Homework #2

Shading, Ray Tracing, Texture Mapping,

Projections, Hidden Surfaces, Anti-Aliasing + Acceleration

Prepared by: Peter Lincoln, Jonathan Su, and Steve Seitz

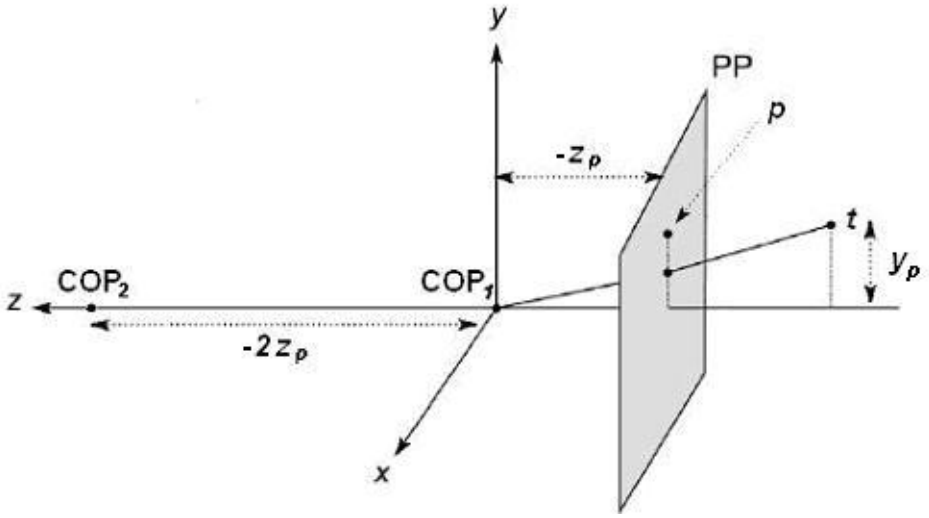
Assigned: Thursday, May 4th
Due: Thursday, May 18th @ 10:30 AM in class

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. Projections

Imagine there is a pin-hole 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), as shown in the figure below. Let there be two points in the scene, $p = [0\ y_p\ z_p\ 1]^T$ and $t = [0\ y_p\ 2z_p\ 1]^T$.



The projection matrix for this camera, as derived in class, 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 *t* to the point $[0\ .5y_p\ 1]^T$.

a) 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) If your matrix in part a) is correct, point *p* should project to the same image point as before. Calculate the projection of point *t*. How does the projection of point *t* change when the camera moves from COP_1 to COP_2? What is this effect called?

c) Now consider moving the COP infinitely far back along the positive *z* axis. Derive the new projection matrix, for this limit case. **Show your work.** What is this sort of projection called?

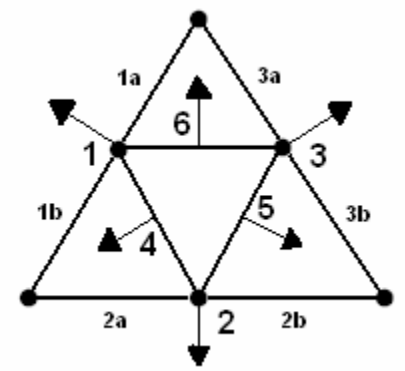
d) Now imagine that the points, the camera (with center of projection COP_2), and *PP* are rotated 90 degrees about the *x*-axis, so that the camera is now looking in the positive *y* direction. Give the new coordinates of *p* and *t*. Give the new projection matrix, except instead of dropping the *z* component, now drop the *y* component. Figure out *p"* and *t"*, the projections of the new *p* and *t* onto the new PP. How has the image changed, compared to part a)?

Problem 2. BSP Trees

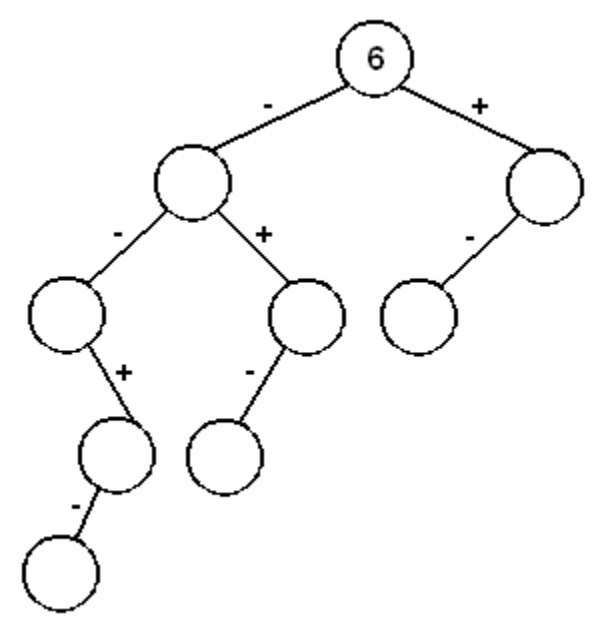
Recall that Binary Space Partitioning (BSP) trees break the world up into a tree of positive and negative half-spaces. Below is a 2D scene containing 6 line segments. Note that each segment normal (shown as arrows) points into the positive (+) half-space of its segment.

(**Point A**)

●



a) Fill in the BSP tree below using this scene and treating segment #6 as the root. We have labeled the "a" and "b" half of segments 1, 2, and 3 in case you find you need to split them. Note that many binary tree configurations are valid, but find one that fits the tree below.



b) Given the viewpoint marked **Point A** in the scene, traverse your BSP tree to list the polygons in the order they would be drawn. For you answer, you only need to provide the list of primitives in the order in which they will be drawn (the first segment in your list is drawn first).

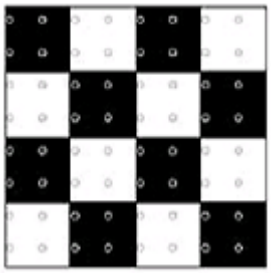
c) In class, we talked about doing a "back to front" traversal of a BSP tree. But it is sometimes preferable to do a "front to back" traversal of the tree, in which we draw polygons closer to the viewer before we draw the polygons farther away. (See part (d) for one reason why this is useful) How should the tree traversal order be changed in order to do a front to back traversal?

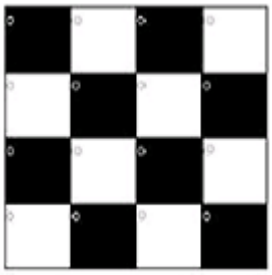
d) When we traverse a BSP tree in back to front order, we may draw over the same pixel location many times; this is inefficient as it requires many "useless" shading computations. Assume we instead traverse the tree in front to back order. As we draw each polygon, we wish to determine whether or not each point will be visible in the final scene (and thus whether we need to compute shading information for that point). This is not a Z-buffer algorithm, and we're not going to keep Z-values around. What simple information (one bit per pixel) about the frame buffer do we need to maintain in order to know if each point in the next polygon we draw will be visible?

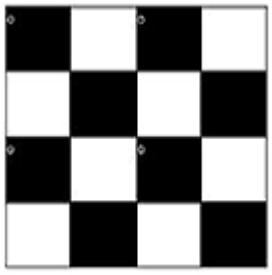
Problem 3. Anti-aliasing

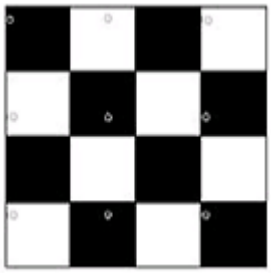
In lecture, we talked about how aliasing can cause "crufty" visual artifacts to appear in images. One reason aliasing occurs is because the function is sampled below the Nyquist rate.

a) For each of the checkboard images below, assume we sample at each point marked by a circle. Describe what image each sampling will produce (i.e. what pattern will be produced, and what colors the pattern will consist of) and whether it will look correct (whether or not there will be aliasing).

i. 

ii. 

iii. 

iv. 

b) Suppose your output resolution is limited to one pixel per 2x2 block (as in case iii in part a)). Suppose you use super-sampling to sample once in each of the four boxes that make up the 2x2 block, and then average the four samples to obtain your output sample value. Describe the resulting image, and explain why this result might be considered better than the non-super-sampled result from case iii in part a)).

c) Suppose that instead of using super-sampling, you used jittered sampling. In particular, you choose one sample per 2x2 block, randomly chosen from within the 2x2 block (assume any position is just as likely). Describe the resulting image and explain why this result might be considered better than the non-super-sampled result from case iii in part a)).

Problem 4. Shading

The Phong shading model for a scene illuminated by global ambient light and a single light is given by the following equation:

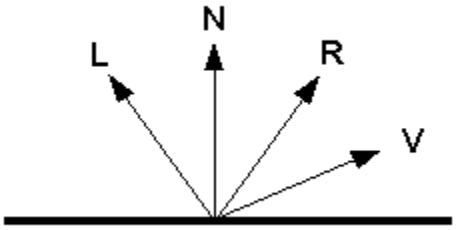
$$I = k_e + k_a L_a + \frac{1}{a + bd + cd^2}\left(k_d L_d (\mathbf{N}\cdot\mathbf{L})_+ + k_s L_s (\mathbf{V}\cdot\mathbf{R})_+^{n_s}\right)$$

OpenGL actually uses a modified version of this shading model, in which the specular component is changed as follows:

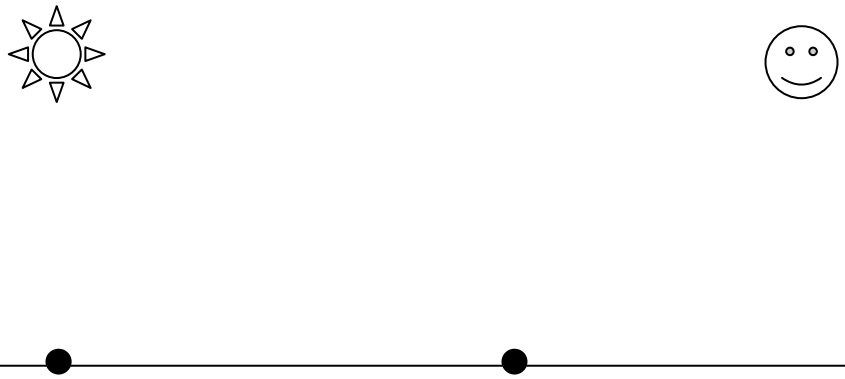
$$I = k_e + k_a L_a + \frac{1}{a + bd + cd^2}\left(k_d L_d (\mathbf{N}\cdot\mathbf{L})_+ + k_s L_s (\mathbf{N}\cdot\mathbf{H})_+^{n_s'}\right)$$

H is defined as the halfway vector, where $\mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{|\mathbf{L} + \mathbf{V}|}$.

a) Considering the surface below, draw the normalized halfway vector, **H**. Assume that all of the drawn vectors are already normalized to be unit vectors

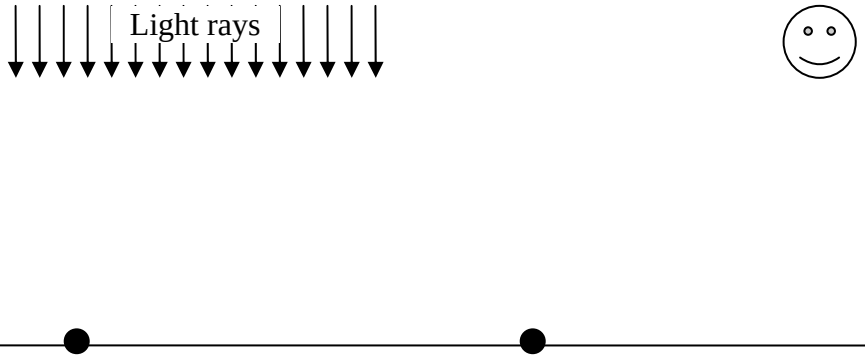


b) Suppose the light source (sun) and camera (face) are as shown below. Draw and label the vectors **L**, **N**, **R**, **V**, and **H** at each of the marked points.



Which vectors, if any, are constant for all three of the marked points?

c) Suppose the light source was translated up by a great distance. Now draw and label the vectors **L**, **N**, **R**, **V**, and **H** for the new situation.



Under this new situation, which vectors, if any, are constant for the three marked points?

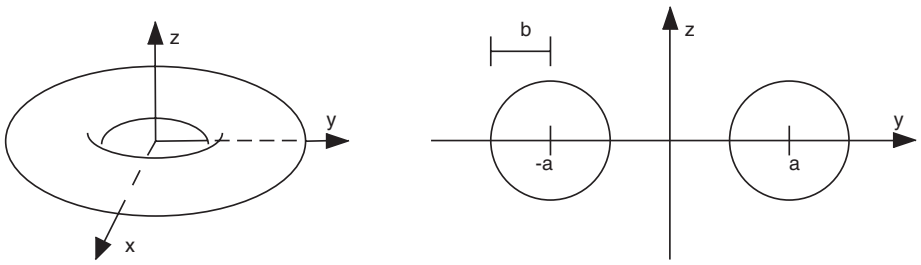
d) Consider the cases in which, relative to a surface, the light is near or far, and the viewer is near or far. Consider also that the surface may be flat or curved. When and how could the modified shading calculations be simplified? A calculation can be considered simplified if one of the computations in the shading model is held constant (i.e., $\mathbf{N} \cdot \mathbf{L}$, $\mathbf{N} \cdot \mathbf{H}$, or $\mathbf{V} \cdot \mathbf{R}$).

For configuration of light, viewer, and surface, write a "*yes*" under the column ($\mathbf{N} \cdot \mathbf{L}$, $\mathbf{N} \cdot \mathbf{H}$, and/or $\mathbf{V} \cdot \mathbf{R}$) if that quantity is constant over the surface.

Light	**Viewer**	**Surface**	$\mathbf{N} \cdot \mathbf{L}$	$\mathbf{N} \cdot \mathbf{H}$	$\mathbf{V} \cdot \mathbf{R}$
Near	Near	Flat			
Near	Near	Curved			
Near	Far	Flat			
Near	Far	Curved			
Far	Near	Flat			
Far	Near	Curved			
Far	Far	Flat			
Far	Far	Curved			

Problem 5. Ray Tracing

Suppose we want to ray trace a torus, as illustrated below:



The implicit surface formula for the torus illustrated above is the following equation:
$f(x,y,z)=(x^2+y^2+z^2-(a^2+b^2))^2-4a^2\cdot(b^2-z^2)=0$

For the remainder of the problem, assume the torus has major radius $a=2$ and minor radius $b=1$.

a) Consider a ray as the equation $r(t)=\vec{P}+t\vec{d}$. For each of the listed values for $\vec{P}$ and $\vec{d}$, determine all of the real values of *t* for which the ray intersects the torus, if any. (hint: you may find it convenient to use the variable substitution $u = t^2$, but your solutions must be in terms of *t*)

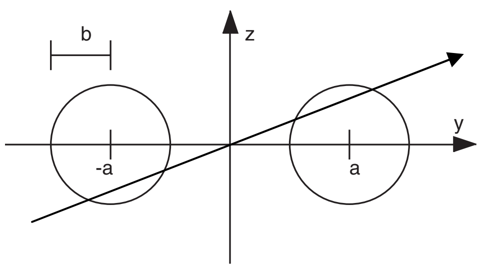
i. $\vec{P}=(0 \quad 0 \quad 0) \quad \vec{d}=(0 \quad 0 \quad 1)$:

ii. $\vec{P}=(0 \quad 0 \quad 0) \quad \vec{d}=(0 \quad 1 \quad 0.5)$:

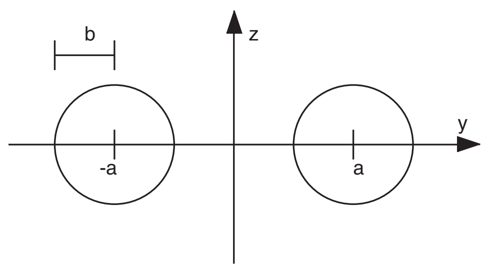
iii. $\vec{P}=(-3 \quad 1 \quad 0) \quad \vec{d}=(0 \quad 1 \quad 0)$

b) A ray may intersect the surface zero or more times. If it does intersect, which of the value(s) is/are relevant for shading calculations in ray tracing? Circle your relevant values from a).

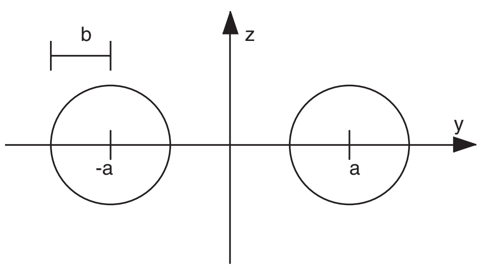
c) Solving for intersections involves evaluating the roots of the polynomial *f*. What are all the possible unique combinations of roots? Count repeated roots only once. For each possible combination, illustrate one ray that produces that combination. The first one has already been done for you. There are exactly six cases.



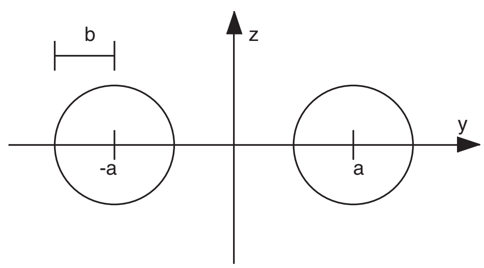
# of unique real roots: 4
# of unique non-real roots: 0



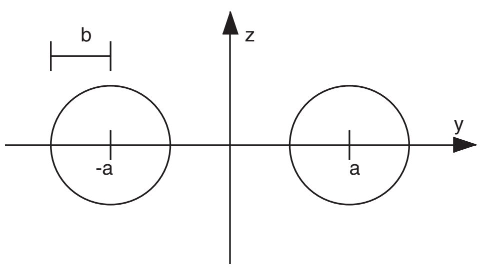
# of unique real roots: ___
# of unique non-real roots: ___



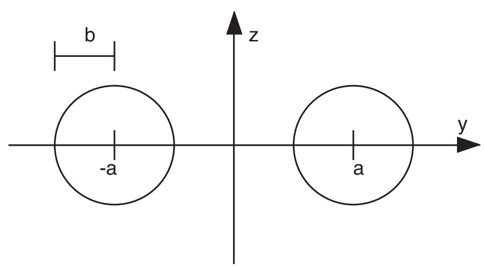
# of unique real roots: ___
# of unique non-real roots: ___



# of unique real roots: ___
# of unique non-real roots: ___



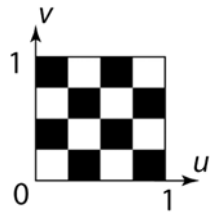
# of unique real roots: ___
# of unique non-real roots: ___



# of unique real roots: ___
# of unique non-real roots: ___

Problem 6. Texture Mapping

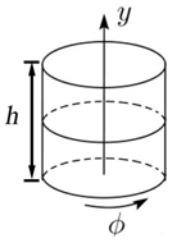
When a texture map is applied to a surface, points that are distinct in the rectangular texture map may be mapped to the same place on the object. For example, when a texture map is applied to a cylinder, the left and right edges of the texture map are mapped to the same place. We call a mapping "valid" if it does not map two points of different colors to the same point on the object. For each of the 16 cases below, state whether the mapping is valid (write yes or no). If it is not valid, mark two points on the texture map with an X that are different colors, but map to the same point on the object. Use the texture mapping formulas specified below for each primitive. Below is a diagram defining the *u* and *v* axes of the texture map:



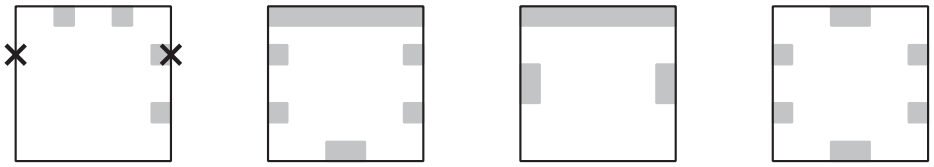
The first of the 16 cases below is done for you.

Uncapped cylinder (top and bottom are open):

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

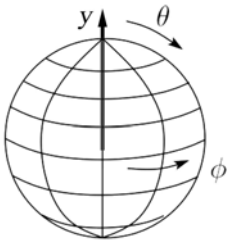


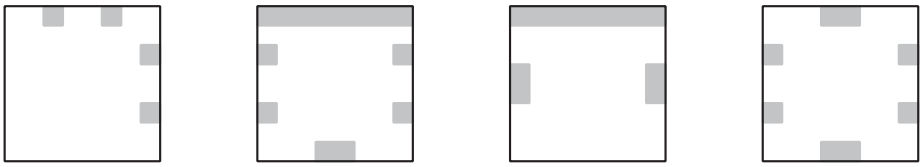
No



Sphere:

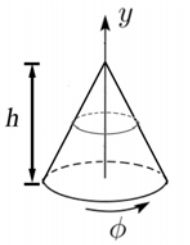
$$\begin{cases} u = \phi/2\pi \\ v = \theta/\pi \end{cases}$$

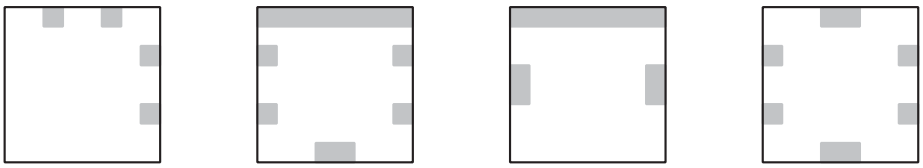




Uncapped cone (bottom is open):

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





Torus:

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