Zoran Popović

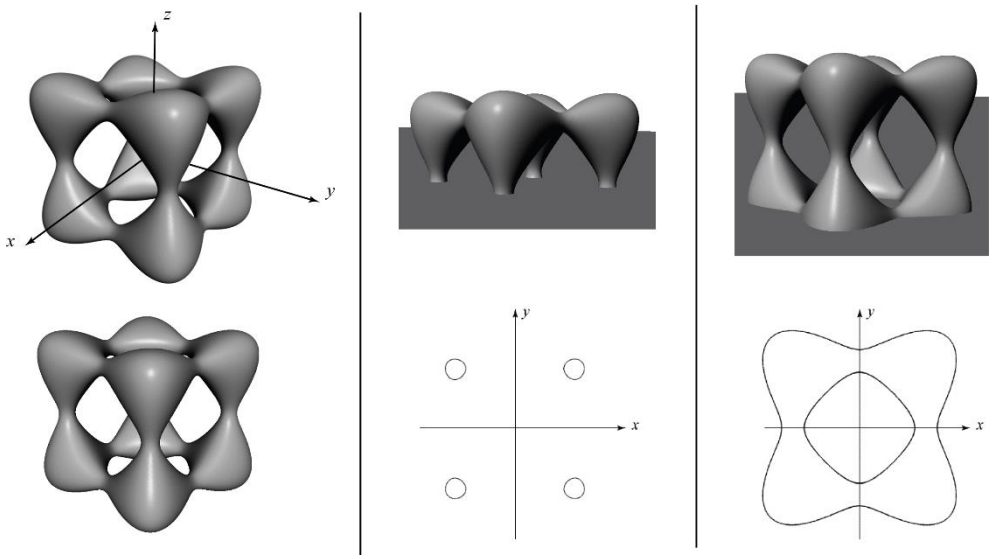
Homework 2

Problem 1 | Ray Tracing (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:

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

In the figure below, the left column shows two renderings of the tangle cube, the middle column illustrates taking a slice through the *x-y* plane (at $z = 0$), and the right column shows a slice parallel to the *x-y* plane taken toward the bottom of the tangle cube (plane at $z \approx -1.5$):



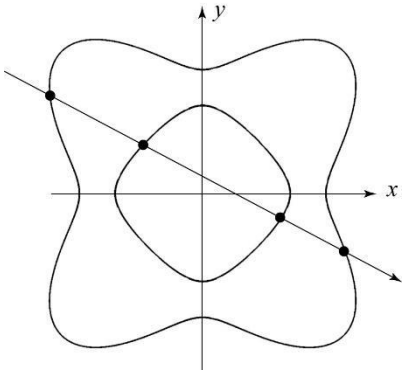
In the next problem steps, you will be asked to solve for and/or discuss ray intersections with this primitive. Performing the ray intersections will amount to solving for the roots of a polynomial, much as it did for sphere intersection. For your answers, you need to keep a few things in mind:

- 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 $P = (0 \quad 0 \quad 0)$ and $\mathbf{d} = (1 \quad 1 \quad 0)$. 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*). Your solution should be algebraic and lead to the **exact** *t* values. Show your work.
- In the process of solving for t, you should have computed the roots of a polynomial. How many distinct real roots did you find? How many of them have multiplicity > 1? How many complex roots did you find?
- Which value of *t* represents the intersection we care about for ray tracing?

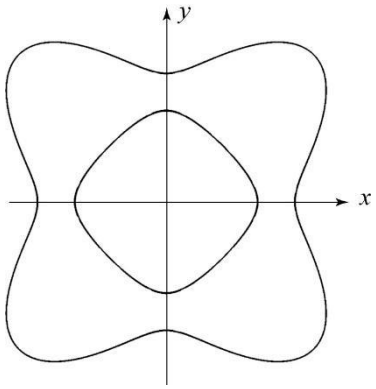
b) (15 points) What are all the possible combinations of roots when ray tracing this surface, not counting the one in part (a)? For each combination, describe the 4 roots as in part (a), draw a ray in the *x*-*y* plane that gives rise to that combination, and place a dot at each intersection point. Assume the origin of the ray is outside of the bounding box of the object. 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.) "# of distinct real roots" is how many roots there are with different values. So, for example, if you had roots 1,2,1,2, the number of distinct real roots is 2. "# of real roots w/ multiplicity > 1" would be 2, since these two real roots have multiplicity > 1. ***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**

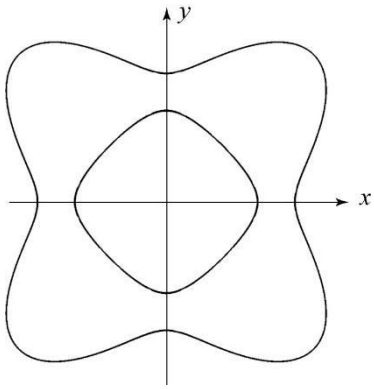
# of complex roots: **0**



# of distinct real roots:

# of real roots w/ multiplicity > 1:

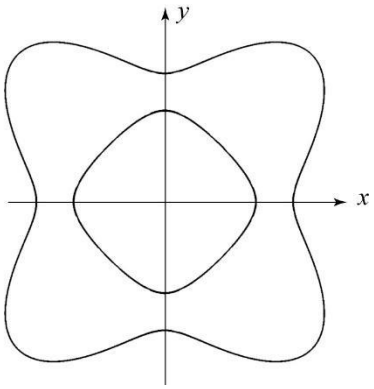
# of complex roots:



# of distinct real roots:

# of roots w/ multiplicity > 1:

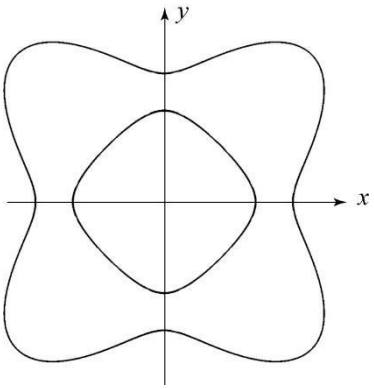
# of complex roots:



# of distinct real roots:

# of real roots w/ multiplicity > 1:

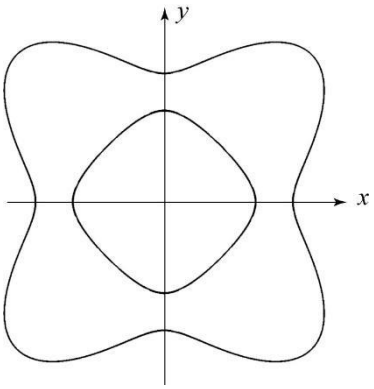
# of complex roots:



# of distinct real roots:

# of real roots w/ multiplicity > 1:

# of complex roots:



# of distinct real roots:

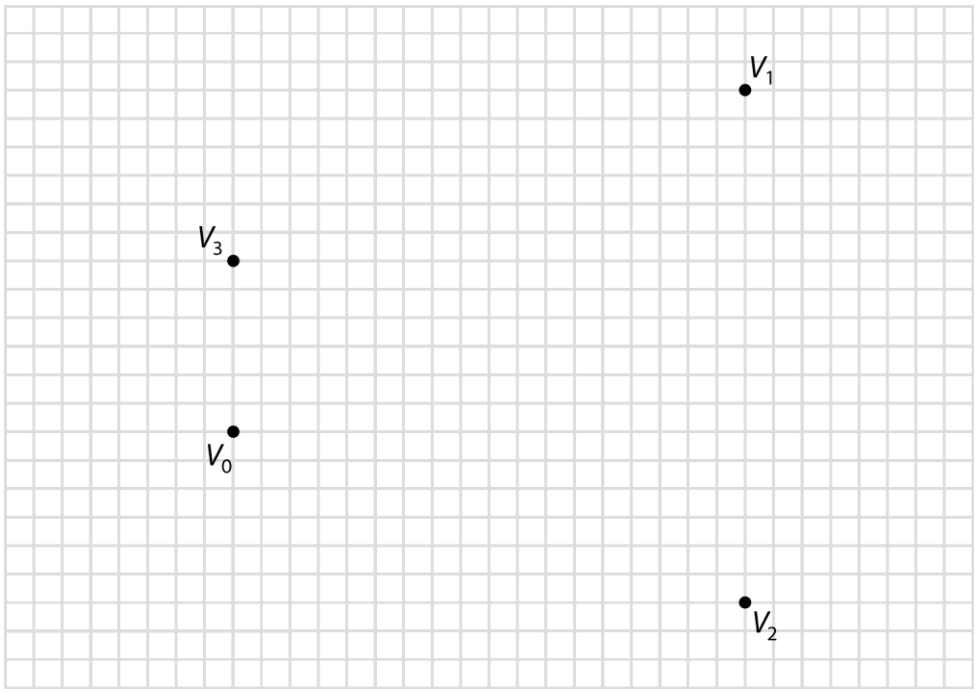
# of real roots w/ multiplicity > 1:

# of complex roots:

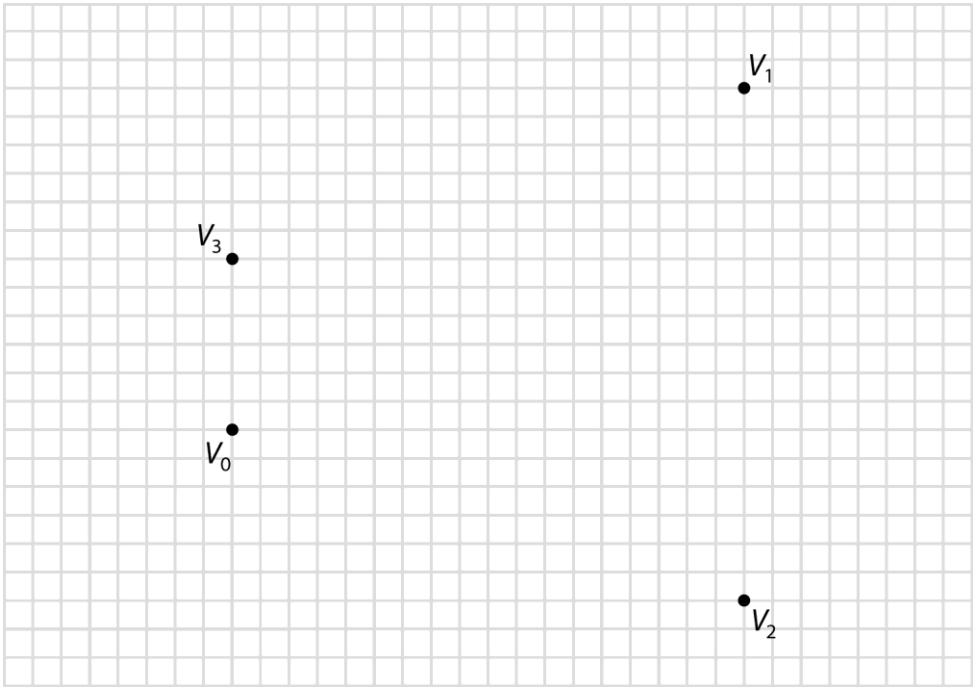
Problem 2 | Parametric Curves (30 points)

In this problem, we will explore the construction of parametric curves. ***Please write on the pages for this problem and include them with your homework solution.***

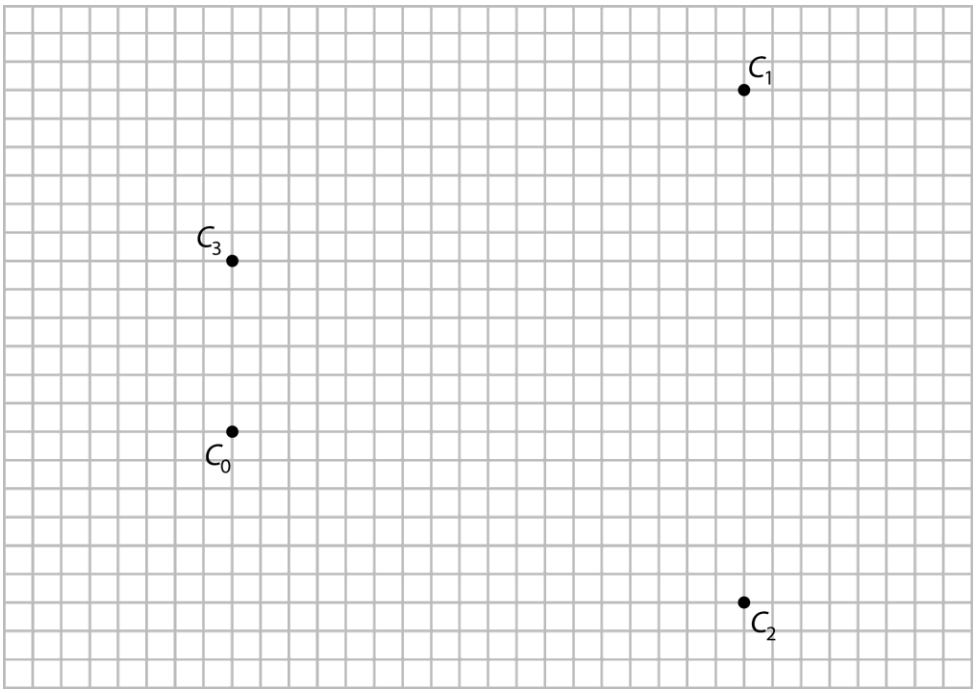
a) (4 points) Given the following Bezier control points, construct all of the de Casteljau lines and points needed to evaluate the curve at u=1/4. Label this point Q(1/4).



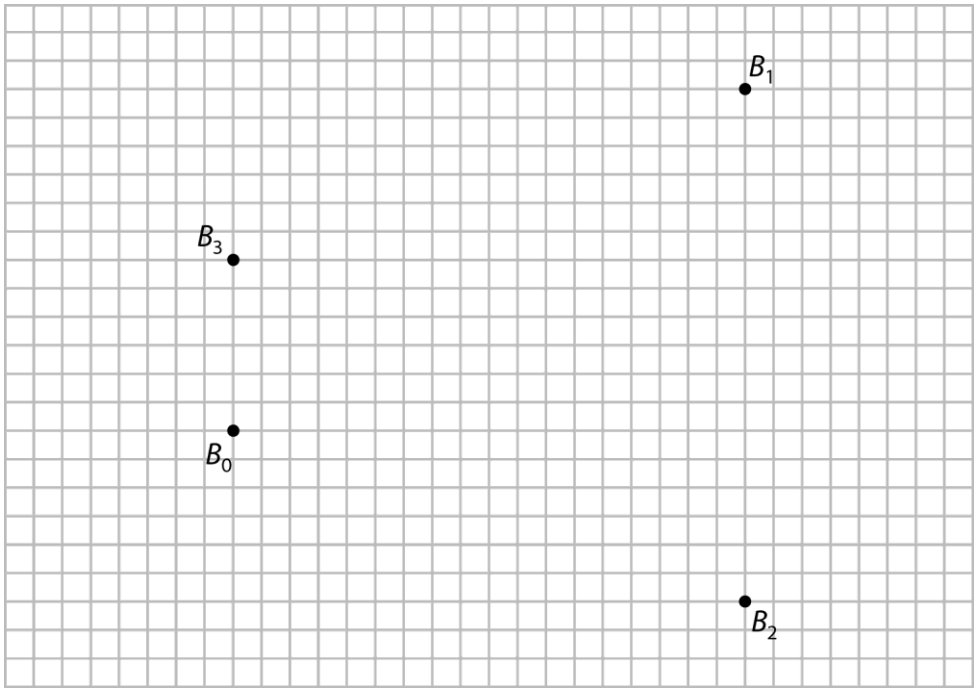
b) (8 points) For the same Bezier control points, shown again below, construct all of the de Casteljau lines and points needed to evaluate the curve at u=1/2. Label this point Q(1/2).Then add the point from part (a) and label it Q(1/4). Now sketch the path the Bezier curve will take. The curve does not need to be exact, but it should conform to some of the geometric properties of Bezier curves (convex hull condition, tangency at endpoints).



c) (8 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 $\boldsymbol{\tau = 1}$. Assume that we insert "phantom" control points $C_{-1}=C_0$ and $C_4=C_3$, so that the spline is endpoint interpolating. You must mark each Bezier point (*including any that coincide with a Catmull-Rom control 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, respecting the properties of Bezier curves noted above.



d) (10 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 we insert "phantom" control points $B_{-2}=B_{-1}=B_0$ and $B_5=B_4=B_3$, so that the spline is endpoint interpolating. You must mark each Bezier point (*including any that coincide with a de Boor 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, respecting the properties of Bezier curves noted above.



Problem 3 | Euler Integration (10 points)

Suppose a 2D particle with mass $m = 2$ has initial position and velocity:

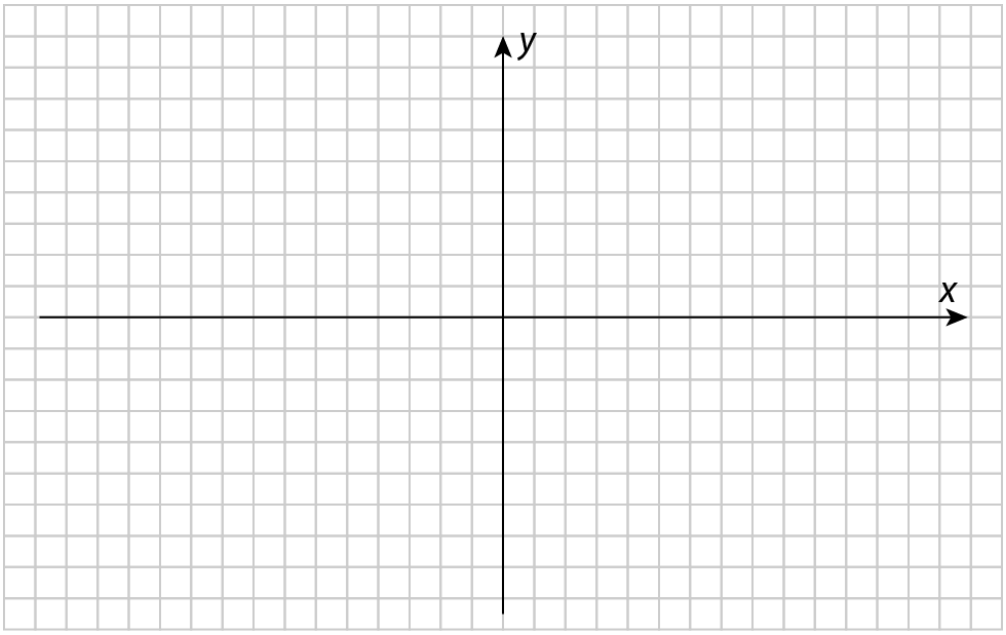
$$\mathbf{x}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{v}_0 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

and is traveling in a force field (defined over spatial position $\mathbf{x} = [x \quad y]^T$):

$$f(\mathbf{x}) = \begin{bmatrix} -y \\ -x \end{bmatrix}$$

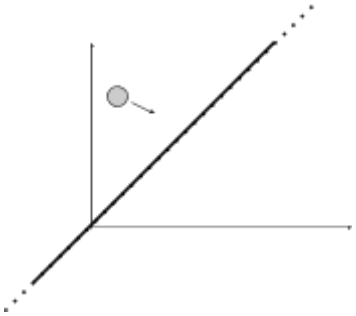
a) (7 points) Using Euler integration with a time step of $\Delta t = 2$, solve for the position and velocity of the particle over three time steps. I.e., solve for $\mathbf{x}_1$ and $\mathbf{v}_1$, $\mathbf{x}_2$ and $\mathbf{v}_2$, and $\mathbf{x}_3$ and $\mathbf{v}_3$. Show your work.

b) (3 points) Plot the particle positions on the graph below and connect them with line segments to form a trajectory.



Problem 4 | Collisions (15 points)

Suppose you have a 2D particle traveling with constant velocity $\mathbf{v} = [2 \quad -1]$ and there is a "wall" at a 45-degree angle that passes through the origin, as shown in the figure (you can assume the wall is infinite).



a) (2 points) Compute the plane equation for the "wall". (Hint: consider simplifying this equation to save you time in the next steps - not required for credit, just a suggestion).

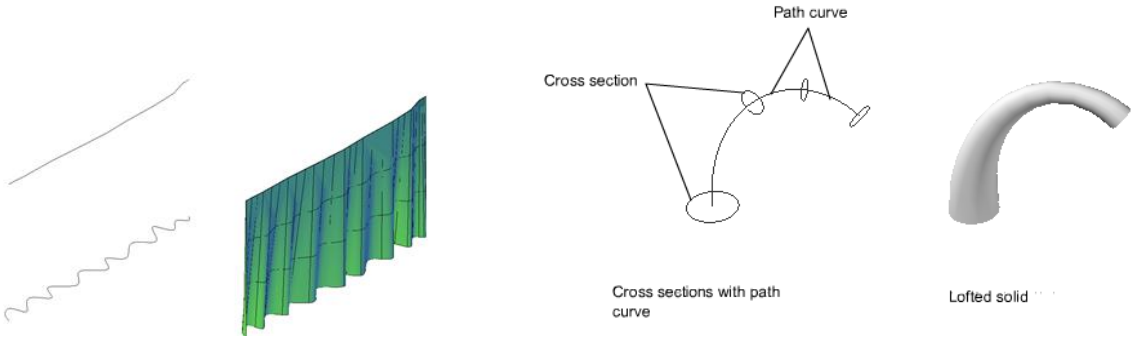
b) (5 points) If the particle is at position $\mathbf{x} = [1 \quad 5]$ at time $t = 0$ compute the time at which the particle will collide with the "wall".

c) (3 points) Compute the position of the particle at the time of the collision.

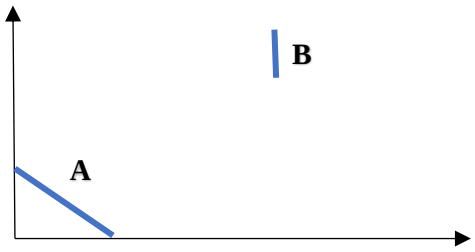
d) (5 points) Compute the new velocity vector $\mathbf{v}'$ after the collision has occurred (assume a restitution coefficient of 1).

Problem 5 | A Simplified "Loft" in 2D (25 points)

In CAD, the "loft" feature connects two curves to generate a surface (images from Autodesk website):



In this problem we will create a simplified version in 2D using some of the ideas we learned about sweep surfaces. Our loft will connect the two line segments A and B shown here:



Where A is the a line segment connecting points [0, 18] and [24 , 0] and B is the a line segment connecting points [60 , 50] and [60 , 40].

a) (5 points) We will define path curve for the loft connecting the center of A to the center of B. This path should be a cubic Bezier curve. Since we know the end points we can directly determine V_0 and V_3. We will define V_1 such that the vector connecting V_0 and $V_1$is orthogonal to the line segment A and its distance is $2/6$ of the distance between V_0 and V_3. Similarly, will define V_2 such that the vector connecting V_3 and V_2 is orthogonal to the line segment B and its distance is $2/6$ of the distance between V_0 and V_3. Compute V_0, V_1 V_2, and V_3.

(note: we have not specified the direction of the vectors connecting V_0 and V_1 and V_2 and V_3 – choose them so that the resulting "loft" would be most likely to satisfy the intent of a user applying your algorithm for design).

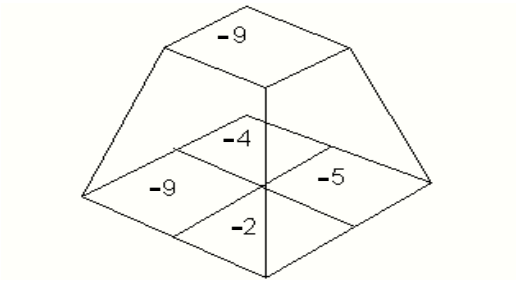
b) (5 points) Let $Q(u)$ be the Bezier curve defined by V_0, V_1 V_2, and V_3. Compute $Q\left(\frac{1}{2}\right)$ and sketch the Bezier curve (include V_0, V_1 V_2, and V_3 and $Q\left(\frac{1}{2}\right)$ in your drawing).

c) (7 points) Compute the cross section in the middle of the loft. Note it will be the line segment L centered at $Q\left(\frac{1}{2}\right)$ and with length equal to the average of the length of A and B. To determine the orientation, consider a simplified version of the Frenet frame discussed in class for 2D. Report the two end points of the lines segment L. (Hint: In your computation of $\frac{Q'\left(\frac{1}{2}\right)}{\left|Q'\left(\frac{1}{2}\right)\right|}$, feel free to round off to second decimal point for simplicity later on).

d) (8 points) Draw a coarse surface created by connecting the 3 cross sections of the loft A, L, and B (note this will look like a closed polygon with 6 vertices).

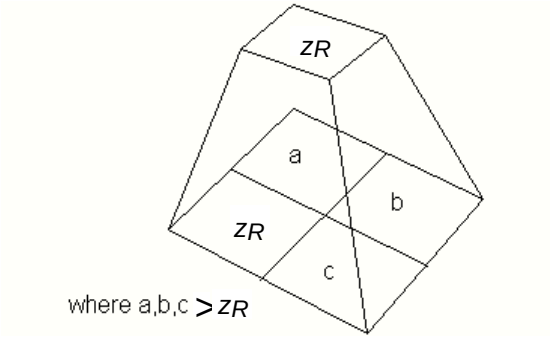
Problem 6. Z-buffer (14 points)

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) (3 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) (3 points) What can we conclude if $z_R < z_T$?

c) (3 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) (5 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?