

## Lecture 25 of 41

# Spatial Sorting: Binary Space Partitioning Quadtrees & Octrees

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course pages: <http://bit.ly/hGvXIH> / <http://bit.ly/eVizrE>

Public mirror web site: <http://www.kddresearch.org/Courses/CIS636>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

### Readings:

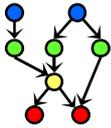
Today: Chapter 6, esp. §6.1, Eberly 2<sup>e</sup> – see <http://bit.ly/ieUq45>

Next class: Chapter 7, §8.4, Eberly 2<sup>e</sup>

Wikipedia, *Binary Space Partitioning*: <http://bit.ly/eE10lc>

Wikipedia, *Quadtree* (<http://bit.ly/ky0Xy>) & *Octree* (<http://bit.ly/dVrthx>)

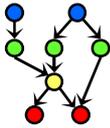




## Lecture Outline

- Reading for Last Class: §2.4.3, 8.1, Eberly 2<sup>e</sup>, **GL handout**
- Reading for Today: Chapter 6, Esp. §6.1, Eberly 2<sup>e</sup>
- Reading for Next Class: Chapter 7, §8.4, Eberly 2<sup>e</sup>
- Last Time: Collision Handling, Part 1 of 2
  - \* Static vs. dynamic objects, testing vs. finding intersections
  - \* Distance vs. intersection methods
  - \* Triangle point containment test
  - \* Method of separating axes
- Today: Adaptive Spatial Partitioning
  - \* Visible Surface Determination (VSD) revisited
  - \* Constructive Solid Geometry (CSG) trees
  - \* Binary Space Partitioning (BSP) trees
  - \* Quadtrees: adaptive 2-D (planar) subdivision
  - \* Octrees: adaptive 3-D (spatial) subdivision
- Coming Soon: Volume Graphics & Voxels





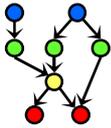
## Where We Are

21	Lab 4a: Animation Basics	Flash animation handout
22	Animation 2: Rotations; Dynamics, Kinematics	Chapter 17, esp. §17.1 – 17.2
23	Demos 4: Modeling & Simulation, Rotations	Chapter 10 <sup>1</sup> , 13 <sup>2</sup> , §17.3 – 17.5
24	Collisions 1: axes, OBBs, Lab 4b	§2.4.3, 8.1, GL handout
25	Spatial Sorting: Binary Space Partitioning	Chapter 6, esp. §6.1
26	Demos 5: More CGA; Picking; HW/Exam	Chapter 7 <sup>1</sup> ; § 8.4
27	Lab 5a: Interaction Handling	§ 8.3 – 8.4; 4.2, 5.0, 5.6, 9.1
28	Collisions 2: Dynamic, Particle Systems	§ 9.1, particle system handout
	Exam 2 review; Hour Exam 2 (evening)	Chapters 5 – 6, 7 <sup>2</sup> – 8, 12, 17
29	Lab 5b: Particle Systems	Particle system handout
30	Animation 3: Control & IK	§ 5.3, CGA handout
31	Ray Tracing 1: intersections, ray trees	Chapter 14
32	Lab 6a: Ray Tracing Basics with POV-Ray	RT handout
33	Ray Tracing 2: advanced topic survey	Chapter 15, RT handout
34	Visualization 1: Data (Quantities & Evidence)	Tufe handout (1)
35	Lab 6b: More Ray Tracing	RT handout
36	Visualization 2: Objects	Tufe handout (2 & 4)
37	Color Basics; Term Project Prep	Color handout
38	Lab 7: Fractals & Terrain Generation	Fractals/Terrain handout
39	Visualization 3: Processes; Final Review 1	Tufe handout (3)
40	Project presentations 1; Final Review 2	–
41	Project presentations 2	–
	Final Exam	Ch. 1 – 8, 10 – 15, 17, 20

Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review, and the green-shaded entry, that of the term project.

Green, blue and red letters denote exam review, exam, and exam solution review dates.





## Acknowledgements: Intersections, Containment – Eberly 1<sup>e</sup>

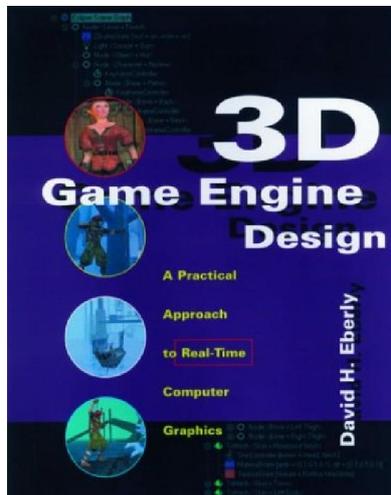
### David H. Eberly

Chief Technology Officer

Geometric Tools, LLC

<http://www.geometrictools.com>

<http://bit.ly/enKbfs>



*3D Game Engine Design* © 2000 D. H. Eberly  
See <http://bit.ly/ieUq45> for second edition table of contents (TOC)

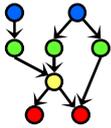
#### Last lecture's material:

- **View Frustum clipping**
  - §2.4.3, p. 70 – 77, 2<sup>e</sup>
  - §3.4.3, p. 93 – 99, & §3.7.2, p. 133 – 136, 1<sup>e</sup>
- **Collision detection: separating axes**
  - §8.1, p. 393 – 443, 2<sup>e</sup>
  - §6.4. p. 203 – 214, 1<sup>e</sup>

#### Later:

- **Distance methods**
  - Chapter 14, p. 639 – 679, 2<sup>e</sup>
  - §2.6, p. 38 – 77, 1<sup>e</sup>
- **Intersection methods**
  - Chapter 15, p. 681 – 717, 2<sup>e</sup>
  - §6.2 – 6.5, p. 188 – 243, 1<sup>e</sup>





## Review [1]: View Frustum Clipping

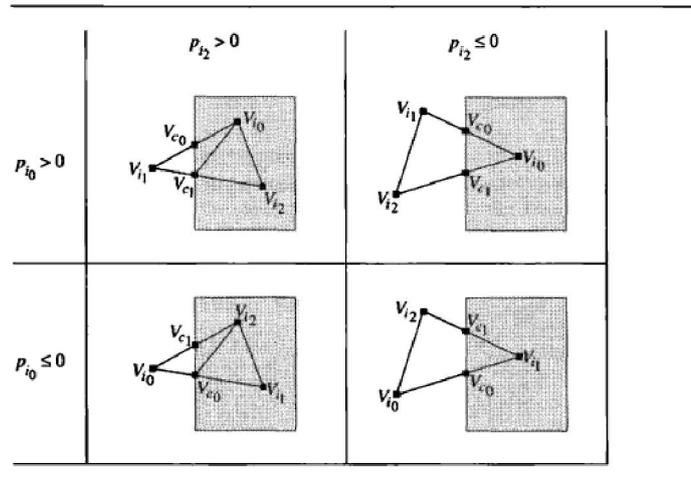
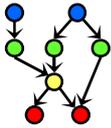


Figure 3.4 Four configurations for triangle splitting. Only the triangles in the shaded region are important, so the quadrilaterals outside are not split.

3D Game Engine Design © 2000 D. H. Eberly  
See <http://bit.ly/ieUq45> for second edition table of contents (TOC)





## Review [2]: Collision Detection vs. Response

### ■ Collision Detection

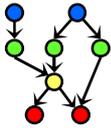
- Collision detection is a geometric problem
- Given two moving objects defined in an initial and final configuration, determine if they intersected at some point between the two states

### ■ Collision Response

- The response to collisions is the actual physics problem of determining the unknown forces (or impulses) of the collision

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD  
CSE169: Computer Animation, Winter 2005, <http://bit.ly/f0ViAN>



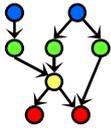


## Review [3]: Queries – Test- vs. Find-Intersection

- **Test-Intersection: Determine If Objects Intersect**
  - \* **Static:** test whether they do at given instant
  - \* **Dynamic:** test whether they intersect at any point along trajectories
- **Find-Intersection: Determine Intersection (or Contact) Set of Objects**
  - \* **Static:** intersection set (compare:  $A \cap B$ )
  - \* **Dynamic:** contact time (interval of overlap), sets (depends on time)

Adapted from *3D Game Engine Design* © 2000 D. H. Eberly  
See <http://bit.ly/ieUq45> for second edition table of contents (TOC)



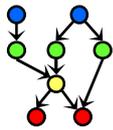


## Review [4]: Queries – Distance vs. Intersection

- **Distance-Based**
  - \* Parametric representation of object boundaries/interiors
  - \* Want: closest points on two objects (to see whether they intersect)
  - \* Use: constrained minimization to solve for closest points
- **Intersection-Based**
  - \* Also uses parametric representation
  - \* Want: overlapping subset of interior of two objects
  - \* General approach: equate objects, solve for parameters
  - \* Use one of two kinds of solution methods
    - Analytical (when feasible to solve exactly – e.g., OBBs)
    - Numerical (approximate region of overlap)
  - \* Solving for parameters in equation
  - \* Harder to compute than distance-based; use only when needed

Adapted from *3D Game Engine Design* © 2000 D. H. Eberly  
See <http://bit.ly/ieUq45> for second edition table of contents (TOC)



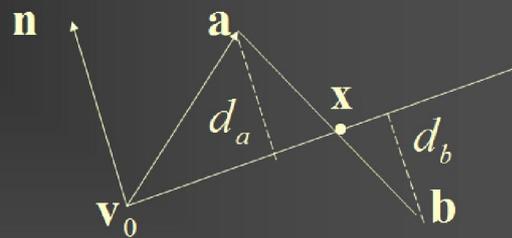


## Review [5]: Segment vs. Triangle – Solution

- First, compute signed distances of  $\mathbf{a}$  and  $\mathbf{b}$  to plane

$$d_a = (\mathbf{a} - \mathbf{v}_0) \cdot \mathbf{n}$$

$$d_b = (\mathbf{b} - \mathbf{v}_0) \cdot \mathbf{n}$$

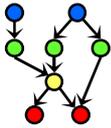


- Reject if both are above or both are below triangle
- Otherwise, find intersection point  $\mathbf{x}$

$$\mathbf{x} = \frac{d_a \mathbf{b} - d_b \mathbf{a}}{d_a - d_b}$$

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD  
CSE169: Computer Animation, Winter 2005, <http://bit.ly/f0ViAN>



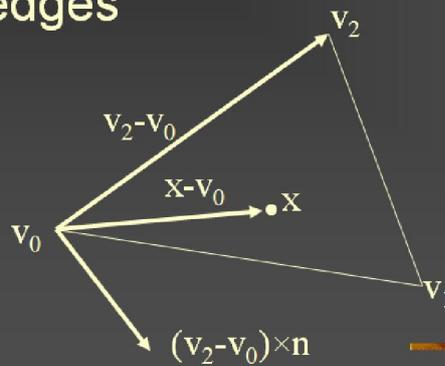


## Review [6]: Segment vs. Triangle – Point Test

- Is point  $x$  inside the triangle?

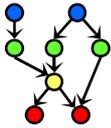
$$(x-v_0) \cdot ((v_2-v_0) \times n) > 0$$

- Test all 3 edges



Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD  
CSE169: Computer Animation, Winter 2005, <http://bit.ly/f0ViAN>





## Review [7]: Faster Triangle – Point Containment

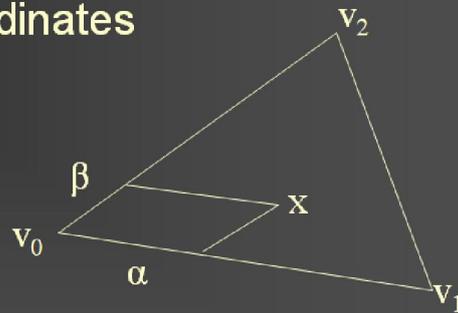
- Reduce to 2D: remove smallest dimension
- Compute barycentric coordinates
 
$$\mathbf{x}' = \mathbf{x} - \mathbf{v}_0$$

$$\mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0$$

$$\mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0$$

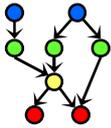
$$\alpha = (\mathbf{x}' \times \mathbf{e}_2) / (\mathbf{e}_1 \times \mathbf{e}_2)$$

$$\beta = (\mathbf{x}' \times \mathbf{e}_1) / (\mathbf{e}_1 \times \mathbf{e}_2)$$
- Reject if  $\alpha < 0$ ,  $\beta < 0$  or  $\alpha + \beta > 1$



Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD  
CSE169: Computer Animation, Winter 2005, <http://bit.ly/f0ViAN>

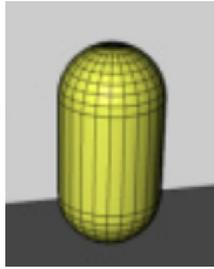




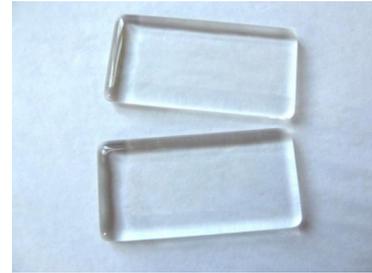
## Review [8]: Sphere-Swept Volumes & Distances



Wikipedia: *Sphere*  
<http://bit.ly/9OWjQi>  
 Image © 2008 ClipArtOf.com  
<http://bit.ly/eKhE2f>



Capsule  
 Image © 2007 Remotion Wiki  
<http://bit.ly/huEzNW>



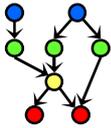
Lozenge  
 Image © 2011 Jasmin Studio Crafts  
<http://bit.ly/euEopw>

Table 6.1 Relationship between sphere-swept volumes and distance calculators (*pnt*, point; *seg*, line segment; *rct*, rectangle).

	<i>Sphere</i>	<i>Capsule</i>	<i>Lozenge</i>
<i>Sphere</i>	dist( <i>pnt</i> , <i>pnt</i> )	dist( <i>pnt</i> , <i>seg</i> )	dist( <i>pnt</i> , <i>rct</i> )
<i>Capsule</i>	dist( <i>seg</i> , <i>pnt</i> )	dist( <i>seg</i> , <i>seg</i> )	dist( <i>seg</i> , <i>rct</i> )
<i>Lozenge</i>	dist( <i>rct</i> , <i>pnt</i> )	dist( <i>rct</i> , <i>seg</i> )	dist( <i>rct</i> , <i>rct</i> )

Adapted from *3D Game Engine Design* © 2000 D. H. Eberly  
 See <http://bit.ly/ieUq45> for second edition table of contents (TOC)





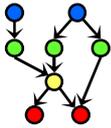
## Review [9]: Method of Separating Axes

Table 6.7 Values for  $R$ ,  $R_0$ , and  $R_1$  for the separating axis test  $R > R_0 + R_1$  for two boxes in the direction of motion.

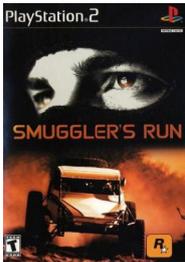
$\vec{L}$	$R_0$	$R_1$	$R$
$\vec{W} \times \vec{A}_0$	$a_1 \alpha_2  + a_2 \alpha_1 $	$\sum_{i=0}^2 b_i c_{1i}\alpha_2 - c_{2i}\alpha_1 $	$ \vec{A}_0 \cdot \vec{W} \times \vec{D} $
$\vec{W} \times \vec{A}_1$	$a_0 \alpha_2  + a_2 \alpha_0 $	$\sum_{i=0}^2 b_i c_{0i}\alpha_2 - c_{2i}\alpha_0 $	$ \vec{A}_1 \cdot \vec{W} \times \vec{D} $
$\vec{W} \times \vec{A}_2$	$a_0 \alpha_1  + a_1 \alpha_0 $	$\sum_{i=0}^2 b_i c_{0i}\alpha_1 - c_{1i}\alpha_0 $	$ \vec{A}_2 \cdot \vec{W} \times \vec{D} $
$\vec{W} \times \vec{B}_0$	$\sum_{i=0}^2 a_i c_{i1}\beta_2 - c_{i2}\beta_1 $	$b_1 \beta_2  + b_2 \beta_1 $	$ \vec{B}_0 \cdot \vec{W} \times \vec{D} $
$\vec{W} \times \vec{B}_1$	$\sum_{i=0}^2 a_i c_{i0}\beta_2 - c_{i2}\beta_0 $	$b_0 \beta_2  + b_2 \beta_0 $	$ \vec{B}_1 \cdot \vec{W} \times \vec{D} $
$\vec{W} \times \vec{B}_2$	$\sum_{i=0}^2 a_i c_{i0}\beta_1 - c_{i1}\beta_0 $	$b_0 \beta_1  + b_1 \beta_0 $	$ \vec{B}_2 \cdot \vec{W} \times \vec{D} $

3D Game Engine Design © 2000 D. H. Eberly  
See <http://bit.ly/ieUq45> for second edition table of contents (TOC)





## Acknowledgements: Collisions, BSP/Quadtrees/Octrees



### Steve Rotenberg

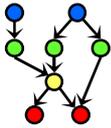
Visiting Lecturer  
Graphics Lab  
University of California – San Diego  
CEO/Chief Scientist, PixelActive  
<http://graphics.ucsd.edu>



### Glenn G. Chappell

Associate Professor  
Department of Computer Science  
University of Alaska Fairbanks  
<http://www.cs.uaf.edu/~chappell/>



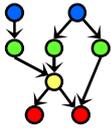


## Data Structures for Scenes [1]: Four Tree Representations

- **Scene Graphs**
  - \* Organized by how scene is constructed
  - \* Nodes hold objects
- **Constructive Solid Geometry (CSG) Trees**
  - \* Organized by how scene is constructed
  - \* Leaves hold 3-D primitives
  - \* Internal nodes hold set operations
- **Binary Space Partitioning (BSP) Trees**
  - \* Organized by spatial relationships in scene
  - \* Nodes hold facets (in 3-D, polygons)
- **Quadtrees & Octrees**
  - \* Organized spatially
  - \* Nodes represent regions in space
  - \* Leaves hold objects

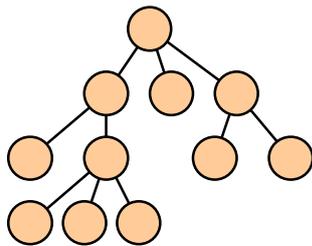
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>



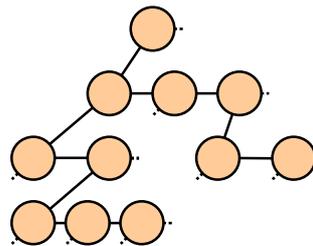


## Data Structures for Scenes [2]: Implementing Scene Graphs

- We think of scene graphs as looking like the tree on the left.
- However, it is often convenient to implement them as shown on the right.
  - \* Implementation is a B-tree.
  - \* Child pointers are first-logical-child and next-logical-sibling.
  - \* Then traversing the logical tree is a simple pre-order traversal of the physical tree. This is how we draw.



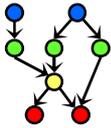
Logical Tree



Physical Tree

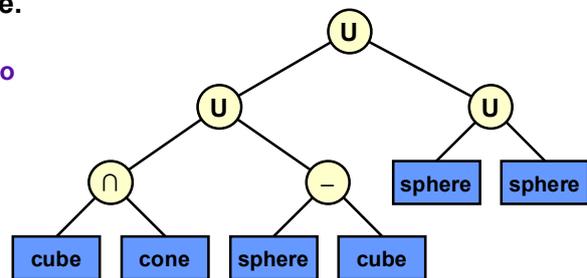
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





## Data Structures for Scenes [3]: Constructive Solid Geometry Trees

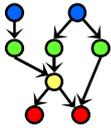
- In Constructive Solid Geometry (CSG), we construct a scene out of primitives representing solid 3-D shapes. Existing objects are combined using set operations (union, intersection, set difference).
- We represent a scene as a binary tree.
  - \* Leaves hold primitives.
  - \* Internal nodes, which always have two children, hold set operations.
  - \* Order of children matters!



- CSG trees are useful for things other than rendering.
  - \* Intersection tests (collision detection, etc.) are not too hard. (Thus: ray tracing.)
- CSG does not integrate well with pipeline-based rendering, so we are not covering it in depth right now.
  - \* *How about a project on CSG?*

Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>



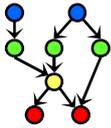


## Binary Space Partitioning Trees [1]: Idea

- **BSP tree: very different way to represent a scene**
  - \* Nodes hold facets
  - \* Structure of tree encodes spatial information about the scene
- **Applications**
  - \* Visible Surface Determination (VSD) aka Hidden Surface Removal
  - \* **Wikipedia: *Visible Surface Determination*, <http://bit.ly/et2yNQ>**
  - \* **Related applications: portal rendering (<http://bit.ly/fYO5T6>), etc.**

Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>



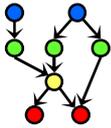


## Binary Space Partitioning Trees [2]: Definition

- **BSP tree: type of binary tree**
  - \* Nodes can have 0, 1, or two children
  - \* Order of child nodes matters, and if a node has just 1 child, it matters whether this is its left or right child
- **Each node holds a facet**
  - \* This may be only part of a facet from original scene
  - \* When constructing a BSP tree, we may need to split facets
- **Organization**
  - \* **Each facet lies in a unique plane**
    - ⇒ In 2-D, a unique line
  - \* **For each facet, we choose one side of its plane to be “outside”**  
Other direction: “inside”
    - ⇒ This can be the side the normal vector points toward
  - \* **Rule: For each node**
    - ⇒ Its left descendant subtree holds only facets “inside” it
    - ⇒ Its right descendant subtree holds only facets “outside” it

Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>



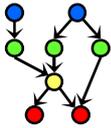


## Binary Space Partitioning Trees [3]: Construction

- To construct a BSP tree, we need
  - \* List of facets (with vertices)
  - \* “Outside” direction for each
- Procedure
  - \* Begin with empty tree
  - \* Iterate through facets, adding new node to tree for each new facet
  - \* First facet goes in root node.
  - \* For each subsequent facet, descend through tree, going left or right depending on whether facet lies inside or outside the facet stored in relevant node
    - ⇒ If facet lies partially inside & partially outside, split it along plane [line] of facet
    - ⇒ Facet becomes two “partial” facets
    - ⇒ Each inherits its “outside” direction from original facet
    - ⇒ Continue descending through tree with each partial facet separately
  - \* Finally, (partial) facet is added to current tree as leaf

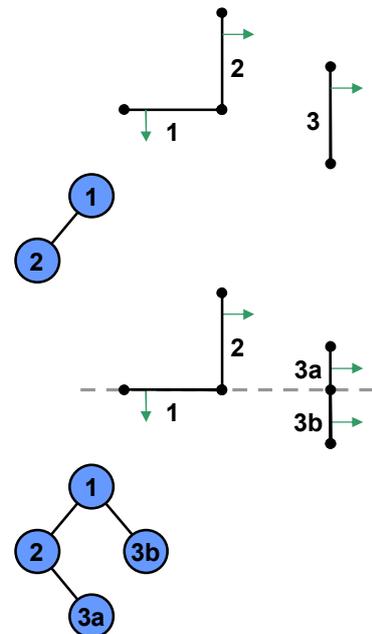
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





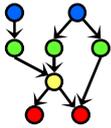
## Binary Space Partitioning Trees [4]: Simple Example

- Suppose we are given the following (2-D) facets and “outside” directions:
- We iterate through the facets in numerical order
  - \* Facet 1 becomes the root
  - \* Facet 2 is inside of 1
  - \* Thus, after facet 2, we have the following BSP tree:
- Facet 3 is partially inside facet 1 and partially outside.
  - \* We split facet 3 along the line containing facet 1
  - \* The resulting facets are 3a and 3b
  - \* They inherit their “outside” directions from facet 3
- We place facets 3a and 3b separately
  - \* Facet 3a is inside facet 1 and outside facet 2
  - \* Facet 3b is outside facet 1
- The final BSP tree looks like this:



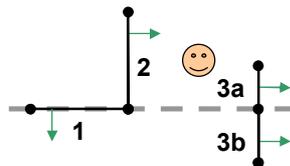
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





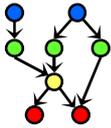
## BSP Tree Traversal [1]

- Important use of BSP trees: provide back-to-front (or front-to-back) ordering of facets in scene, from point of view of observer
  - \* When we say “back-to-front” ordering, we mean that no facet comes before something that appears directly behind it
  - \* This still allows nearby facets to precede those farther away
  - \* Key idea: All descendants on one side of facet can come before facet, which can come before all descendants on other side
- Procedure
  - \* For each facet, determine on which side of it observer lies
  - \* Back-to-front ordering: in-order traversal of tree where subtree opposite from observer comes before subtree on same side



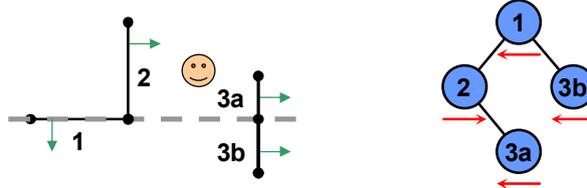
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





## BSP Tree Traversal [2]

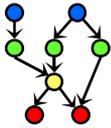
- Procedure:
  - \* For **each facet**, determine on which side of it the observer lies.
  - \* Back-to-front ordering: Do an in-order traversal of the tree in which the subtree opposite from the observer comes before the subtree on the same side as the observer.
- Our observer is inside 1, outside 2, inside 3a, outside 3b.



- Resulting back-to-front ordering: 3b, 1, 2, 3a.
- Is this really back-to-front?

Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>



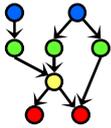


## BSP Trees: What Are They Good For?

- **BSP trees are primarily useful when a back-to-front or front-to-back ordering is desired:**
  - \* For HSR
  - \* For translucency via blending
- **Since it can take some time to construct a BSP tree, they are useful primarily for:**
  - \* Static scenes
  - \* Some dynamic objects are acceptable
- **BSP-tree techniques are generally a waste of effort for small scenes. We use them on:**
  - \* Large, complex scenes

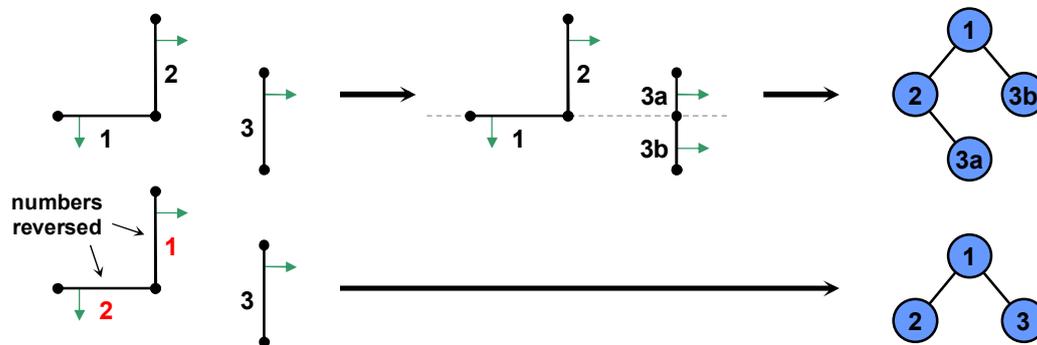
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





## BSP Tree Optimization

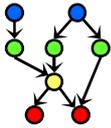
- Order in which we iterate through the facets can matter a great deal
  - Consider our simple example again
  - If we change the ordering, we can obtain a simpler BSP tree



- If a scene is not going to change, and the BSP tree will be used many times, then it may be worth a large amount of preprocessing time to find the best possible BSP tree

Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





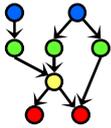
## BSP Trees: Finding Inside/Outside [1]

- When dealing with BSP trees, we need to determine inside or outside many times. What exactly does this mean?
  - \* A facet lies entirely on one side of a plane if all of its vertices lie on that side.
  - \* Vertices are points. The position of the observer is also a point.
  - \* Thus, given a facet and a point, we need to be able to determine on which side of the facet's plane the point lies.
- We assume we know the normal vector of the facet (and that it points toward the "outside").
  - \* If not, compute the normal using a cross product.
  - \* If you are using `vecpos.h`, and three non-colinear vertices of the facet are stored in `pos` variables `p1`, `p2`, `p3`, then you can find the normal as follows.

```
vec n = cross(p2-p1, p3-p1).normalized();
```

Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





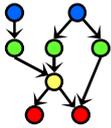
## BSP Trees: Finding Inside/Outside [2]

- To determine on which side of a facet's plane a point lies:
  - \* Let  $N$  be the normal vector of the facet
  - \* Let  $p$  be a point in the facet's plane
    - ⇒ Maybe  $p$  is a vertex of the facet?
  - \* Let  $z$  be the point we want to check
  - \* Compute  $(z - p) \cdot N$ 
    - ⇒ If this is positive, then  $z$  is on the outside
    - ⇒ Negative: inside
    - ⇒ Zero: on the plane
- Using `vecpos.h`, and continuing from previous slide:

```
pos z = ...; // point to check
if (dot(z-p1, n) >= 0.)
    // Outside or on plane
else
    // Inside
```

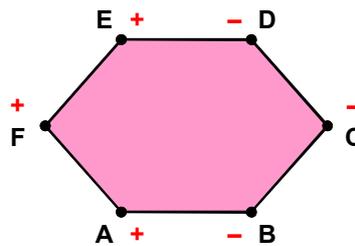
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





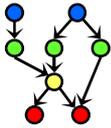
## BSP Trees: Splitting Polygons [1]

- May need to split facet when constructing BSP tree
- Example
  - ★ Suppose we have the facet shown below.
  - ★ If all vertices are (say) outside, then no split required
  - ★ But if A, E, and F are outside (+), and B, C, and D are inside (-), then we must split into two facets



Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





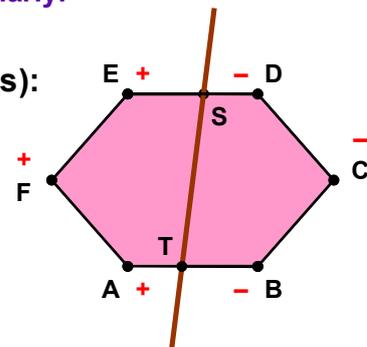
## BSP Trees: Splitting Polygons [2]

- **Where do we split?**

- \* Since the expression  $(z - p) \cdot N$  is positive at E and negative at D, it must be zero somewhere on the line segment joining D and E. Call this point S. This is one place where the facet splits.
- \* Let  $k_1$  be the value of  $(z - p) \cdot N$  at D, and let  $k_2$  be the value at E.
- \* Then  $S = (1/(k_2 - k_1)) (k_2 D - k_1 E)$ .
- \* Point T (shown in the diagram) is computed similarly.

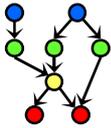
- **Using `vecpos.h` (continuing from earlier slides):**

```
double k1 = dot(D-p1, n);
double k2 = dot(E-p1, n);
pos S = affinecomb(k2, D, -k1, E);
// Explanation of above line?
```



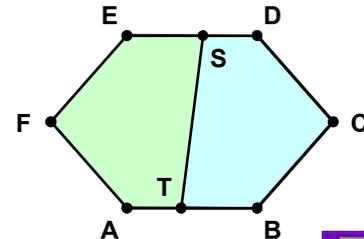
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





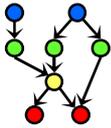
## BSP Trees: Splitting Polygons [3]

- We were given vertices A, B, C, D, E, F in order
- We computed S and T
  - \* S lies between D and E
  - \* T lies between A and B
- We have A, (split at T), B, C, D, (split at S), E, F
- We form two polygons as follows:
  - \* Start through vertex list
  - \* When we get to split, use that vertex, and skip to other split
  - \* Result: A, T, S, E, F
  - \* Do the same with the part we skipped
  - \* Result: B, C, D, S, T



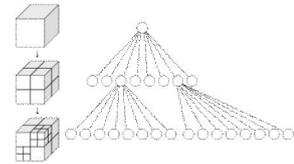
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





## Quadtrees & Octrees [1]: Background

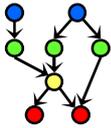
- Idea of binary space partition: good general applicability
- Variations used in several different structures
  - \* **BSP trees (of course)**
    - ⇒ Split along planes containing facets
  - \* **Quadtrees & octrees (next)**
    - ⇒ Split along pre-defined planes.
  - \* **K-d trees (Lecture 28)**
    - ⇒ Split along planes parallel to coordinate axes, so as to split up the objects nicely.
    - ⇒ *How about a project on K-d trees?*
- **Quadtrees** used to partition 2-D space; **octrees** are for 3-D
  - \* Two concepts are nearly identical
  - \* Unfortunate that they are given different names



Wikipedia, Octree  
<http://bit.ly/dVrthx>

Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>



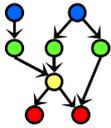


## Quadtrees & Octrees [2]: Definition

- **In general**
  - \* **Quadtree**: tree in which each node has at most 4 children
  - \* **Octree**: tree in which each node has at most 8 children
  - \* **Binary tree**: tree in which each node has at most 2 children
- **In practice, however, we use “quadtree” and “octree” to mean something more specific**
  - \* Each node of the tree corresponds to a square (quadtree) or cubical (octree) region
  - \* If a node has children, think of its region being chopped into 4 (quadtree) or 8 (octree) equal subregions
  - \* Child nodes correspond to these smaller subregions of parent’s region
  - \* Subdivide as little or as much as is necessary
  - \* Each internal node has exactly 4 (quadtree) or 8 (octree) children

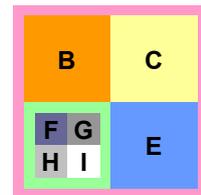
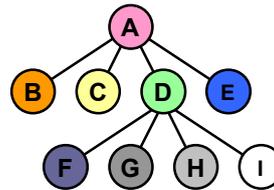
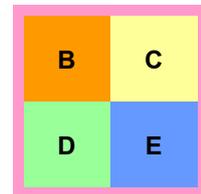
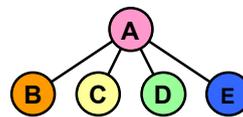
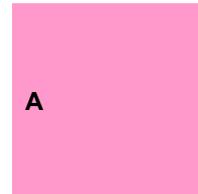
Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>





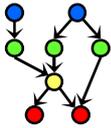
## Quadtrees & Octrees [3]: Example

- Root node of quadtree corresponds to square region in space
  - \* Generally, this encompasses entire “region of interest”
- If desired, subdivide along lines parallel to the coordinate axes, forming four smaller identically sized square regions
  - \* Child nodes correspond to these
- Some or all of these children may be subdivided further
- Octrees work in a similar fashion, but in 3-D, with cubical regions subdivided into 8 parts



Adapted from slides ♥ 2004 G. G. Chappell, UAF  
CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>



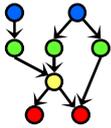


## Quadtrees & Octrees [4]: What Are They Good For?

- **Handling Observer-Object Interactions**
  - \* Subdivide the quadtree/octree until each leaf's region intersects only a small number of objects
  - \* Each leaf holds a list of pointers to objects that intersect its region
  - \* Find out which leaf the observer is in. We only need to test for interactions with the objects pointed to by that leaf
- **Inside/Outside Tests for Odd Shapes**
  - \* The root node represent a square containing the shape
  - \* If node's region lies entirely inside or entirely outside shape, do not subdivide it
  - \* Otherwise, do subdivide (unless a predefined depth limit has been exceeded)
  - \* Then the quadtree or octree contains information allowing us to check quickly whether a given point is inside the shape
- **Sparse Arrays of Spatially-Organized Data**
  - \* Store array data in the quadtree or octree
  - \* Only subdivide if that region of space contains interesting data
  - \* This is how an octree is used in the BLUisculpt program

Adapted from slides ♥ 2004 G. G. Chappell, UAF  
 CS 481/681: Advanced Computer Graphics, Spring 2004, <http://bit.ly/eivvVc>

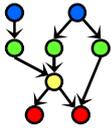




## Summary

- Reading for Last Class: §2.4.3, 8.1, Eberly 2<sup>e</sup>, **GL handout**
- Reading for Today: Chapter 6, Esp. §6.1, Eberly 2<sup>e</sup>
- Reading for Next Class: Chapter 7, §8.4, Eberly 2<sup>e</sup>
- Last Time: Collision Detection Part 1 of 2
  - \* Static vs. dynamic, testing vs. finding, distance vs. intersection
  - \* Triangle point containment test
  - \* Lots of intersections: spheres, capsules, lozenges
  - \* Method of separating axes
- Today: Adaptive Spatial Partitioning
  - \* Visible Surface Determination (VSD) revisited
  - \* Constructive Solid Geometry (CSG) trees
  - \* Binary Space Partitioning (BSP) trees
  - \* Quadtrees: adaptive 2-D (planar) subdivision
  - \* Octrees: adaptive 3-D (spatial) subdivision
- Coming Soon: Volume Graphics & Voxels





## Terminology

- **Collision Detection**
  - \* Static vs. dynamic objects
  - \* Queries: test-intersection vs. find-intersection
  - \* Parametric methods: distance-based, intersection-based
- **Bounding Objects**
  - \* Axis-aligned bounding box
  - \* Oriented bounding box: can point in arbitrary direction
  - \* Sphere
  - \* Capsule
  - \* Lozenge
- **Constructive Solid Geometry Tree: Regularized Boolean Set Operators**
- **Adaptive Spatial Partitioning: Calculating Intersection, Visibility**
  - \* Binary Space Partitioning tree – 2-way decision tree/surface
  - \* Quadtree – 4-way for 2-D
  - \* Octree – 8-way for 3-D

