CS 465 Homework 2

out: Wednesday 5 September 2007
due: Monday 17 September 2007

Problem 1: [Viewing]

Consider a camera eye positioned at $\mathbf{e}$ in the world coordinate system, and pointing in the unit view direction $\mathbf{d}$.

1. Given the unit "up vector" $\mathbf{n}$, *give an expression for the orthonormal camera basis* $(\mathbf{u}, \mathbf{v}, \mathbf{w})$, assuming that $\mathbf{w}$ points in the opposite direction of $\mathbf{d}$, and that $\mathbf{v}$ has maximal component in the up direction.

2. Consider an image region located z units from $\mathbf{e}$ along $\mathbf{d}$ (and perpendicular to it). Assuming $\mathbf{d}$ passes through the center of the image, and that the image has horizontal length ℓ, and vertical height h, *what are the positions of the four corners of the image in the world coordinate system?*

Problem 2: [Surface shading]

Answer the following questions from Shirley et al., Chapter 9:

1. The moon is poorly approximated by diffuse or Phong shading. What observations tell you this is true?

2. Velvet is poorly approximated by diffuse or Phong shading. What observations tell you this is true?

Problem 3: [Ray intersection with quadrics]

You know how to intersect a ray with a sphere, but it turns out that this is just a special case of a more general surface intersection problem. *Quadric surfaces* are a class of shapes which include spheres, as well as ellipsoids, cylinders, paraboloids and hyperboloids. They are defined as a surface in 3-space consisting of points satisfying a polynomial of degree 2. In particular, they are defined as the implicit surface $f(x, y, z) = 0$ where

$$f(x, y, z) = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Jz + K.$$

1. Given a ray $\mathbf{e} + t\mathbf{d}, \quad t \in (0, \infty)$, clearly describe how to intersect a ray with a quadric surface and determine the first ray intersection, or that none exists.

2. For shading you also need to know the surface normal. Describe how to compute the unit surface normal, and give an expression for it.

Problem 4: [Point-in-polygon test]

In ray tracing and other graphics applications, we often need to determine if a ray hits a planar polygon. This can be done by seeing if the ray hits the polygon's plane, and if so, then projecting the point and polygon vertices to an appropriate plane, and determining whether the projected 2D point is inside the projected 2D polygon.

One of those methods to determine whether a point is inside a general polygon in 2D is shown in Figure 1. We send a ray out from **p** and count the number of intersections between the ray and the boundary of the polygon. If the number of intersections is odd, then the point is inside the polygon, and otherwise it is not.

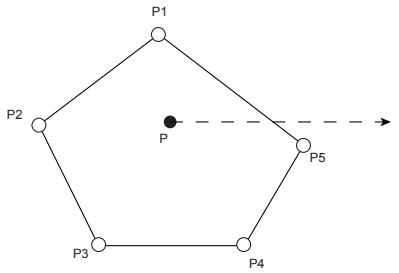

Figure 1: Shooting a ray to determine whether a point is inside a polygon.

However, in most applications we break concave polygons into convex polygons for efficiency. There is a simple way to determine if a point is inside a convex polygon, as shown in Figure 2. We walk around the polygon counterclockwise(or clockwise), if the point is always to the same side of the edge we are walking on, the point must be inside the polygon.

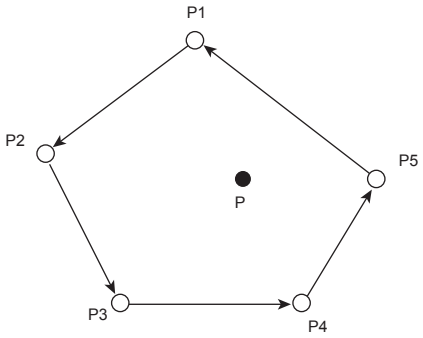

Figure 2: Walking around the polygon to determine if the point is inside.

In this question you will write pseudocode to determine if a point is inside a convex polygon in 2D (*Hint: You can utilize the fact that the cross product of two vectors is the signed area of the parallelogram defined by them.*):

1. First, define a predicate function of the signature
   `boolean LeftTurn(Point p1, Point p2, Point p3)`
   to decide if a point p is to the left side of a (directed) line segment $\overline{p_1 p_2}$.
2. Next use this function to write an algorithm to determine if a point is inside a polygon. You can assume the class/structure `Point` has appropriate fields and the points in the polygon are ordered in some way so that you can easily iterate in the counterclockwise order.