Computer Graphics ***Instructor: Zoran Popović***

CSE P 557, Autumn 2010

Homework #2

Hidden Surfaces, Shading, Texture Mapping, Parametric Curves

Assigned: Wed, November 24, 2010
Due: Tuesday, Dec 7, 2010, **at the beginning of class**

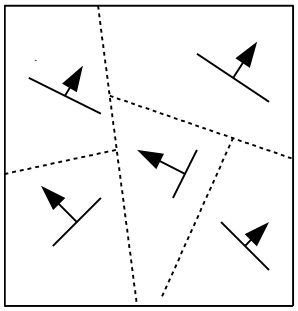
Directions: Please provide short written answers to the questions in the space provided. If you require extra space, you may staple additional pages to the back of your assignment. Feel free to discuss the problems with classmates, but please ***answer the questions on your own*.**

Name:_____

Problem 1: BSP Trees (15 points)

BSP trees are widely used in computer graphics. Many variations can be used to increase performance. The following questions deal with some of these variations.

For the version of BSP trees that we learned about in class, polygons in the scene (or more precisely, their supporting planes) were used to do the scene splitting. However, it is not necessary to use existing polygons – one can choose arbitrary planes to split the scene:



a. (5 pts) What is one advantage of being able to pick the plane used to divide the scene at each step? What is one disadvantage of not just using existing polygons? Draw a two dimensional example to illustrate your point for each case.

Problem 1 - continued

Recall that when using a BSP tree as described in class, we must draw *all* the polygons in the tree. This is inefficient, since many of these polygons will be completely outside of the view frustum. However, it is possible to store information at the internal nodes in a BSP tree that will allow us to easily determine if any of the polygons below that node will be visible. If none of the polygons in that sub-tree will be visible, we can completely ignore that branch of the tree.

b. (5pts) Explain what extra information should be stored at the internal nodes to allow this, and how it would be used to do this "pruning" of the BSP tree.

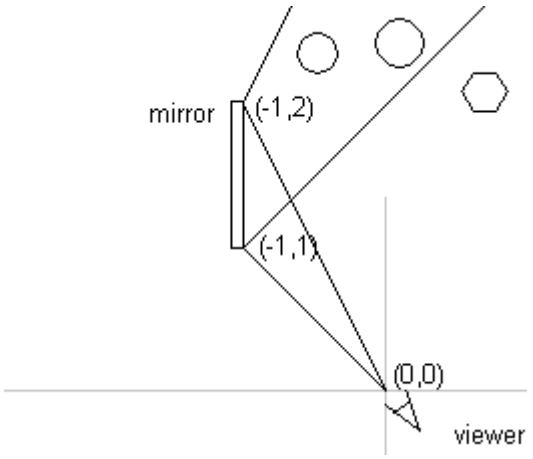
c. (5 pts) In class, we talked about doing a "back to front" traversal of a BSP tree. But it is sometimes preferable to do a "front to back" traversal of the tree, in which we draw polygons closer to the viewer before we draw the polygons farther away. How should the tree traversal order be changed in order to do a front to back traversal?

Problem 2. More uses of texture mapping (20 points)

Texture mapping can be used in many ways in addition to simply providing realistic looking surface material treatments on objects.

For example, the texture applied to a surface doesn't have to be a representation of the surface material at all. It could be an image rendered from another point of view in the model.

a. (3 pts) Consider the scene shown in the figure. The viewer is at the origin, and there is a perfect ***planar*** (flat) mirror sitting in the scene nearby. Think about the image in the mirror. Calculate where you would place another center of projection (viewer) in order to render the exact image that is visible in the mirror from the viewer at the origin, and draw it on the diagram. Show the coordinates of the added center of projection.



b. (5 pts) Describe a general procedure for calculating a mirror's viewpoint given a scene, a planar mirror polygon, and a viewer positioned at the origin. Describe your steps in words.

Problem 2 – continued

c. (6 pts) The psuedocode below describes the basic Z-buffer algorithm we discussed in class.

```
for each pixel (i, j) do {
       Z-buffer[i,j] = FAR
       Framebuffer[i,j] = background color
}
for each polygon A do {
       for each pixel (i,j) in A do {
              z = compute depth of A at (i,j)
              s = shade of A at (i,j)
              if z < Z-buffer[i,j] {
                     Z-buffer[i,j] = z
                     Framebuffer[i,j] = s
              }
       }
}
```

Consider the function **image = RenderMirror(polygon *mirror*)** that takes in a polygon ***mirror*** and renders the scene as it would be seen by the mirror, returning the rendered **image**. Write a pseudocode for **RenderMirror**, using a modified version of the Z-buffer algorithm above. To simulate a simple mirror, your algorithm should disregard cases such as infinite reflections from other mirrors, and instead just render the objects in front of the mirror. Feel free to reference your explanation in part b) instead of writing out detailed code for viewpoint transformations.

Problem 2 – continued

d. (6 pts) Using your function **RenderMiror** from part c), write an algorithm pseudocode that can render a scene that includes simple mirrors. Your algorithm should basically be a modified version of the Z-buffer algorithm.

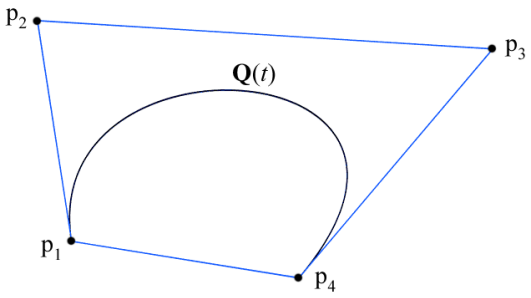
Problem 3: Parametric Curves (4 points)

Bezier curves are very simple and so versatile that almost all graphics packages feature a Bezier curve tool. Now suppose you bought **BezierDraw**, a graphics program that only has a Bezier curve tool and nothing else. (Assume that all curves that **BezierDraw** creates are third-order Beziers.)

a. (4 pts) Is it possible to draw a perfect circle with **BezierDraw**? Explain **in detail** why or why not. (Hint: what's the parametric formulation of a circle?)

Problem 4 - Properties of Bezier Curves (15 points)

A nice property of Bezier curves is that the curve itself will always remain within the *convex hull* of its control points. The convex hull of a set of points is defined as the smallest convex polygon containing all those points. Intuitively, you might imagine the convex hull of a set of points in two dimensional space to be the polygon defined by wrapping a rubber band around those points. In three dimensional space, imaging using a rubber sheet instead.



An intuitively true property about convex hulls is as follows. Suppose we are given *n* points; call these $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$. Now suppose we are given *n* real numbers, $w_1, w_2, \ldots, w_n$. If $0 \le w_i \le 1$ for all $1 \le i \le n$ and $w_1 + w_2 + \ldots + w_n = 1$, then $\mathbf{q} = w_1\mathbf{p}_1 + w_2\mathbf{p}_2 + \ldots + w_n\mathbf{p}_n$ lies within the convex hull of the points $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$. In other words, taking a weighted average of a set of points necessarily gives a point within the convex hull of those points.

a. (3 Points) A point on a cubic Bezier curve can be defined by the function

$$\mathbf{Q}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \mathbf{p}_4 \end{bmatrix}$$

where $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$ are the control points of the curve and $0 \le t \le 1$. Write out the Bezier basis functions $f_1(t),\ f_2(t),\ f_3(t),\ f_4(t)$ such that

$$\mathbf{Q}(t) = f_1(t)\mathbf{p}_1 + f_2(t)\mathbf{p}_2 + f_3(t)\mathbf{p}_3 + f_4(t)\mathbf{p}_4 .$$

Problem 4 - continued

b. (4 Points) Show that $f_1(t) \geq 0,\ f_2(t) \geq 0,\ f_3(t) \geq 0,\ f_4(t) \geq 0$ for all $0 \leq t \leq 1$.

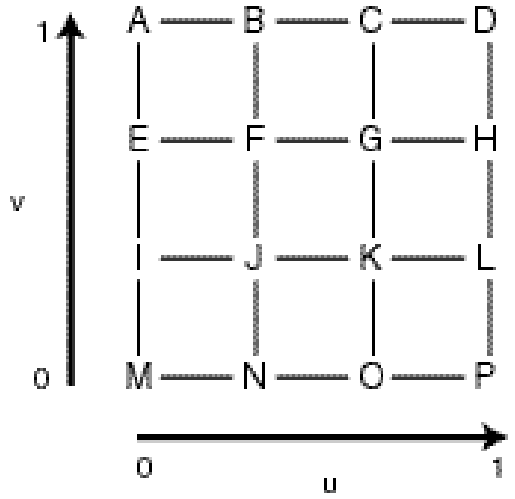
c. (3 Points) Show that $f_1(t) + f_2(t) + f_3(t) + f_4(t) = 1$ for all $0 \leq t \leq 1$.

d. (2 Point) Using the property about convex hulls stated previously, argue that any Bezier curve must lie within the convex hull of its control points. (Make sure you use the convex hull property exactly as it is stated)

e. (3 Points) Give an example of a situation in which the convex hull property of Bezier curves might be useful.

Problem 5. Surfaces (10 points)

Below is a diagram indicating the sixteen control points for specifying a cubic Bézier tensor product surface.



a) **(2 points)** Which control points are guaranteed to be interpolated by the surface?

b) **(4 points)** Which control points determine the normal (**hint**: see Problem 1) to the surface at u=0, v=0? Justify your answer.

c) **(4 points)** Which control points determine the normal to the surface at u=1/2, v=1/2? Justify your answer.