CISC 829 Computational Geometry

HW2 due April 15

WRITTEN EXERCISES:

Problem 1: Textbook, exercise 3.13, page 61.

Problem 2: Textbook, exercise 5.4a,b,c, p. 117.

Problem 3: Textbook, exercise 8.7, p. 181.

PROGRAMMING EXERCISES:

The programming exercise this time has these objectives:

1. Gain familiarity with manipulating higher-order geometric primitives;
2. Interleave interaction, computation and display;
3. Produce an “inspection" capability for construction and display of a data structure;
4. Get some hands-on experience with an interesting geometric data structure.

The data structure we will use for the exercise is BSP trees, but the development will use DCELs and ideas from point location and arrangement walking as well.

You will need to use the basic interface implemented in PS1. There are some public-domain Java BSP implementations out there, which you might want to look at as well, but you should write the BSP code yourself.

This exercise will end up with a set of methods for incremental construction and querying of a 2D BSP tree of 2D line segments. Here are the suggested steps (components) for how you might proceed.

- Define a Vertex class to represent a 2D point (x; y).
- Define a Halfspace class to represent the halfspace $ax + by + c > 0$.
- Define a Segment class to represent a 2D line segment with two Vertex endpoints P and Q, and the Halfspace defined by the left side of the line directed from P to Q.
- Define a Face class (after the DCEL shown in the text, and described several times in class) to represent a planar subdivision. Make your constructor initialize the Face to (the CCW contour of) a 2D bounding box.
- Define a BSPCell class, which has a Face, and (if not the root) a pointer to the parent BSPCell, and (if not a leaf) a Segment upon which the Cell has been split, and two (can be NULL) pointers to child BSPCells.

- Define a colorful .Draw() method for each class above. (You can assume that a Halfspace is always accompanied by a Segment, so you know where to draw it.)

- Define Segment.Clip(H), which clips the segment to a given halfspace H.

- Define BSPCell.Split(S), which splits the (leaf) cell by a given segment S.

- Define BSPCell.Insert(S), which inserts a given segment S into the BSP tree. There are several ways to do this. Here's a suggestion for a recursive, top-down definition of Insert():

  ```
  Insert(BSP Cell B, Segment S)
    Clip S to B
    If it clips out, return
    if B is a leaf
        Split B by S
    else
        Insert S into B's children
  ```

  Now initialize a root BSPCell with a Face that has been initialized to the 2D bbox (say, your working screen area). Define a segment entry method (probably the easiest is to mouse-down, sweep out the segment, then mouse up), and Insert each generated Segment into the BSP tree.

- Define BSPCell.Locate(P), which identifies the cell containing a given point P. Put an eye-point icon E (say, a point from PS1) into the scene, and make it selectable and draggable by the mouse. Point-locate on E, and highlight the resulting cell.

- Define BSPCell.Traverse(P), which generates a back-to-front ordering of the Segment fragments in the BSP tree.

- Draw the fragments. Assign each Segment a unique color and \render" the scene onto the view-line (analogous to the view-plane, in 3D) of the eyepoint.