int sp = 0;       // stack pointer into the above
```

BVH Traversal - Pseudocode


```
current_node = root
while(true) {
    if( ray intersects current_node ) {
        if( current_node._is_interior() ) {
            stack._push( current_node._right_child_id() )
            current_node = current_node._left_child_id()
            continue
        }
        else
            intersect all triangles in leaf
    }
    if( stack._is_empty() )
        break
    current_node = stack._pop()
}
```

BVH Traversal - Optimizations

- traverse closer child first
- don't traverse subtree if closer hit found



Axis Aligned Bounding Box

- Let’s try to derive an intersection test
- Box representation?

End