Data Structures & Algorithms for Geometry

- Agenda:
  - Assignment #3, part 1 due
  - BSP tree overview
    - Node storing
    - Leaf storing
    - Solid-leaf storing
  - Creating BSP trees
    - Selecting & evaluating split planes
    - Classifying polygons w.r.t. split planes
    - Splitting polygons

Binary Space Partition Trees

- As the name implies, this is a binary tree where each node splits space.
  - Each node contains an *n*-dimensional split plane.
  - One child is the positive space of the split plane, and the other child is the negative space.
- Numerous applications:
  - Hidden surface removal
  - Constructive solid geometry (CSG)
  - Collision detection
  - Many, many more

Types of BSP trees

- Three common types of BSP tree:
  1. Node storing – object geometry stored in inner nodes and leaf nodes.
  2. Leaf storing – object geometry stored only in leaf nodes.
  3. Solid-leaf – positive leaf nodes represent empty space, negative leaf nodes represent solid space.

Node Storing

- Auto-partitioning
  - Meaning split-planes come from object geometry
- Each node stores all object surfaces that are coplanar with the split-plane.
- Used to be used for surface sorting for software rendering
  - Doom popularized this technique.
  - Not useful for hardware rendering.
  - Not good for collision detection, either.

Leaf Storing

- Each inner node stores only the split-plane
  - Can be auto-partitioning or general.
  - Polygons coplanar to split-planes must be consistently sent to the positive or negative space.
- Geometry stored only in leaf-nodes.
- Generalization of k-d trees, quadtrees, octrees, etc.

Solid-Leaf

- Used to represent the volume occupied by input geometry.
  - *Every* face must be used as a split-plane
  - Other planes can *also* be used as split-planes
- Very useful for collision detection
  - No need to perform polygon-polygon tests.
  - If part of the test polygon is in solid space, there is a collision.

Hybrid Solid-Leaf / Leaf-Storing

- Extends solid-leaf tree to store polygons in solid-space nodes.
  - Each node stores the polygons visible from that solid-space.
  - Other data can be stored to accelerate eventual AABB tests.
    - We'll talk about this next week.
- Popularized by Quake II and Quake III.
  - Called *brush-storing* trees in Quake terminology.

Construction Overview

- Only 3 steps:
  1. Select split-plane.
  2. Divide polygons into two groups based on split-plane.
     - This may include dividing polygons that straddle the split-plane.
  3. Repeat on each subgroup.
     - As with other subdivision trees, stop splitting when we have few enough polygons or reach a deep enough level.
     - May also stop if a good split-plane cannot be found.

Split-planes

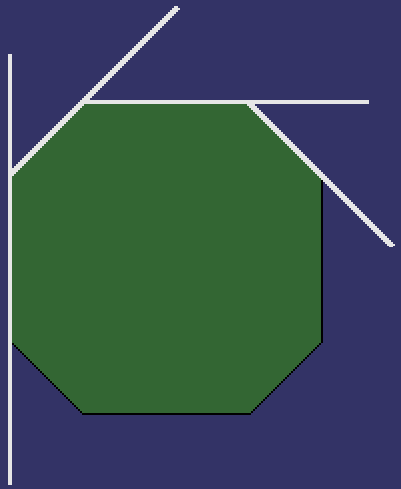
- Split-plane selection determines performance of final tree
  - Just like picking the subdivision of BV hierarchies.
  - Ideally we want O(log n) tree depth
    - For the same reasons as with regular binary search trees
- Two general partition strategies:
  - Auto-partition: select planes from the geometry
  - General: pick any arbitrary planes
  - Hybrid methods also exist
    - We'll talk more about these next week.

Auto-partition

- Generally easier to implement
  - All possible split-planes are known in advance
- May result in poor performance
  - Any selection of these planes will result in numerous polygon splits.
  - Selecting other planes initially may avoid splits.

Auto-partition (cont.)

- Auto-partition also performs poorly on convex objects.
  - By definition, all faces lie on one side of each possible plane.
  - The result is no splits, but O(n) tree depth.
- If the search goes outside early, the search may terminate faster than O(log n).



General Plane Selection

- A very hard problem.
  - Even harder than finding optimal OBB orientation
  - Need to narrow the search space

General Plane Selection

- A very hard problem.
  - Even harder than finding optimal OBB orientation
  - Need to narrow the search space
- Try some of the following:
  - Planes aligned to the axes
    - Like k-d trees
  - Planes through the edge of one polygon and a vertex of another
  - Allow user selected *hint planes.*
    - Humans can provide good possible planes early on

Evaluating Split-planes

- Need some metrics to determine which possible split-plane is best.

Evaluating Split-planes

- Need some metrics to determine which possible split-plane is best.
  - Minimize number of polygons split (a.k.a. *least-crossed*)
    - In the worst case, each split can create n new planes resulting in $O(n^2)$ total planes.
  - Balance number of polygons in each subtree.
    - Using least-crossed can lead to $O(n)$ tree depth.
- Reality: neither heuristic works well in isolation.
  - Use some linear combination of score from both

Polygon Classification

- Relative to a split-plane, a polygon can be:
    a. Completely in positive half-space

Polygon Classification

- Relative to a split-plane, a polygon can be:
  a. Completely in positive half-space
  b. Completely in negative half-space

Polygon Classification

- Relative to a split-plane, a polygon can be:
  a. Completely in positive half-space
  b. Completely in negative half-space
  c. Coplanar

Polygon Classification

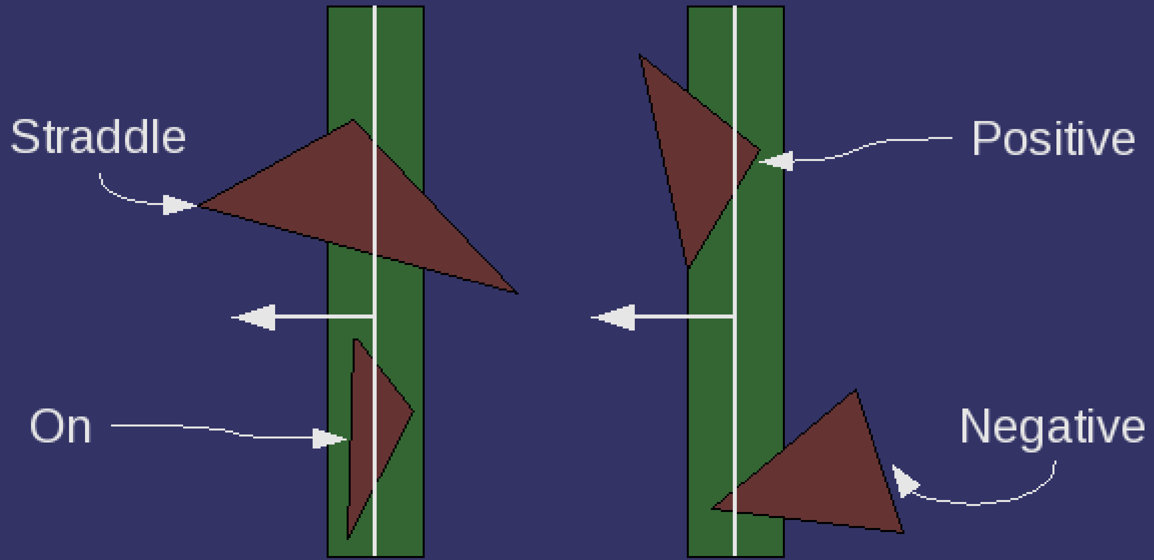
- Relative to a split-plane, a polygon can be:
  a. Completely in positive half-space
  b. Completely in negative half-space
  c. Coplanar
  d. Straddling plane

Polygon Classification

- Relative to a split-plane, a polygon can be:
  a. Completely in positive half-space
  b. Completely in negative half-space
  c. Coplanar
  d. Straddling plane
- Have to be careful with polygons that are "close" to the plane.
  - Floating point math is *not* an exact science. If points are close the the split-plane, the polygon splitting routine will produce erroneous results.

Thick Planes

- Solve the “close to plane” problem by making split-planes *thick*.
  - Points within some small distance, ε, of the plane are considered to be on the plane.



Polygon Splitting

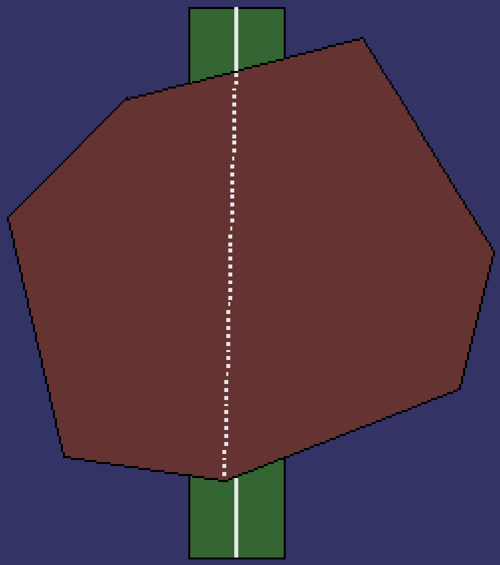
- Split using modified Sutherland-Hodgman polygon clipping
  - Classify each vertex as *in*, *out*, or *on.*
    - Use a 2-bit *out-code.*
  - Each edge is coded with the directed pair of its vertex out-codes.
  - Out-code pair determines what to do with the edge:
    - Add to outside polygon
    - Add to inside polygon
    - Split and add each half to one polygon

Polygon Splitting (cont.)

- In first 4 cases, order is unimportant:
  - Outside ↔ outside *or* outside ↔ on – add edge to outside polygon
  - Inside ↔ inside *or* inside ↔ on – add edge to inside polygon
- Only two cases remain:
  - Inside → outside
  - Outside → inside
    - Split edge. Add one half to each polygon.

Adding Edges

- Real brains of algorithm in code to add edges.
  - Easy (common) cases: Edge added to polygon shares vertex with last added edge, link the two edges.
  - Hard case: Edge added to polygon does not share vertex with last edge, insert new edge connecting them.
    - Care must be taken to handle case where last edge is split or has end-point on split-plane.



References

http://www.gamedev.net/reference/articles/article657.asp

- This is the mother of all BSP references!

http://symbolcraft.com/graphics/bsp/

- Interactive Java applet that builds & views BSP trees

http://en.wikipedia.org/wiki/BSP_tree

- Not too useful, but has links to other resources

Next week...

- BSP trees, part 2
  - Advanced split-plane selection
  - Intersection tests
- Assignment #3, part 2 due.
- Assignment #4.
- Quiz #3.

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Quake II and Quake III are trademarks of id Software.
- Other company, product, and service names may be trademarks or service marks of others.