Homework #2

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

Assigned: Saturday, May 5th

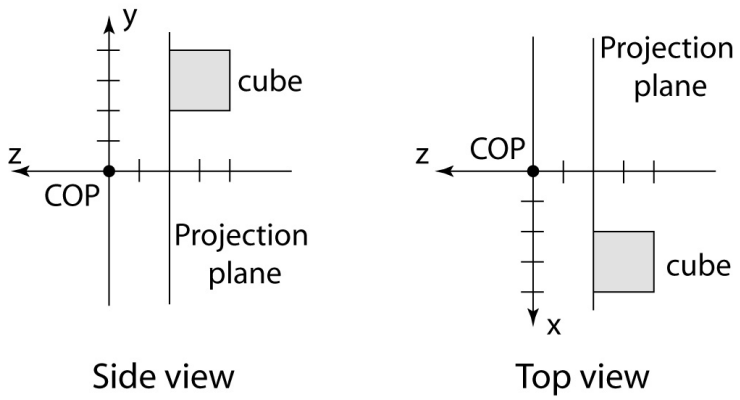
Due: Wednesday, May 16th
at the beginning of class

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

Name:_____

Problem 1: Projections (18 points)

Shown below are two parallel projections of a cube, a projection plane, and a center of projection (COP). The "side view" is the parallel projection looking down the -x direction and the "top view" is the parallel projection looking down the -y direction. As can be determined from the drawings, the corner of the cube closest to the origin is at (2,2,-2).



For the questions below, you will start by considering the perspective projection that arises from the geometry illustrated in the figure.

As part of this problem, you will be asked about the vanishing points of the cube. In particular, each line segment of the cube lies on a line that has a vanishing point. We will refer to this as one of the vanishing points of the cube. However, some vanishing points may be at infinity; we will not consider or count these vanishing points at infinity in this problem.

(a) (4 points) Compute the (x,y) coordinates of the eight corners of the cube after projection.

(b) (4 points) Sketch the cube as it would be seen in the perspective projection given the center of projection and projection plane. Place marks at unit spacing on the axes (as has been done in the figure) so that the locations of the cube corners can be read from the drawing. Draw hidden lines as dashed, and identify any vanishing points (labeled "VP" on your drawing).

(c) (2 points) What happens to the vanishing point(s) as we translate the cube in the –y direction? Explain.

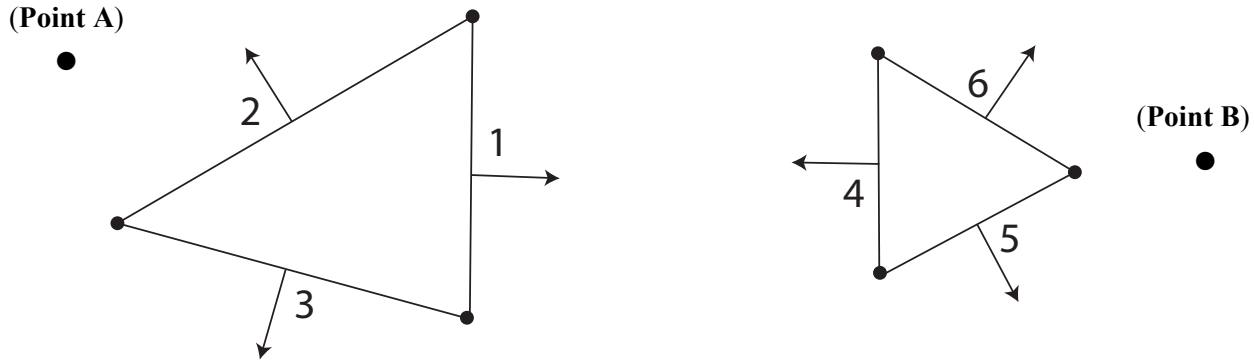
(d) (4 points) If you can freely rotate and translate the cube, what are the minimum and maximum number of vanishing points you will see in its projection? Explain.

(e) (4 points) In general, if we rotate and translate the projection plane without moving the center of projection, how do the occlusion relationships change? That is, do surfaces that were occluded in one image become unoccluded in another? Explain.

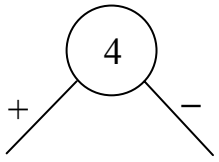
Problem 2. BSP trees (19 points)

Recall that Binary Space Partitioning (BSP) trees break the world up into a tree of positive and negative half-spaces. As spatial partitioning data structures they are very versatile and can be applied to aid in hidden surface removal, collision detection, and even robot motion planning.

Below is a world in two dimensions described by a set of numbered line segments. Note that each segment normal (shown as arrows) points into the positive (+) half-space of its segment.



(a) (4 points) Draw a diagram of a BSP tree that could be generated from this scene using segment #4 as the root (as illustrated below). Note: there are multiple valid answers.



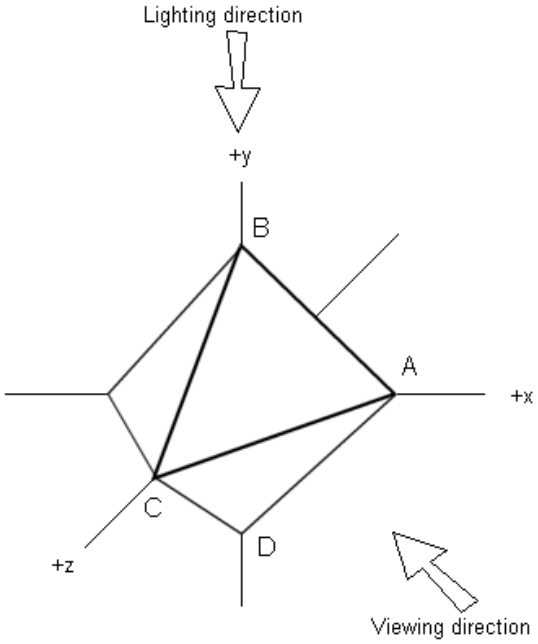
(b) (3 points) Given the viewpoint marked **Point A** in the scene, traverse your BSP tree to list the polygons in the order they would be rendered for hidden surface removal so that no Z-buffer is required (i.e., using the "Painter's algorithm"). For you answer, you only need to provide the list of primitives in the order in which they will be drawn.

(c) (5 points) An alternative approach to hidden surface removal is Z-buffering. Suppose we are performing time-consuming shading at each pixel while Z-buffering. As noted in class, performing shading calculations *after* doing the Z comparison will save some work, but you can still end up overwriting pixel values many times. How can Z-buffering and BSP trees be combined to minimize the number of shading calculations needed to render a scene? Assuming that all objects (primitives) in an arbitrary scene are opaque and non-overlapping, would we ever shade any pixel more than once? Justify your answer.

(d) (3 points) Using the Z-buffer + BSP tree combination from (c) and given your BSP tree from (a), in what order would you draw the line segments for the viewpoint marked **Point B** above?

(e) (4 points) Suppose we render with "backface culling," so that we don't draw any polygons that are facing away from the viewer. How would you modify the BSP-based traversal and drawing algorithm? Explain and then write out the ordered list of primitives that would be drawn for part (b) of this problem and then for part (d), if we used backface culling.

Problem 3. Interpolated shading (23 points)



The faceted polyhedron shown in the figure at right is an octahedron and consists of two pyramids connected at the base comprised of a total of 8 equilateral triangular faces with vertices at (1,0,0), (0,1,0), (0,0,1), (-1,0,0), (0,-1,0), and (0,0,-1). The viewer is at infinity (i.e., views the scene under parallel projection) looking in the (-1,0,-1) direction, and the scene is lit by directional light shining down from above parallel to the y-axis with intensity $L = (1,1,1)$. The octahedron's materials have both diffuse and specular components, but no ambient or emissive components. The Phong shading equation thus reduces to:

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

For this problem, $k_d = k_s = (0.5, 0.5, 0.5)$ and $n_s=40$.

(a) (2 points) In order to draw the faces as flat-shaded triangles, we must shade them using only their face normals. In OpenGL, this could be accomplished by specifying the vertex normals as equal to the face normals. (The same vertex would get specified multiple times, once per triangle with the same coordinates but different normal each time.) What are the unit normals for triangles ABC and ACD?

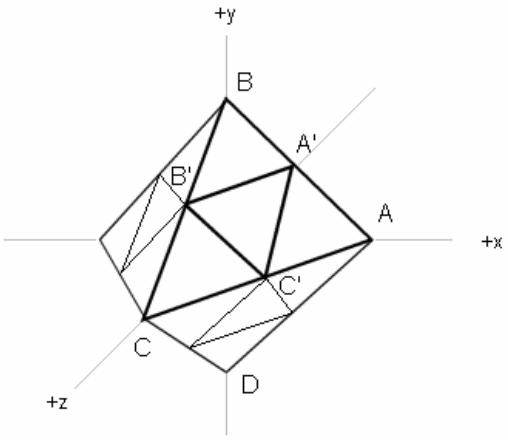
(b) (3 points) Assume that this object is really just a crude approximation of a sphere (e.g., perhaps you are using the octahedron to represent the sphere because your graphics card is slow). If you want to shade the octahedron so that it approximates the shading of a sphere, what would you specify as the unit normal at each vertex of triangle ABC?

(c) (6 points) Given the normals in (b), compute the rendered colors of vertices A, B, and C. Show your work.

(d) (3 points) Describe the appearance of triangle ABC as seen by the viewer using Gouraud interpolation.

(e) (4 points) Assume that the normals have been specified as in (c). Now switch from Gouraud-interpolated shading to Phong-interpolated shading. How will the appearance of triangle ABC change?

(f) (3 points) Remember that this object is being used to simulate a sphere. One simple improvement to the geometry of the model is to subdivide each triangular face into four new equilateral triangles and then specify three new additional vertices (sometimes called 4-to-1 triangular subdivision). If you subdivided triangle ABC this way, as shown in the figure to the right, what would be the best choices for the new coordinates and unit normals of the three added vertices A', B', and C' in order to more closely approximate a sphere?



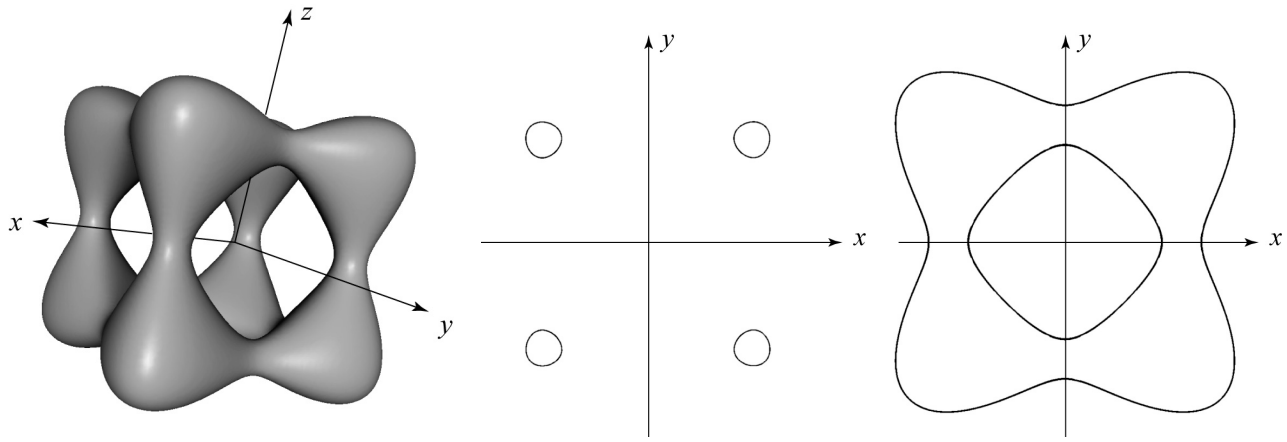
(g) (2 points) If you continued this subdivision process – repeatedly performing 4-to-1 subdivision, repositioning the inserted vertices, and computing their ideal normals – would the Gouraud-interpolated and Phong-interpolated renderings converge toward the same answer, i.e., the appearance of a ray traced sphere?

Problem 4. 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 so-called "tangle cube," described by the equation:

$$x^4+y^4+z^4-5x^2-5y^2-5z^2+12=0$$

On the left is a picture of a tangle cube, in the middle is the slice through the *x*-*y* plane (at *z* = 0), and on the right is a slice parallel to the *x*-*y* plane taken toward the bottom of the tangle cube (plane at z ≈ -1.5):



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:

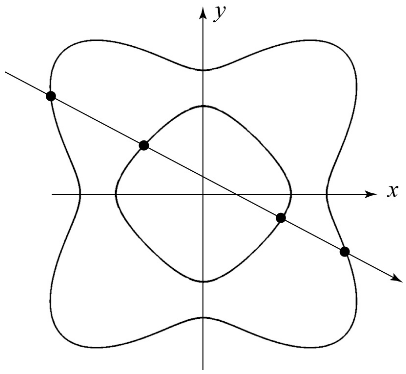
- You will find as many roots as the order (largest exponent) of the polynomial.
- You may find a mixture of real and complex roots. When we say complex here, we mean a number that has a non-zero imaginary component.
- All complex roots occur in complex conjugate pairs. If $A + iB$ is a root, then so 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 $\mathbf{d}=\begin{pmatrix}1 & 1 & 0\end{pmatrix}$. Typically, we normalize **d,** but for simplicity (and without loss of generality) you can work with the un-normalized **d** as given here.

- Solve for all values of *t* where the ray intersects the tangle cube (**including** any negative values of *t*). Show your work.
- 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 4 (cont'd)

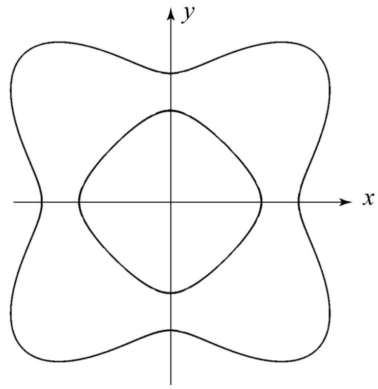
(b) (15 points) What are all the possible combinations of roots, not counting the one in part (a)? For each combination, describe the 4 roots as in part (a), draw a ray in the x-y plane that gives rise to that combination, and place a dot at each intersection point. You may not need all of the given diagrams. The first one has already been filled in. (Note: not all conceivable combinations can be achieved on this particular tangle cube. 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 solutions. You do not need to justify your answers.***



# of distinct real roots: **4**

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

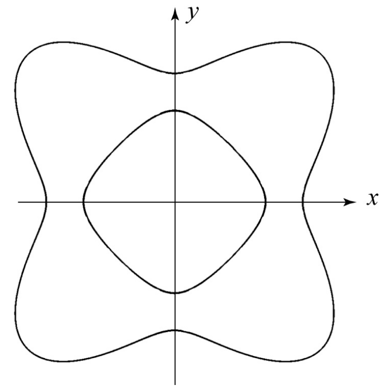
# of complex roots: **0**



# of distinct real roots:

# of roots w/ multiplicity > 1:

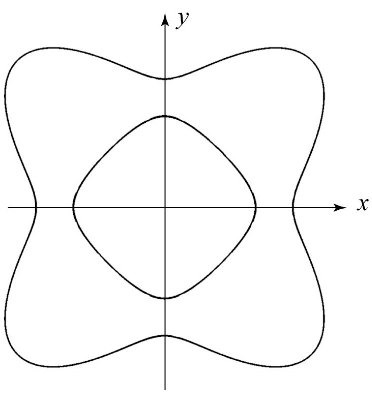
# of complex roots:



# of distinct real roots:

# of roots w/ multiplicity > 1:

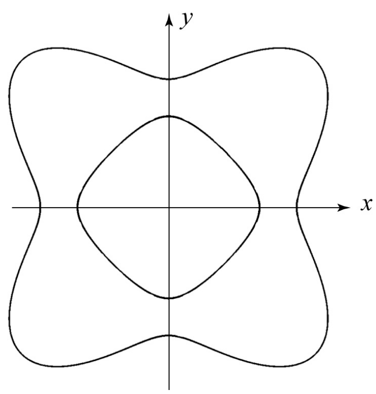
# of complex roots:



# of distinct real roots:

# of roots w/ multiplicity > 1:

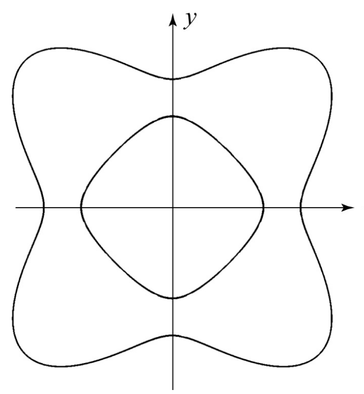
# of complex roots:



# of distinct real roots:

# of roots w/ multiplicity > 1:

# of complex roots:



# of distinct real roots:

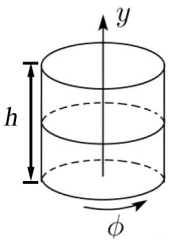
# of roots w/ multiplicity > 1:

# of complex roots:

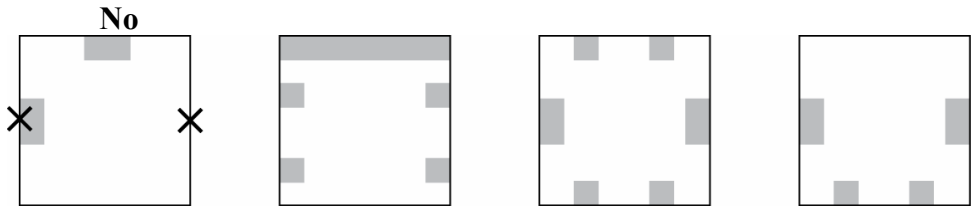
Problem 5. Texture mapping (15 points)

When a texture map is applied to a surface, points that are distinct in the rectangular texture map may be mapped to the same place on the object. For example, when a texture map is applied to a cylinder, the left and right edges of the texture map are mapped to the same place. We call a mapping "valid" if it does not map two points of different colors to the same point on the object. For each of the 16 cases below, indicate whether the mapping is valid (write "Yes" or "No"). If it is not valid, mark with an **X** two points on the texture map that map to the same point on the object, but have different colors. Use the texture mapping formulas specified below for each primitive, where the u, v parameters range from 0 to 1. The first of the 16 cases below is done for you.

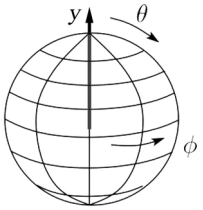
Uncapped cylinder (top and bottom are open)



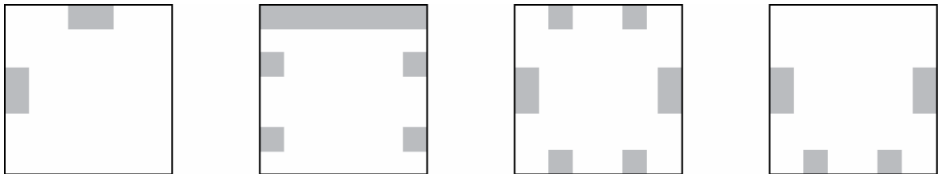
$$\begin{cases} u = \phi/2\pi \\ v = y/h \end{cases}$$



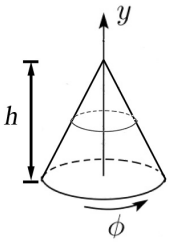
Sphere



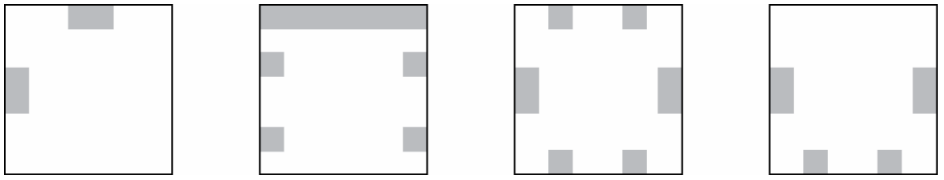
$$\begin{cases} u = \phi/2\pi \\ v = \theta/\pi \end{cases}$$



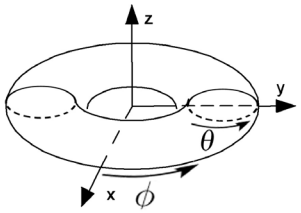
Uncapped cone (bottom is open)



$$\begin{cases} u = \phi/2\pi \\ v = y/h \end{cases}$$



Torus



$$\begin{cases} u = \phi/2\pi \\ v = \theta/2\pi \end{cases}$$