Computer Graphics	Prof. Brian Curless
CSE 457	Spring 2004

Homework #2

Hidden Surfaces, Shading, Ray Tracing,

Texture Mapping, Parametric Curves

Prepared by: Matthew Burkhart, Andrew Stoneman, and Brian Curless

Assigned: Saturday, May 8th
Due: Friday, May 21st
at the beginning of class

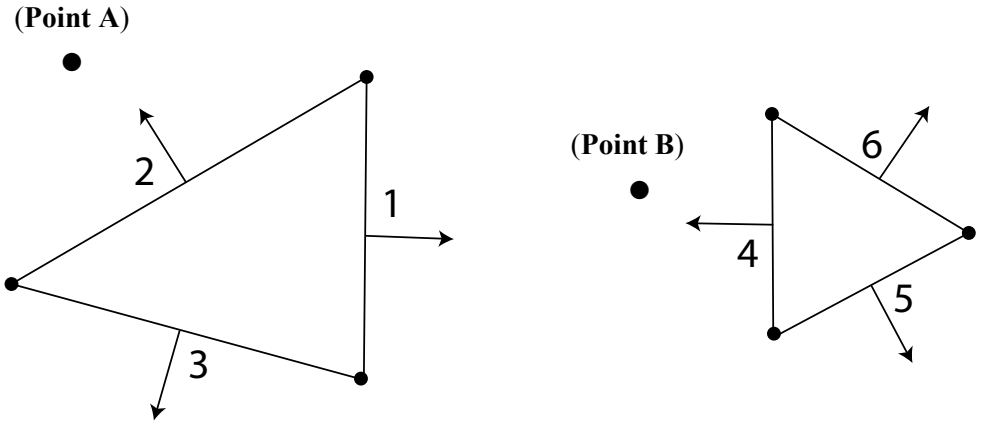
Directions: Please provide short written answers to the questions in the space provided. If you require extra space, you may staple additional pages to the back of your assignment. Feel free to discuss the problems with classmates, but please ***answer the questions on your own*.**

Name:_____

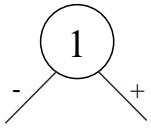
Problem 1. BSP Trees (15 points)

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, constructive solid geometry, 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 #1 as the root. Note that while many binary tree configurations are valid, you should try to make your tree reasonably balanced.



Problem 1. BSP Trees (cont'd)

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.

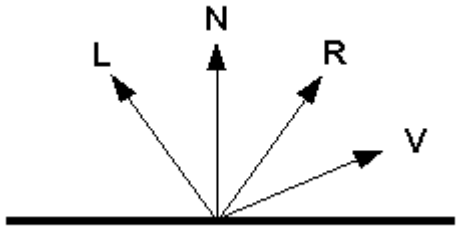
c. (5 points) An alternative approach to hidden surface removal is Z-buffering. Suppose we are performing time-consuming shading at each pixel while Z-buffering. As noted in class, performing shading calculations *after* doing the Z comparison will save some work, but you can still end up overwriting 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 in an arbitrary scene are opaque, can we still end up computing the shade at 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** in the diagram at the beginning of this problem?

Problem 2. Halfway vector specular shading (10 points)

Blinn and Newell have suggested that if **V** and **L** are each assumed to be constants the computation of $\mathbf{V} \cdot \mathbf{R}$ in the Phong shading model can be simplified by associating with each light source a fictitious light source that will generate specular reflections. This second light source is located in a direction **H** halfway between **V** and **L**. The specular component is then computed from $(\mathbf{N} \cdot \mathbf{H})^{n_s}$, instead of from $(\mathbf{V} \cdot \mathbf{R})^{n_s}$.

a. (2 points) On the diagram below, assume that **V** and **L** are the new constant viewing direction and lighting direction vectors. Draw the new direction **H** on the diagram.

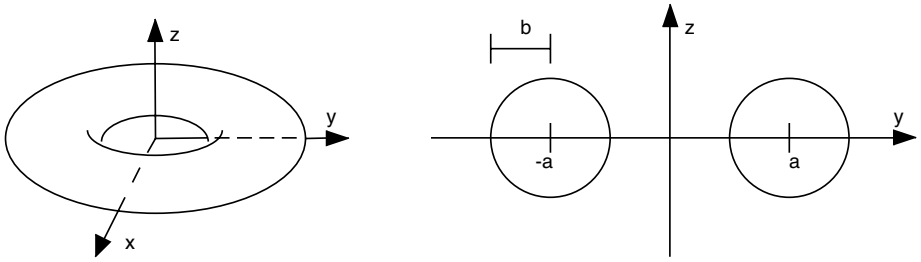


b. (4 points) Under what circumstances or by making what approximations might **L** and **V** be assumed constant (or, at least, roughly so) for every point in the scene as seen through every pixel on the image plane?

c. (4 points) Let's make the constant L, V assumption and use the halfway vector for shading. What is an advantage of this approach? For general lighting and viewing conditions, what is a disadvantage of this approach?

Problem 3. Ray tracing of implicit surfaces (23 points)

There are many ways to represent a surface. One such 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 torus. Tori are described by the equation $(x^2+y^2+z^2-(a^2+b^2))^2-4a^2\cdot(b^2-z^2)=0$, with a and b being the major and minor radii, respectively, as pictured in the y-z slice below.



For this problem, we will use the torus with major radius $a = 2$ and minor radius $b = 1$. This gives us the equation $(x^2+y^2+z^2-5)^2-16\cdot(1-z^2)=0$.

a. (5 points) Consider the ray $P+t\mathbf{d}$, where $P=\begin{pmatrix}0 & 0 & 0\end{pmatrix}$ and $\mathbf{d}=\begin{pmatrix}0 & 1 & 0\end{pmatrix}$. Solve for *t* such that the ray intersects the torus. Which value of *t* represents the intersection we care about for ray tracing?

Problem 3 (cont'd)

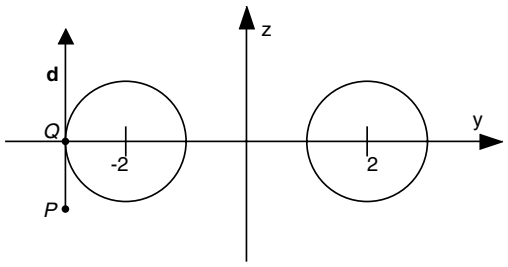
b. (3 points) Any ray substituted into the equation for the torus will result in a 4^{th} degree polynomial in *t*. This means that the resulting polynomial will have 4 roots, although some of them may be complex, and some of them may be repeated. Answer the following questions about your roots from part a.

i) How many distinct, real roots were there?

ii) Did any of these roots have multiplicity greater than one? If so, how many, and what multiplicity?

iii) How many complex roots does this leave?

c. (3 points) Consider the following ray passing through *P* in the direction **d** in the x-z plane and intersecting the torus at *Q*:



Answer the following questions about the 4 roots that will result when this ray is substituted into the torus equation. (Note: You do not have to give values for the roots, or show any math to derive your answers. Just reason from what you learned about roots and intersections when we studied ray-sphere intersections in class.)

i) How many distinct, real roots are there?

ii) Do any of these roots have multiplicity greater than one? If so, how many, and what multiplicity?

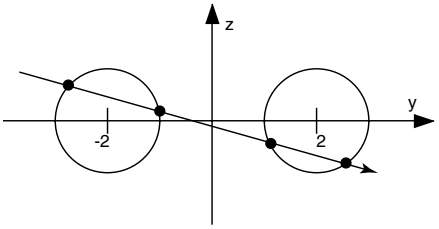
iii) How many complex roots does this leave?

Problem 3 (cont'd)

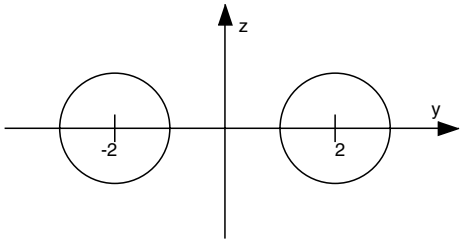
d. (12 points) What are all the possible combinations of roots, not counting the one in part (**c**)? For each combination, describe the 4 roots as in part (**c**), draw a ray in the x-z plane that gives rise to that combination, and place a dot at each intersection point. You may not need all of the given diagrams. The first one has already been filled in. (Hint: remember that complex roots always occur in pairs; you cannot have an odd number of them. Also, not all conceivable combinations can be achieved on this particular torus. For example, there is no ray that will give a root with multiplicity 4.)

Roots:

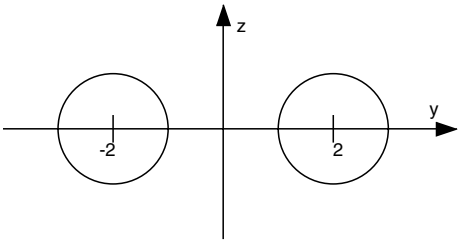
4 distinct real roots
no real roots with multiplicity > 1
no complex roots



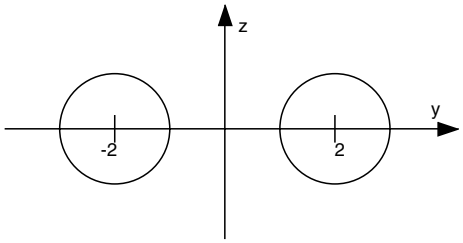
Roots:



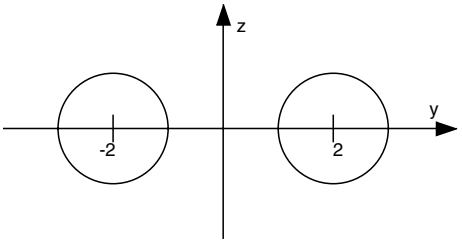
Roots:



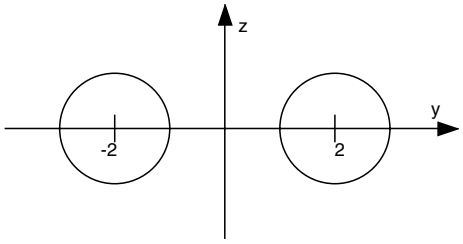
Roots:



Roots:



Roots:

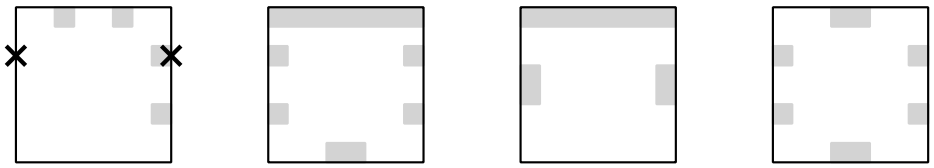


Problem 4. Texture mapping (15 points)

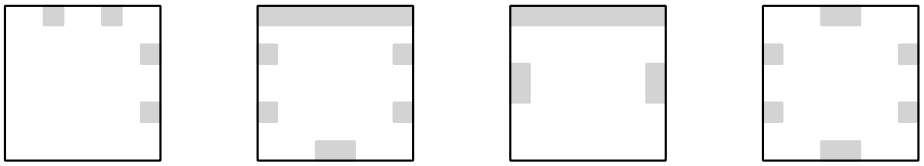
When a texture map is applied to a surface, points that are distinct in the rectangular texture map may be mapped to the same place on the object. For example, when a texture map is applied to a cylinder, the left and right edges of the texture map are mapped to the same place. We will call a mapping "allowed" if it does not map two points of different colors to the same point on the object to be texture mapped. For each of the combinations below, state whether the mapping is allowed (write yes or no). If it is not, mark two points on the texture with an X that are different colors, but map to the same point on the object. The first texture map for a cylinder primitive is done as an example.

Uncapped cylinder (top and bottom are open):

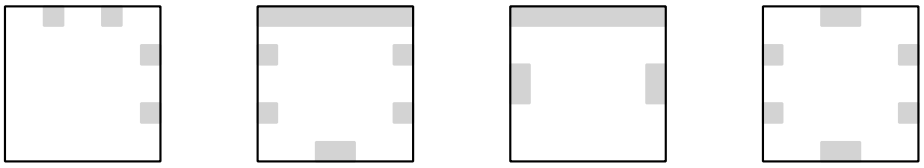
No



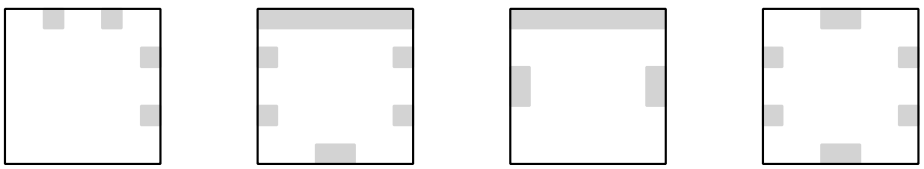
Sphere:



Uncapped cone (bottom is open):



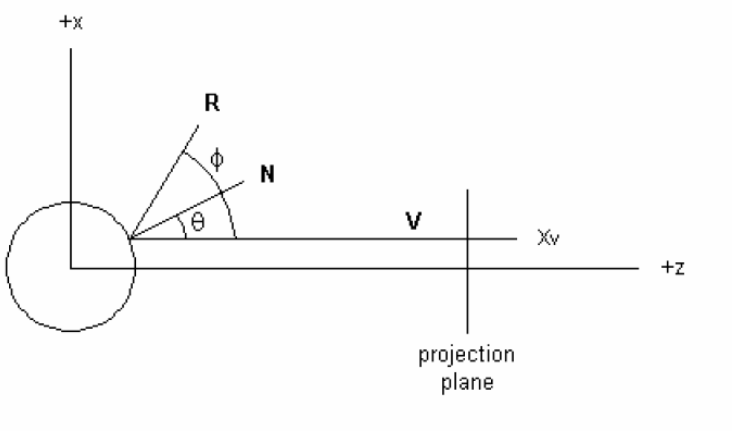
Torus:



Problem 5. Environment mapping (17 points)

One method of environment mapping (reflection mapping) involves using a "gazing ball" to capture an image of the surroundings. The idea is to place a chrome sphere in an actual environment, take a photograph of the sphere, and use the resulting image as an environment map. Let's examine this in two dimensions, using a "gazing circle" to capture the environment around a point.

Below is a diagram of the setup. In order to keep the intersection and angle calculations simple, we will assume that each view ray **V** that is cast through the projection plane to the gazing circle is parallel to the z-axis, meaning that the viewer is located at infinity on the z-axis. The circle is of radius 1, centered at the origin.



a. (5 points) If the x-coordinate of the view ray is x_v, what are the (x,z) coordinates of the point at which the ray intersects the circle? What is the unit normal vector at this point?

b. (3 points) What is the angle between the view ray **V** and the normal **N** as a function of x_v?

Problem 5 (cont'd)

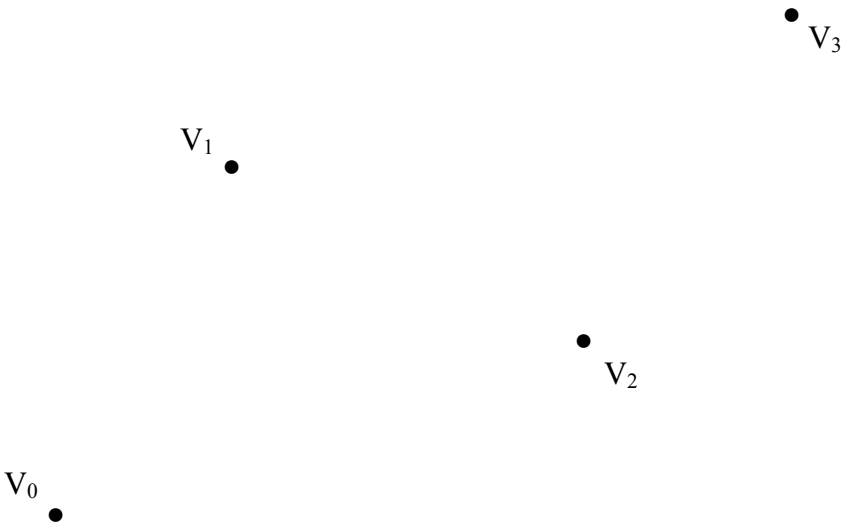
c. (5 points) Note that the angle φ between the view ray **V** and the reflection direction **R** is equal to 2θ, where θ is the angle between **V** and the normal **N**. Plot φ versus x_v. In what regions do small changes in the intersection point result in large changes in the reflection direction?

d. (4 points) We can now treat the photograph of the chrome circle as an environment map. If we were to ray-trace a new, shiny object and index into the environment map according to each reflection direction, would we expect to get the same rendering as if we had placed the object into the original environment we photographed? Why or why not?

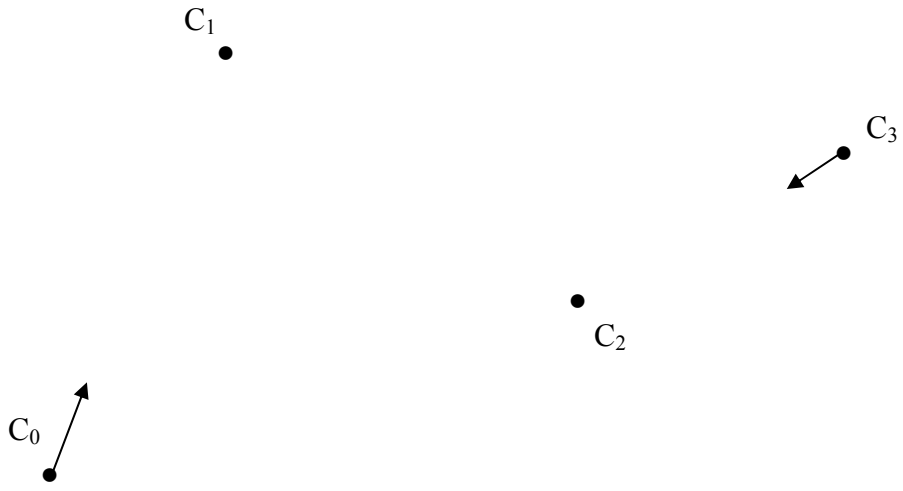
Problem 6. Parametric curves (20 points)

In this problem, we will explore the construction of parametric curves (a-c) and develop an understanding of parametric and geometric continuity for spline curves (d).

a. (4 points) Given the following Bezier control points, construct all of the de Casteljau lines and points needed to evaluate the curve at $u=1/3$. Mark this point on your diagram and then sketch the path the Bezier curve will take.

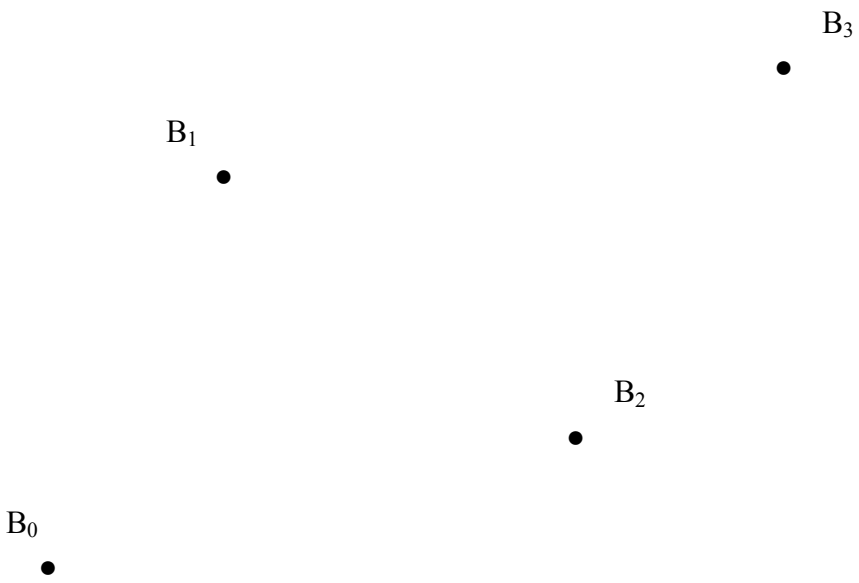


b. (4 points) Given the following Catmull-Rom control points, construct all of the lines and points needed to generate the Bezier control points for the Catmull-Rom curve. Use a tension value of $\tau=3/4$ and position the first control point after C_0 and the last control point before C_3 at the tips of the given arrows. You must mark each Bezier point with an X, but you do not need to label it (i.e., no need to give each Bezier point a name). Sketch the resulting spline curve.



Problem 6 (cont'd)

c. (4 points) Given the following de Boor points, construct all of the lines and points needed to generate the Bezier control points for the B-spline. Assume that the first and last points are repeated three times, so that the spline is endpoint interpolating. You must mark each Bezier point with an X, but you do not need to label it (i.e., no need to give each Bezier point a name). Sketch the resulting spline curve.

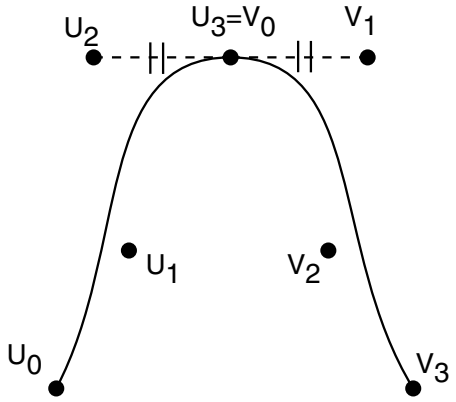


Problem 6 (cont'd)

d. (8 points) Consider two Bezier curves controlled by two sets of control points $U_0...U_3$, $V_0...V_3$. When splining these two curves together, we seek to attain a certain degree of continuity where one curve meets the next. In class, we discussed C^n continuity, which means that the *n*-th derivative with respect to the parameter value, *u*, is continuous. In particular, C^0 continuity implies that the curve does not jump from one position to another when moving between the Bezier curves, and C^1 continuity implies that the "velocity" (first derivative vector) is continuous, in addition to requiring C^0 continuity.

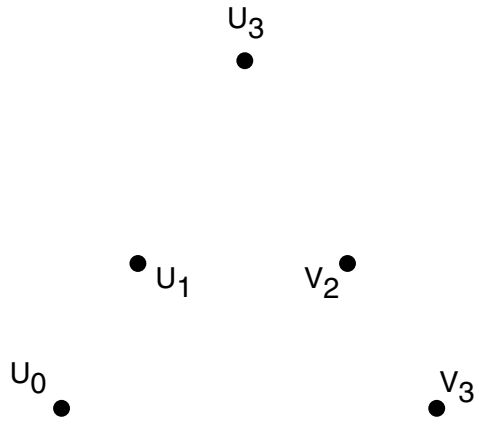
Another kind of continuity is "geometric" continuity, which requires the shape of the curve, when drawn, to meet certain smoothness criteria. It turns out that G^0 continuity is the same as C^0 continuity. However, G^1 continuity means that the tangent direction is continuous (no sharp corners), in addition to requiring G^0 continuity. The tangent to a curve, geometrically, is what you might intuitively expect. Analytically, since the velocity vector (parametric first derivative) is tangent to the curve, you can compute the tangent direction by normalizing the velocity vector (when the velocity is non-zero).

Below is an example of a curve that is both C^1 and G^1 continuous.



For the next two curves, the control points U_0, U_1, U_3, V_2, and V_3 are fixed as shown, and you must specify the locations of and label the remaining control points to establish the requested continuities. You must also sketch the resulting spline curve.

i. **C^0 & G^1** continuous spline



ii. **C^1 & G^0** continuous spline