Computer Graphics II

Autumn 2015-2016
Outline

1. Space Decomposition
The Need for Spatial Organisation

- We have seen how culling, lighting models, picking and collision detection all require some method of quickly eliminating from consideration parts of the scene that are irrelevant to the computation.

- We can use techniques such as bounding spheres or rectangles (nesting, if necessary) but other techniques can also be used:

  - **Binary Space Partition (BSP) Trees**, **Quadtrees**, **Octrees** and **K-d Trees** are some ways of managing a decomposition of the space.

- While these data structures were primarily designed to deal with cartesian space they can be used in more general settings where an item in a database can be classified according to a $k$-tuple.
BSP Trees

- In addition to the preceding applications, BSP Trees can be used as a method of improving rendering efficiency by deciding which objects appear closer to the screen than others.
- (Two alternatives are the painter’s algorithm and Z-buffering.)
- The central idea of BSP is to recursively divide a space into convex sets (no angle larger than $180^\circ$) by means of hyperplanes; following is courtesy of wikipedia.org.
Space is decomposed by using a **hyperplane** to split into two regions, one side “in front of” and one “behind”. Generalises to 3D, too.

Each region of space corresponds to a **leaf** of the BSP tree and in the applications described previously the **tree** can be used to determine the region; leaf node has linked list of assets in region.

Shape of resulting tree greatly influences its efficiency of traversal; much effort is put in to computing “good” decompositions so the initialisation process can be slow, although usually very fast thereafter.

BSP Trees can also be used for shadow generation in the presence of lights: Shadow Volume BSP (SVBSP)
Space Decomposition

Using BSPs for Scene Management
Quadtree decomposition

- In 2D the BSP tree can be made **shallower** by increasing the branching factor; this is what a **quadtree** does.
- By splitting the space into **four** cells (buckets) each time a faster data structure can result.
- Two types of quadtrees:
  - **Point quadtree** – points in the space are of interest.
  - **Region quadtree** – decompose space into equal regions.

(going left to right in tree is clockwise ordering of 4 regions)
Instead of points we could store other things, such as edges or polygons.

The process recurses as long as a bucket has a population above some given threshold.

Quadtrees are useful for decomposing landscapes and other “huge” areas into manageable sizes.
The 3-D analogue of the quadtree is the octant tree, or octree.

Octrees, like quadtrees, are sensitive to the particular data they are given.

Consider the octree that results from a set of 8 points, \((1, 1, 1)\) and the other 7 being grouped closely around \((10, 10, 10)\).

Compare with the octree based on the 8 points that are the corners of a cube.
**kd Trees**

- Although we are concerned with at most 3 dimensions, sometimes data is categorised according to several (> 3) dimensions.
- $k$-d trees are a generalisation of octrees to $k$ dimensions.
- They also have the important property that they are sensitive to the layout of the data:

(Images courtesy of wikipedia.org.)
The hyperplane (split line) is chosen so that as close as is possible to half of the points are on either side.

While more balanced trees can result from \( kd \) trees “someone” needs to find the best balanced tree.