Homework #2

Hidden Surfaces, Projections, Shading, Ray Tracing, and Texture Mapping

Assigned: Tuesday, May 10th

Due: Wednesday, May 18th
at the beginning of class

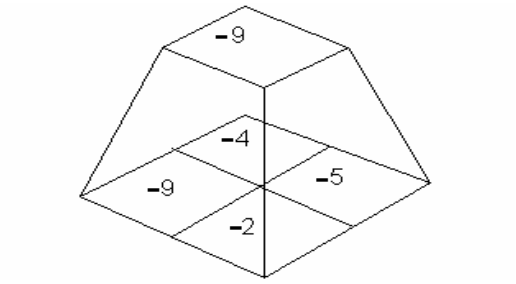
Directions: Please provide short written answers to the following questions, using this page as a cover sheet. Be sure to justify your answers when requested. Feel free to discuss the problems with classmates, but please ***answer the questions on your own*.**

Be sure to write your name on your homework solution.
You may (optionally) use this page as a cover sheet.

Name:_____

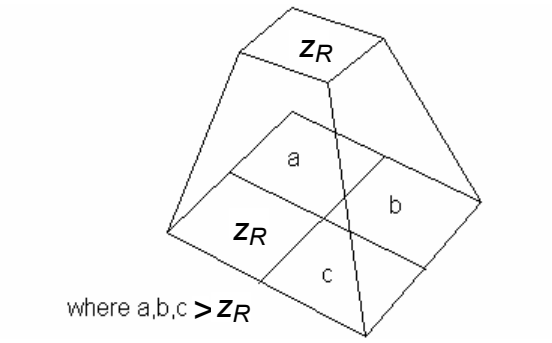
Problem 1. Z-buffer (9 points)

The z-buffer algorithm can be improved by using an image space "z-pyramid." The basic idea of the z-pyramid is to use the original z-buffer as the finest level in the pyramid, and then combine four *z*-values at each level into one *z*-value at the next coarser level by choosing the farthest (most negative) z from the observer. Every entry in the pyramid therefore represents the farthest (most negative) *z* for a square area of the z-buffer. In this problem, assume the image is always square with side length that is a power of 2. (Handling non-square, non-powers-of-2 images is a simple generalization of this.) Before each primitive is rendered, the z-pyramid is updated to reflect the current state of the z-buffer. When you have a new primitive to draw, you are then testing against the current, up-to-date z-pyramid. A z-pyramid for a single 2x2 image is shown below:



a) (2 points) At the coarsest level of the z-pyramid there is just a single *z* value. What does that *z* value represent?

Suppose we wish to test the visibility of a triangle **T**. Let z_T be the nearest *z* value of triangle **T**. **R** is a region on the screen that encloses the triangle **T**, and is the smallest region of the z-pyramid that does so. Let z_R be the *z* value that is associated with region **R** in the z-pyramid.



b) (2 points) What can we conclude if $z_R < z_T$?

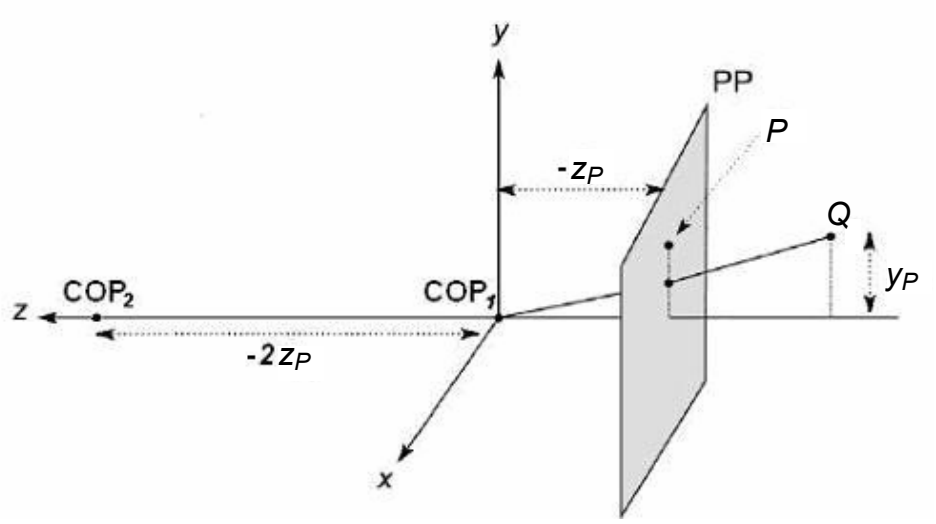
c) (2 points) What can we conclude if $z_T < z_R$?

If the visibility test is inconclusive, then the algorithm applies the same test recursively: it goes to the next finer level of the pyramid, where the region **R** is divided into four quadrants, and attempts to prove that triangle **T** is hidden in each of the quadrants of **R** that **T** intersects. Since it is expensive to compute the closest *z* value of **T** within each quadrant, the algorithm just uses the same z_T (the nearest *z* of the entire triangle) in making the comparison in every quadrant. If, at the bottom of the pyramid, the test is still inconclusive, the algorithm resorts to ordinary z-buffered rasterization to resolve visibility.

d) (3 points) Suppose that, instead of using the above algorithm, we decided to go to the expense of computing the closest *z* value of **T** within each quadrant. Finding the closest value amounts to clipping the triangle to each region and analytically solving for the closest z. This approach also applies to the finest level of the pyramid, where the pixels are abutting square regions. Would it then be possible to always make a definitive conclusion about the visibility of **T** within each pixel, without resorting to rasterization (effectively intersecting the viewing ray with the triangle)? Why or why not?

Problem 2. Projections (11 points)

Imagine there is a pinhole camera located at the origin, COP_1, that is looking in the $-z$ direction. The projection plane (PP) is the plane $z = z_P$ (note z_P is a negative number), so that the distance from COP_1 to PP is $d = -z_P$, as shown in the figure below. Let there be two points in the scene, $P = [0\ y_P\ z_P\ 1]^T$ and $Q = [0\ y_P\ 2z_P\ 1]^T$.



The projection matrix for this camera is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/z_p & 0 \end{bmatrix}$$

This projects P to the point $[0\ y_P\ 1]^T$ and Q to the point $[0\ \tfrac{1}{2}y_P\ 1]^T$.

a) (3 points) Now, assume that the camera has moved to COP_2 (shown above) at $[0\ 0\ -2z_P\ 1]^T$. Assume PP stays at $z = z_P$. Derive the new projection matrix that maps points onto PP. Show your work.

b) (2 points) If your matrix in part a) is correct, point P should project to the same image point as before. Calculate the projection of point Q. How did the projection of point Q change when the camera moved from COP_1 to COP_2?

c) (4 points) Suppose we want to keep the projection of Q constant at $[0\ \tfrac{1}{2}y_P\ 1]^T$. Suppose the center of projection is at $COP = [0\ 0\ z_{COP}\ 1]^T$. To keep the projection of Q constant, we will need to vary the z-coordinate of PP; let the updated PP be $z = \tilde{z}_p$. Solve for $\tilde{z}_p$ needed to keep Q's projection constant as z_{COP} varies. Show your work. Note that the z-coordinate of Q is fixed at $2z_P$, but the z-coordinate of PP is now a variable, $\tilde{z}_p$.

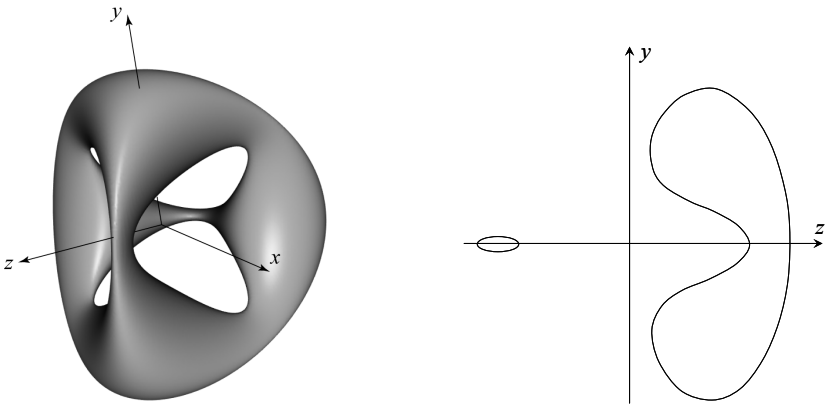
d) (2 points) Now consider moving the COP infinitely far back along the positive z-axis while keeping PP at its original location, $z = z_P$. Derive the new projection matrix for this case. Show your work. What is this sort of projection called?

Problem 3. Ray intersection with implicit surfaces (25 points)

There are many ways to represent a surface. One way is to define a function of the form $f(x,y,z)=0$. Such a function is called an *implicit surface* representation. For example, the equation $f(x,y,z)=x^2+y^2+z^2-r^2=0$ defines a sphere of radius *r*. Suppose we wanted to ray trace a "quartic chair," described by the equation:

$$(x^2+y^2+z^2-ak^2)^2-b\left[(z-k)^2-2x^2\right]\left[(z+k)^2-2y^2\right]=0$$

On the left is a picture of a quartic chair, and on the right is a slice through the *y*-*z* plane.



For this problem, we will assume a = 0.95, b = 0.8, and k = 5.

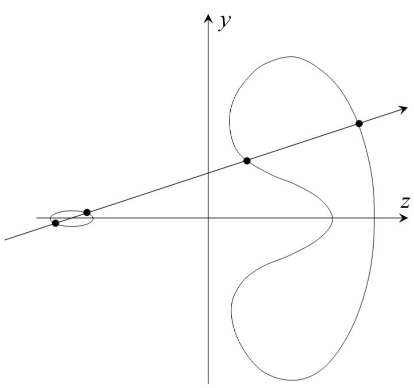
In the next problem steps, you will be asked to solve for and/or discuss ray intersections with this primitive. Performing the ray intersections will amount to solving for the roots of a polynomial, much as it did for sphere intersection. For your answers, you need to keep a few things in mind:

- You will find as many roots as the order (largest exponent) of the polynomial.
- You may find a mixture of real and complex roots. When we say complex here, we mean a number that has a non-zero imaginary component.
- All complex roots occur in complex conjugate pairs. If $A + iB$ is a root, then so is $A - iB$.
- Sometimes a real root will appear more than once, i.e., has multiplicity > 1. Consider the case of sphere intersection, which we solve by computing the roots of a quadratic equation. A ray that intersects the sphere will usually have two distinct roots (each has multiplicity = 1) where the ray enters and leaves the sphere. If we were to take such a ray and translate it away from the center of the sphere, those roots get closer and closer together, until they merge into one root. They merge when the ray is tangent to the sphere. The result is one distinct real root with multiplicity = 2.

a) (10 points) Consider the ray $P+t\mathbf{d}$, where $P=\begin{pmatrix}0 & 0 & 0\end{pmatrix}$ and $\mathbf{d}=\begin{pmatrix}0 & 0 & 1\end{pmatrix}$. Solve for all values of *t* where the ray intersects the quartic chair (including negative values of *t*). Which value of *t* represents the intersection we care about for ray tracing? In the process of solving for t, you will be computing the roots of a polynomial. How many distinct real roots do you find? How many of them have multiplicity > 1? How many complex roots do you find?

Problem 3 (cont'd)

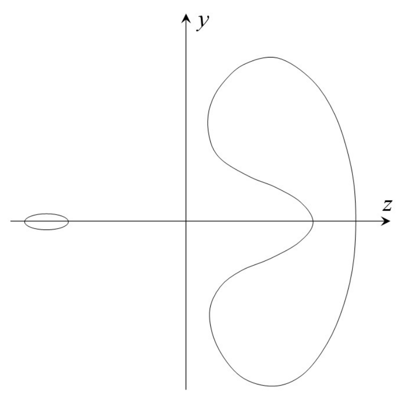
b) (15 points) What are all the possible combinations of roots, not counting the one in part (a)? For each combination, describe the 4 roots as in part (a), draw a ray in the *y*-*z* plane that gives rise to that combination, and place a dot at each intersection point. There are five diagrams below that have not been filled in. You may not need all five; on the other hand, if you can actually think of more distinct cases than spaces provided, then we might just give extra credit. The first one has already been filled in. (Note: not all conceivable combinations can be achieved on this particular implicit surface. For example, there is no ray that will give a root with multiplicity 4.) ***Please write on this page and include it with your homework solution. You do not need to justify your answers.***



\# of distinct real roots: **4**

\# of real roots w/ multiplicity > 1: **0**

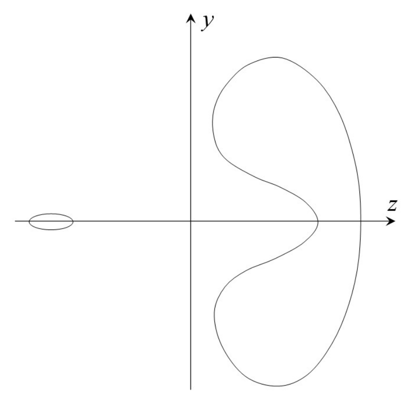
\# of complex roots: **0**



\# of distinct real roots:

\# of real roots w/ multiplicity > 1:

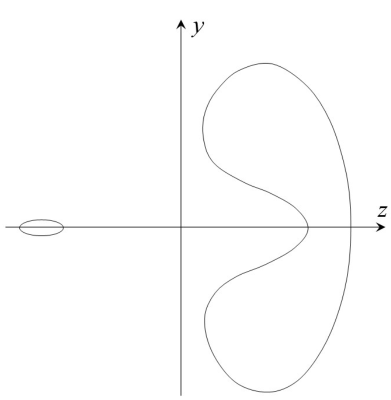
\# of complex roots:



\# of distinct real roots:

\# of real roots w/ multiplicity > 1:

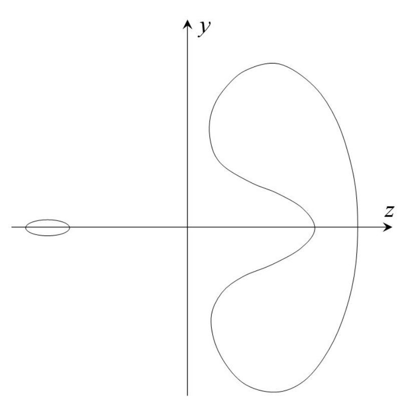
\# of complex roots:



\# of distinct real roots:

\# of real roots w/ multiplicity > 1:

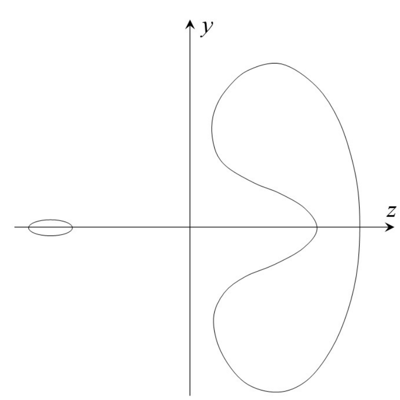
\# of complex roots:



\# of distinct real roots:

\# of real roots w/ multiplicity > 1:

\# of complex roots:



\# of distinct real roots:

\# of real roots w/ multiplicity > 1:

\# of complex roots:

Problem 4. Counting rays (25 points)

In this problem, we study the number of rays traced for using different ray tracing algorithms. Consider the following setup:

m x *m* pixels
k x *k* supersampling
n geometric primitives
ℓ light sources
d bounces (reflections and/or refractions)

For each of the algorithms and scenarios discussed in parts (a)-(e) below, assume the following:

- You are counting rays cast, including primary rays, shadow (light) rays, reflected rays, and (when asked for in the problem) refracted rays.
- *No* acceleration techniques are used.
- *Every* recursively traced (reflected or refracted) ray hits an object, including the primary rays.
- You will always cast a ray to the light source after intersecting an object, and this does not count as a recursive "bounce" (but certainly counts as a cast ray).
- Each ray cast to a light source counts as a single ray-cast, even when accounting for transparent shadows. (The transparent shadow case can be handled by keeping track of all intersections encountered – not just the closest – when casting a ray to a light, so this is a reasonable assumption.)

Explain your steps in arriving at answers to the questions below. For each sub-problem, in some cases, you can write out a closed form solution directly, but you must explain your reasoning. In other cases, you might need to write out a summation (with the Σ symbol for the summation); if possible, convert the summation to a closed form answer.

a) (5 points) For Whitted ray tracing, assuming reflection (but *no* refraction) at every surface, how many rays are cast?

b) (5 points) For Whitted ray tracing, assuming reflection *and* refraction at every surface, how many rays are cast?

c) (5 points) Suppose now, in order to get glossy reflections, you recursively cast *k* x *k* rays around the reflection direction at each bounce. Assuming glossy reflection (but *no* refraction) at every surface, how many rays are cast?

d) (5 points) In addition, in order to get translucent (blurry) refraction effects, you recursively cast *k* x *k* rays around the refraction direction at each bounce. Assuming glossy reflection and translucent refraction at every surface, how many rays are cast?

e) (5 points) Suppose now you switch to using distribution ray tracing. Assuming glossy reflection and translucent refraction at every surface, how many rays are cast?

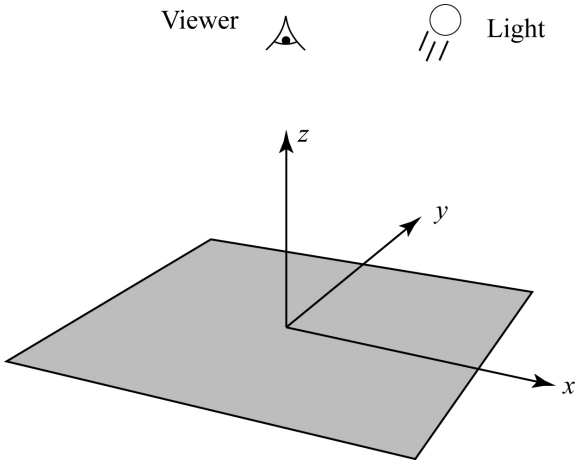
Problem 5. Shading, displacement mapping, and normal mapping (30 points)

In this problem, an opaque surface will be illuminated by one directional light source and will reflect light according to the following Phong shading equation:

$$I = A_{shadow} L\left(k_d \left(\mathbf{N}\cdot\mathbf{L}\right)_+ + k_s B\left(\mathbf{V}\cdot\mathbf{R}\right)_+^{n_s}\right)$$

Note the inclusion of a shadowing term, which takes on a value of 0 or 1. For simplicity, we will assume a monochrome world where I, L, k_d, and k_s are scalar values.

Suppose a viewer is looking down at an infinite plane (the x-y plane) as illustrated below. The scene is illuminated by a directional light source, also pointing straight down on the scene.



Answer the following questions below, giving brief justifications of each answer. Note that lighting and viewing directions are from the point of view of the light and viewer, respectively, and need to be negated when considering the surface-centric shading equation above. [In general, you don't need to solve equations and precisely plot functions. It is enough to describe the variables involved, how they relate to each other, and how this relationship will determine, e.g., the appearance of the surface. If you're more comfortable making the answers analytical with equations and plots, however, you are welcome to do so.]

a) (2 points) Assume: Perspective viewer at (0,0,1) looking in the (0,0,-1) direction, angular field of view of 90 degrees, lighting direction of (0,0,-1), $k_d = 0.5$, $k_s = 0$. Describe the brightness variation over the image seen by the viewer. Justify your answer.

b) (2 points) Assume: Perspective viewer at (0,0,1) looking in the (0,0,-1) direction, , angular field of view of 90 degrees, lighting direction of (0,0,-1), $k_d = 0.5$, $k_s = 0.5$, $n_s = 10$. Describe the brightness variation over the image seen by the viewer. Justify your answer.

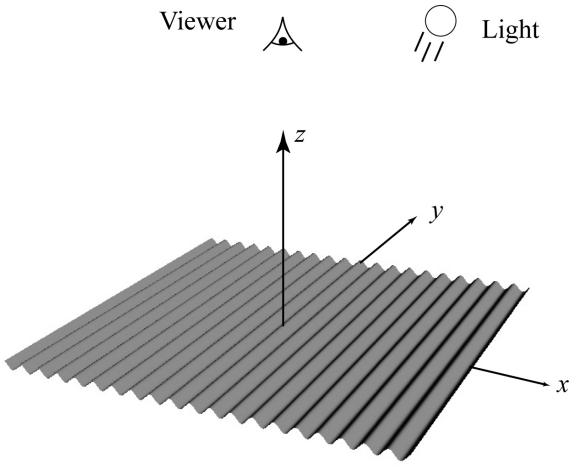
c) (2 points) Assume: Orthographic viewer looking in the (0,0,-1) direction, lighting direction of (0,0,-1), $k_d = 0.5$, $k_s = 0.5$, $n_s = 10$. Describe the brightness variation over the image seen by the viewer. Justify your answer.

d) (3 points) Assume: Orthographic viewer looking in the (0,0,-1) direction, $k_d = 0.5$, $k_s = 0$. The lighting direction starts at (-sqrt(2)/2 ,0, -sqrt(2)/2) and then rotates around the z-axis. Describe the brightness variation over time, as seen by the viewer. Justify your answer.

Problem 3. (cont'd)

e) (3 points) Assume: Orthographic viewer looking in the (0,0,-1) direction, $k_d = 0.5$, $k_s = 0.5$, $n_s = 10$. The lighting direction starts at (-sqrt(2)/2 ,0, -sqrt(2)/2) and then rotates around the z-axis. Describe the brightness variation over time, as seen by the viewer. Justify your answer.

Suppose now the infinite plane is replaced with a surface $z = \cos(x)$:



We can think of this as simply adding a displacement d=cos(x) in the normal direction to the x-y plane.

f) (5 points) Assume: Orthographic viewer looking in the (0,0,-1) direction, lighting direction of (0,0,-1), k_d = 0.5, k_s = 0. At what values of x is the surface brightest? At what values is it dimmest? Describe the appearance of the surface. Justify your answers.

g) (5 points) Assume: Orthographic viewer looking in the (0,0,-1) direction, lighting direction of (0,0,-1), k_d =0, k_s =0.5, n_s =10. At what values of x is the surface brightest? Describe the appearance of the surface. How does the appearance change as n_s increases to 100? Justify your answers.

Suppose now that we simply keep the normals used in (f)-(g) and map them over the plane from the first part of the problem. The geometry will be flat, but the shading will be based on the varying normals.

h) (5 points) Assume: Orthographic viewer looking in the (0,0,-1) direction, k_d =0.5, k_s =0. If we define the lighting to have direction (-sinθ, 0, -cosθ), will the normal mapped rendering look the same as the displacement mapped rendering for each of θ = 0, 10, and 80 degrees? Justify your answer.

i) (3 points) Assume: Orthographic viewer, lighting direction of (0,0,-1), k_d =0.5, k_s =0. As we generally move the viewer around – rotating it to various viewing direcitons – will the normal mapped rendering look the same as the displacement mapped rendering? Justify your answer.