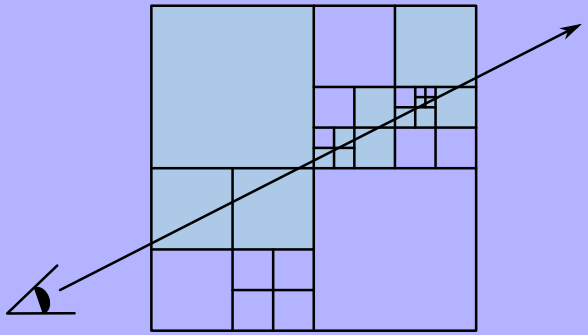
BSP Trees & k-D Trees

Motivation

- Octrees can give us dramatic speedups
- But: They might still be doing too much work

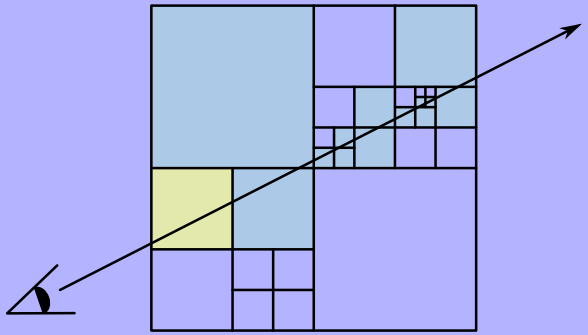
Consider

- Imagine a scene like this (side view)
  - We need to examine all the highlighted boxes



Consider

- But what if we have an intersection in the yellow box?
- Can't possibly have a closer intersection in any of the other boxes!
  - Why bother examining them?



Alternatives

- Some alternate approaches to searching: BSP trees and k-D trees

BSP

- We are building a tree, so we must define a node structure

```
class Node{
  public:
    unsigned left, right;   //0=no child
    Triangle tri;
    vec4 plane;          //plane containing triangle
    static vector<Node> nodes;
};
```

Building BSP

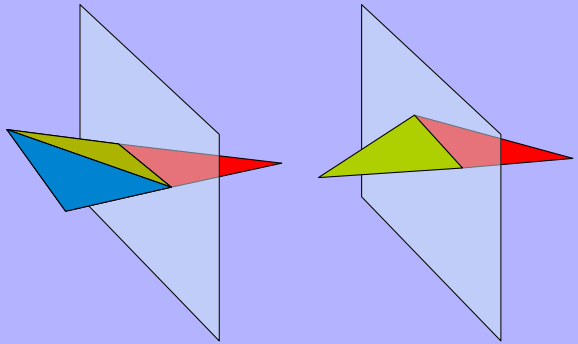
- Pick a triangle at random
- The plane containing the triangle divides the world in two parts
  - “Front” (side that plane normal faces)
  - “Back” (other side)

Building

- Take all triangles that are on front side and put them in one list
- Take all triangles that are on back side and put them in another list
- What about triangles that span the plane?

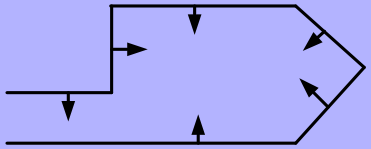
Span

- Choice 1: (Easy, lower performance): Put the triangle in both lists
- Choice 2: (More work, better performance): Split the triangle. Two ways triangle can be split...

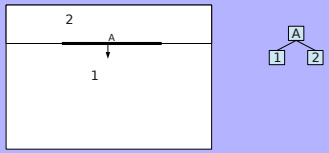


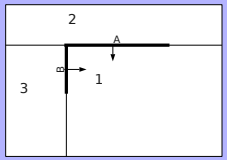
Example

- BSP is often used for world collision detection, so we'll use that for an example here
- Overhead view of the room we want to build BSP tree for:

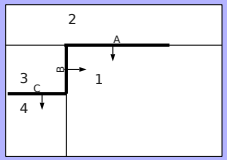


Building

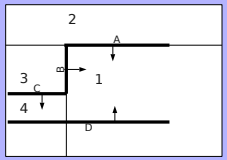


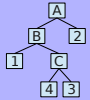

2
A
B
1
3

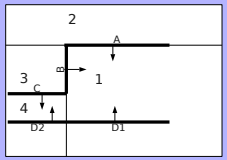

A
B
2
1
3

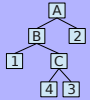

2
A
B
3
1
C
4

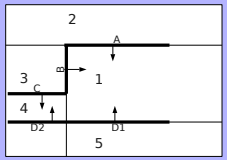

A
B
2
1
C
4
3

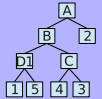

2
A
3
B
1
C
4
D

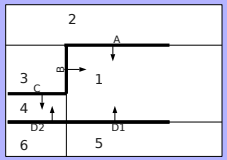

A
B
2
1
C
4
3

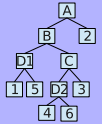

2
A
3
1
C
4
D2
D1


A
B
2
1
C
4
3

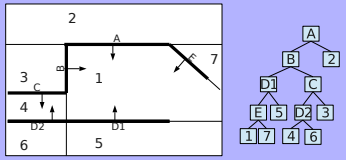

2
A
B
3
1
C
4
D2
D1
5


A
B
2
D1
C
1
5
4
3

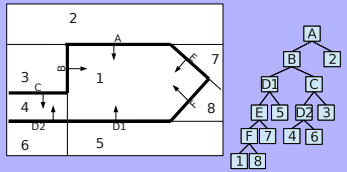

2
A
B
3
1
C
4
D2
D1
6
5


A
B
2
D1
C
1
5
D2
3
4
6

Building

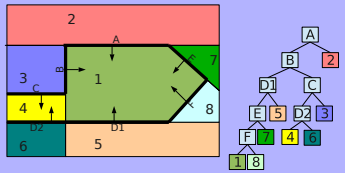


Building



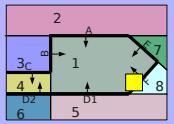
Building

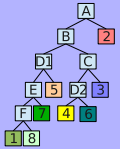
- As a nice bonus, the BSP tree also divides the world into regions
- Useful for doing things like object-floor collision tests (each region can have different floor height)



Collision Test

- Suppose we have a bounding box like the one shown in yellow
- (Demonstrate how we do collision test with walls)





3D

- BSP tree is similar in 3D except we have planes instead of lines

k-D Tree

- D = “Dimensional”
- People often say things like “3 dimensional k-D tree”
  - Which is not really *correct*
- k-D trees are like BSP trees, but we do the splitting a bit differently
  - Note: Several flavors...We’ll only discuss one way
  - If you search the internet, you may find other permutations of this approach

k-D Tree

- We restrict the splitting plane to be one of the {XY, XZ, YZ} planes
- We also cyclically alternate them
- Ex: First time we split, we use XY plane
- Next time, we use XZ plane
- Next time, we use YZ plane
- And then go back to XY plane

Selecting Plane

- How to choose the plane? Need plane A,B,C,D values
  - We know its normal, so we know A,B,C
  - Just need to pick D

Selecting Plane

- One way: Always take median point for region (center)

Selecting Plane

- Another way: Pick vertex that's close to center and align plane with that
  - This can reduce number of splits
- Note: Since k-D planes are always axis aligned, we have fewer floating point precision issues to worry about when splitting triangles
  - BSP splits are sometimes problematic due to FP precision

Stopping

- Stopping criteria: Can choose:
  - When one triangle in region
  - When < threshhold triangles in region
  - When splitting doesn't reduce number of triangles
  - When we hit some predetermined depth

Note

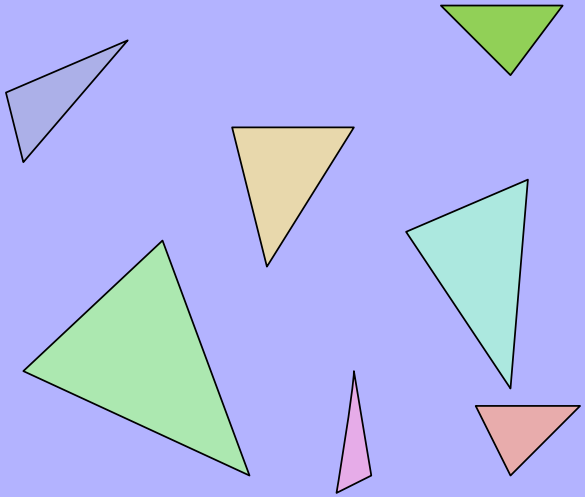
- We can also dispense with storing the plane A,B,C for a node
  - We know the A,B,C by what level we are at in the tree
  - Saving memory = Saving cache space

Example

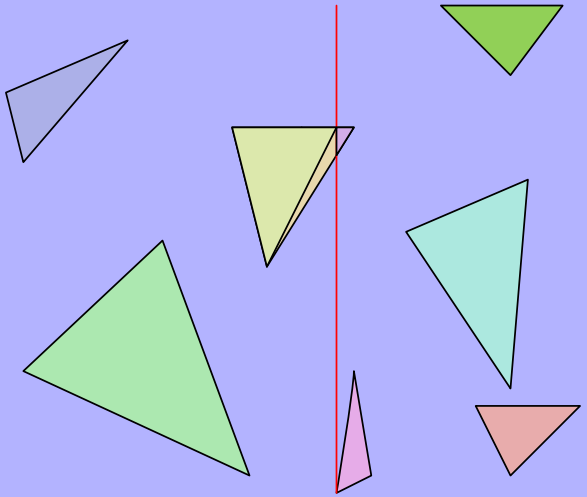
- Example node definition (not saving memory!)

```
class Node{
  public:
    vec4 plane;              //plane A,B,C,D
    unsigned frontChild,backChild;  //0=no child
    vector<Triangle> triangles; //only used for leaves
    static vector<Node> nodes;
};
```

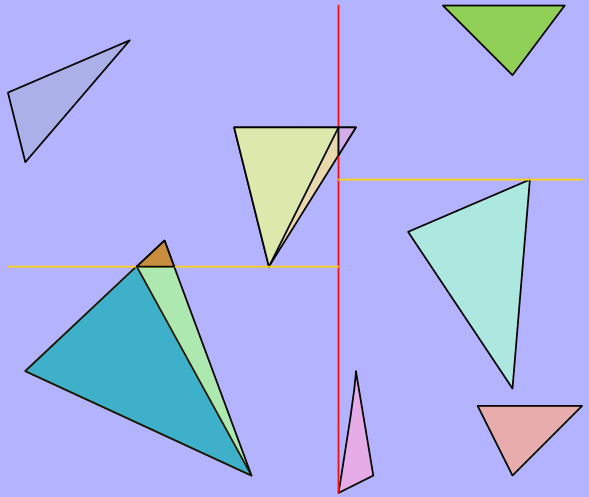
Example



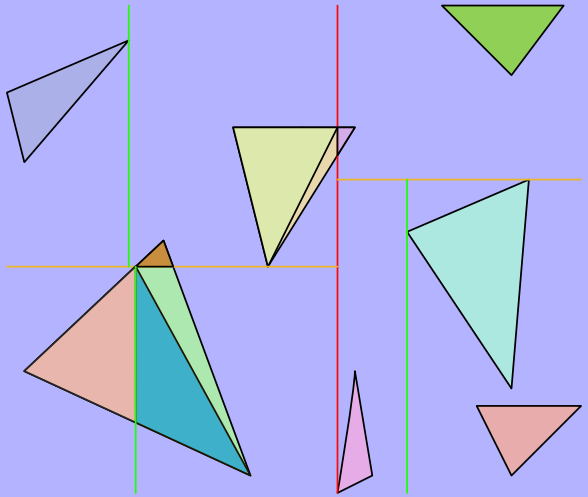
Example



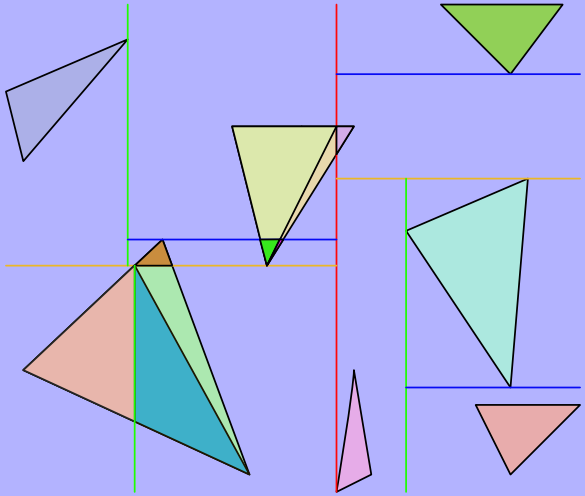
Example



Example



Example



Traversal

- To traverse

```
void trace(vec3& s, vec3& v){
    float t = infinity;
    stack<unsigned> stk;
    stk.push(0);        //root
    vec4 s_ = vec4(s.x,s.y,s.z,1);
    while(true){
        auto nidx = stk.pop();
        Node& node = Node::nodes[nidx];
        if( node.triangles.empty() ){
            //non-leaf
            unsigned side = dot(s_,node.plane);
            if( side >= 0 ){
                //ray start is on "positive" side of plane
                stk.push(node.backChild);
                stk.push(node.frontChild);  //this one will be done first
            } else {
                stk.push(node.frontChild);
                stk.push(node.backChild);
            }
        } else {
            for(auto& T: node.triangles ){
                if( ray intersects T )
                    return;
            }
        }
}
```

Note

- We can stop after the first intersection
- Why?
  - Consider splitting plane P
  - Suppose ray start is on positive side of P
  - If there's a ray-triangle intersection on positive side of P, it must be closer than any possible ray-triangle intersection on the negative side of P
  - (Diagram on board)

Sources

- http://www.sci.utah.edu/~wald/PhD/index.html
- https://stackoverflow.com/questions/4632951/kdtree-splitting/4633332#4633332
- https://stackoverflow.com/questions/13704762/is-k-d-tree-suited-for-keeping-triangles-or-i-need-some-changes-in-classic-k-d-t
- Christer Ericson. Real Time Collision Detection. CRC Press.

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org