CS 130, Final

Solutions

This exam was originally given in two parts due to COVID-19. The first seven problems were given in a timed 40-minute format. The rest of the exam was a 48-hour take-home exam. A full score was given for receiving 90 of the 100 possible points.

Problem 1 (2 points)

GPU's must render millions of triangles per frame of a game. To do this, they have hundreds or even thousands of threads. A simple strategy to parallelize rasterization is for each thread (or group of threads) to have a separate image and z-buffer. Each thread group rasterizes its fair share of the triangles to its own image and z-buffer. This then leads to a new problem - these separate images and z-buffers must be merged into one to produce the correct final result. Describe a simple algorithm to correctly merge the image and z-buffer from two different threads into one.

For each pixel, identify the z-buffer that stores the smaller value. Copy the color and z value from that z-buffer and the corresponding image into the output for that pixel.

Problem 2 (2 points)

The clipper tends to remove the significant majority of triangles. Since the vertex shader can be quite expensive, running the vertex shader after clipping could result in a significant performance boost. Nevertheless, this is never done. Why do we run the vertex shader before performing clipping?

The vertex shader transforms vertices into normalized device coordinates. Since the triangles need to be in normalized device coordinates in order to do clipping, it is not possible to clip before running the vertex shader.

Problem 3 (5 points)

Which of the following pairs of transformations commute? (Operations commute if changing the order does not change the result.) You do not need to justify your answer, just list the ones that commute. (A) Non-uniform scale by 2 in the y direction, uniform scale by 3. (B) Rotate by 30 degrees in the $x-y$ plane, non-uniform scale by 2 in the z direction. (C) Rotate by 30 degrees in the $x-y$ plane, non-uniform scale by 2 in the y direction. (D) Rotate by 30 degrees in the $x-y$ plane, rotate by 50 degrees in the $x-y$ plane. (E) Rotate by 30 degrees in the $x-y$ plane, rotate by 50 degrees in the $y-z$ plane. (F) Rotate by 30 degrees in the $x-y$ plane, translate 2 units in the z direction. (G) Rotate by 30 degrees in the $x-y$ plane, translate 2 units in the y direction. (H) Translate 3 units in the x direction, translate 2 units in the y direction. (I) Translate 3 units in the x direction, non-uniform scale by 3 in the y direction. (J) Translate 3 units in the x direction, uniform scale by 3.

These commute: A, B, D, F, H, I

Problem 4 (2 points)

What geometrical transformation does this 2D homogeneous transformation matrix correspond to? $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$.

It scales by 1/3 in the x direction and 2/3 in the y direction. Note that it scales the w by 3, so that when the perspective divide is performed, this 3 is divided from the x and the y.

Problem 5 (2 points)

A ray tracing framework such as we used for the first project allows us to calculate all of the intersections between a ray and an object. More generally, these objects could be very complex (especially an object represented by a triangle mesh). One application of this intersection capability is the ability to classify a given point as being inside or outside of bounded objects. (These objects are closed surfaces without holes, and the objects do not go to infinity.) Suggest an algorithm for answering the inside/outside query given a list of ray-object intersections. You may ignore degeneracies.

Count the number of intersections. If it is even, the point is outside. If it is odd, the point is inside. If you are walking along the ray, each time you intersect the surface you pass inside to outside or outside to inside. If you start walking along the ray at infinity, you are outside. Note that there is a degenerate possibility, which is that you graze the surface but do not cross it. This possibility complicates real-world implementations of this approach.

Problem 6 (2 points)

In class, we discussed different strategies for rasterizing the boundary of an implicit surface $f(x) = 0$. If we assume that $f(x) < 0$ corresponds to inside, then it also makes sense to talk about rasterizing the interior of the object as well. Suggest an algorithm for rasterizing the interior of the implicit object.

For each pixel, test to see if the pixel is inside the object (by calling the function). If it is inside, set the pixel.

Problem 7 (2 points)

On the second project, we rasterized a triangle ABC by doing inside-outside tests on points P using the signed areas of triangles ABC, PBC, APC, and ABP. The algorithm relied critically on the signs of these areas. It is actually possible to perform these inside-outside tests using only the unsigned areas. Explain how to do this.

P is inside ABC if and only if the sum of the unsigned areas of PBC, APC, and ABP equals the unsigned area of ABC. Framed in terms of barycentric coordinates, the point P is inside the triangle if and only if the barycentric coordinates add up to 1.

Problem 8 (6 points)

Construct a 3×3 matrix that performs each of the following 2D operations (in homogeneous coordinates). You may express it as the product of other matrices if you prefer.

(a) Rotate 45° counterclockwise about the origin.

$$\begin{pmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} & 0 \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(b) Translate 2 units in the positive x direction.

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(c) Rotate 90° counterclockwise about the point $(1, 2)$.

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 3 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

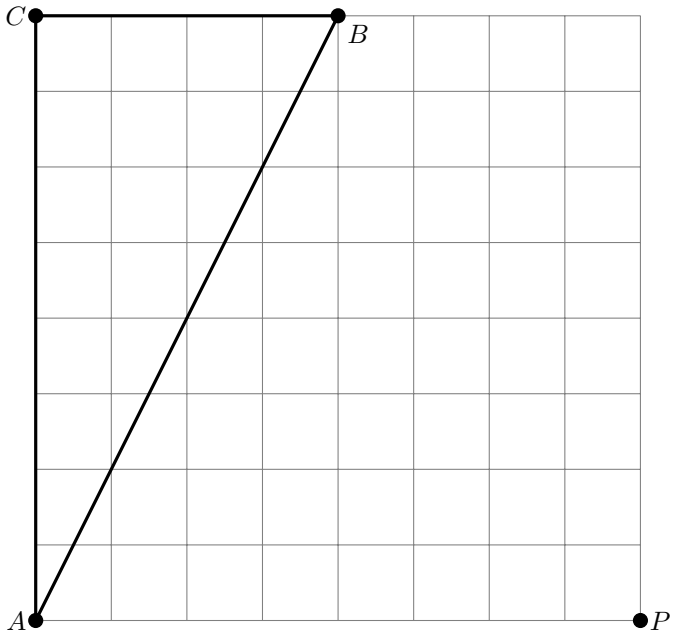
Problem 9 (2 points)

During rasterization, we had to convert from normalized device coordinates ($-1 \le x \le 1$, $-1 \le y \le 1$) to pixel indices, where the image was $w \times h$ pixels in size. If instead our starting coordinate system was $0 \le x \le 1$, $0 \le y \le 1$, derive formulas for computing pixel indices (i, j) from starting coordinates (x, y).

Our pixel indices are $(0, 0)$ to $(w-1, h-1)$ and represent the *center* of the pixels. Thus, for x we must map $[0, 1]$ to $[-\frac{1}{2}, w-\frac{1}{2}]$, which is accomplished by $i = wx - \frac{1}{2}$. Similarly, $j = hy - \frac{1}{2}$.

Problem 10 (2 points)

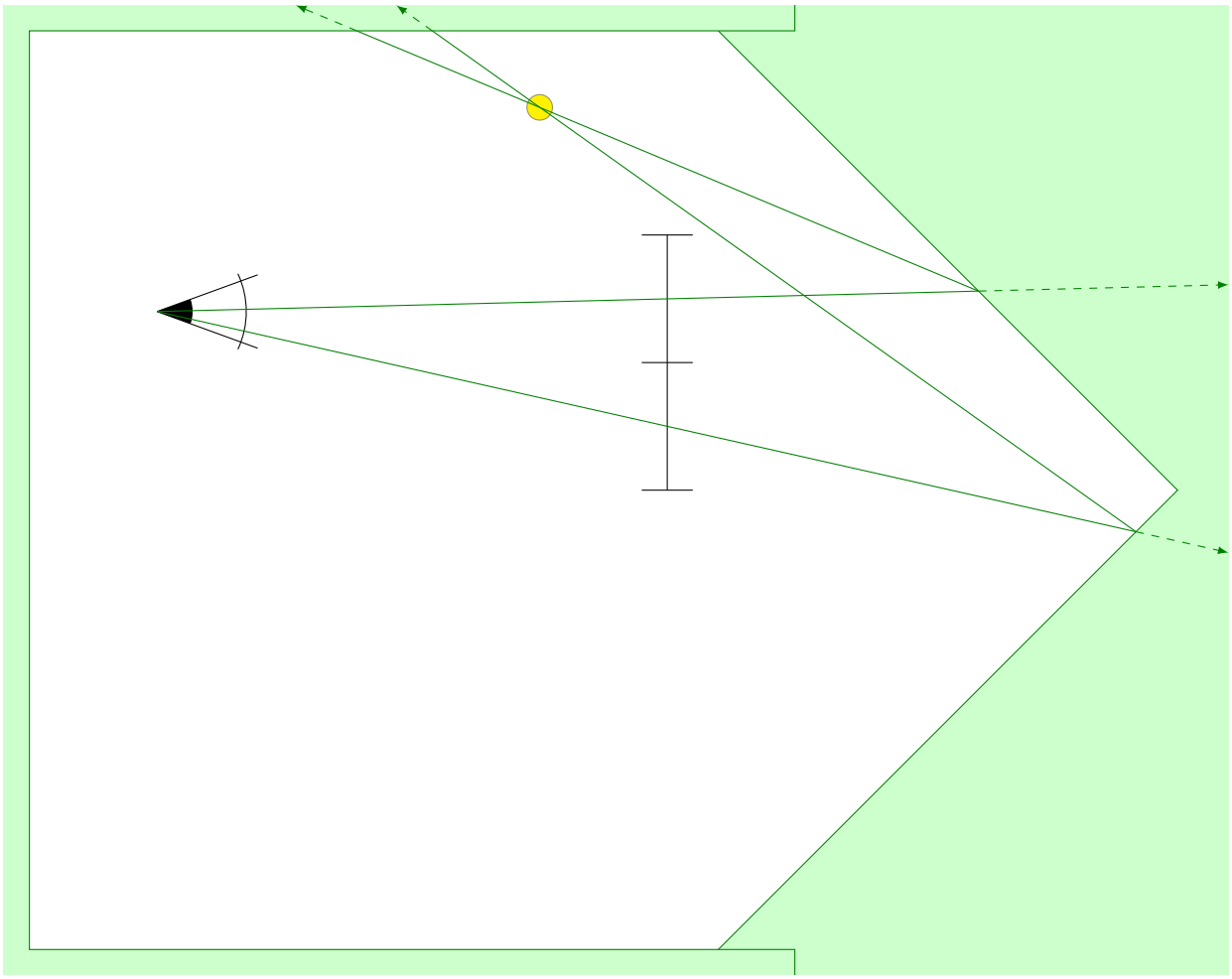
Compute the barycentric weights of the point P.



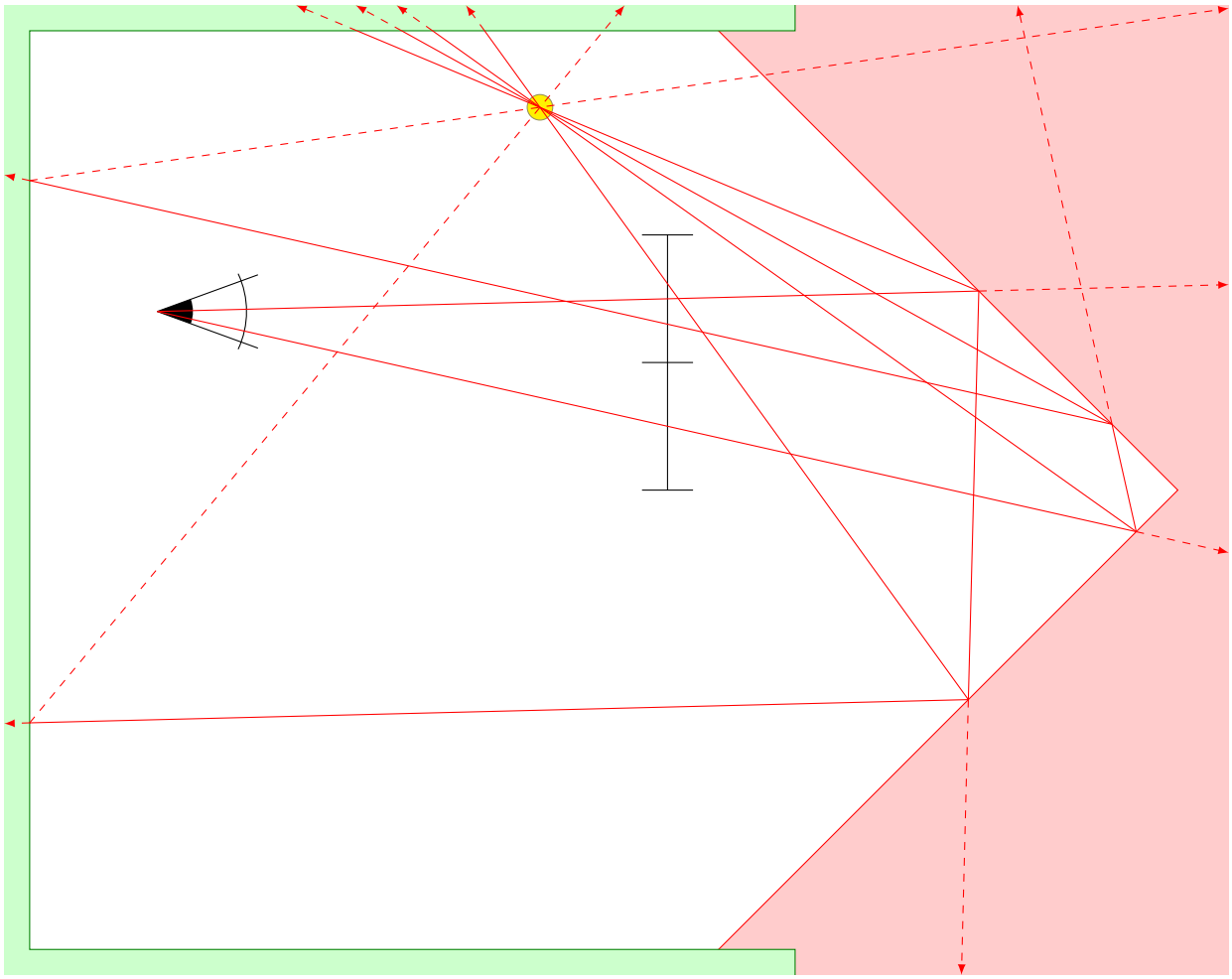
Noting that barycentric coordinates are constant along lines parallel to the triangle edges and equally spaced, we immediately conclude that $\alpha = 1$ and $\beta = 2$, from which $\gamma = 1 - \alpha - \beta = -2$. The coordinates are also straightforward to compute with triangle areas, since all of the triangles have a horizontal or vertical edge.

In each of the raytracing problems, green objects are wood, red objects are reflective, and blue objects are transparent. The scenes are in 2D with a 1D image. Each image has two pixels. yellow circles are point lights; the ray tracer supports shadows. Draw all of the rays that would be cast while raytracing each scene. Use a maximum recursion depth of 3. (Don't worry about precisely what counts as depth 3 or depth 4; I just care that recursion is being performed correctly when necessary and that important rays are not missing. No problem contains more than 20 rays in the "exact" solution.)

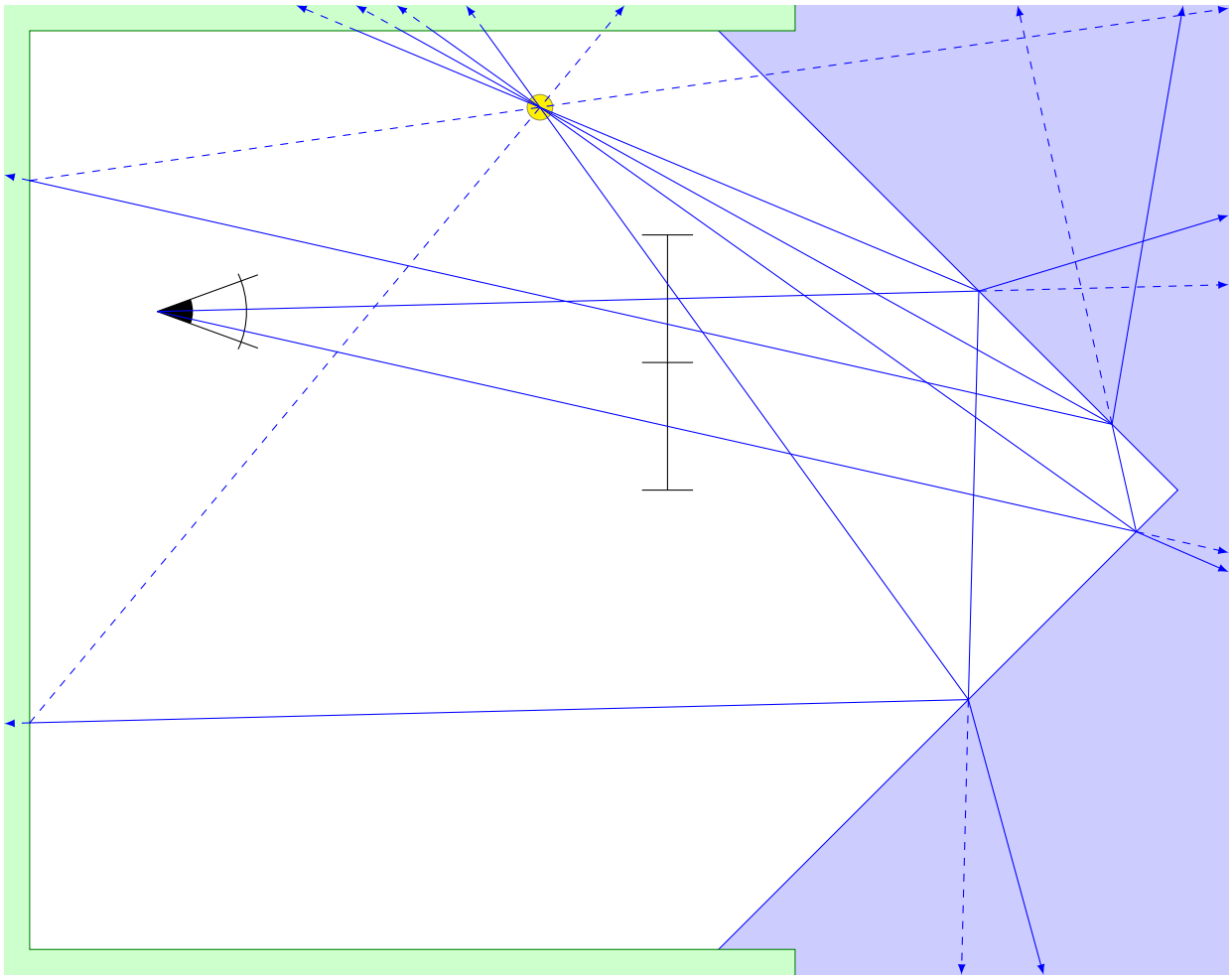
Problem 11 (6 points)



Problem 12 (6 points)



Problem 13 (6 points)



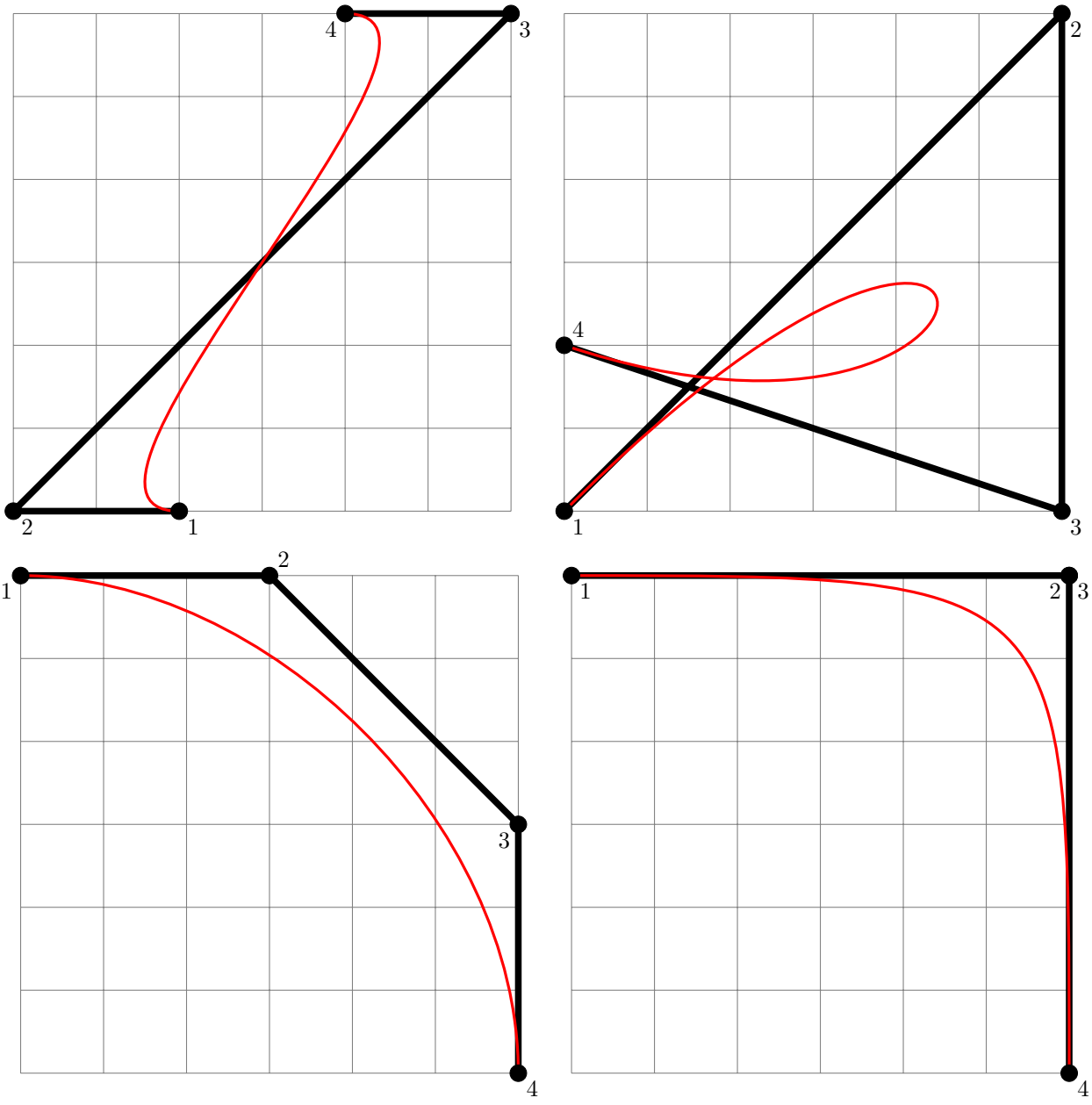
Problem 14 (2 points)

Construct a homogeneous transform matrix that performs the transformation $\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \frac{x}{x+y+z} \\ \frac{y}{x+y+z} \\ \frac{z}{x+y+z} \end{pmatrix}$.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

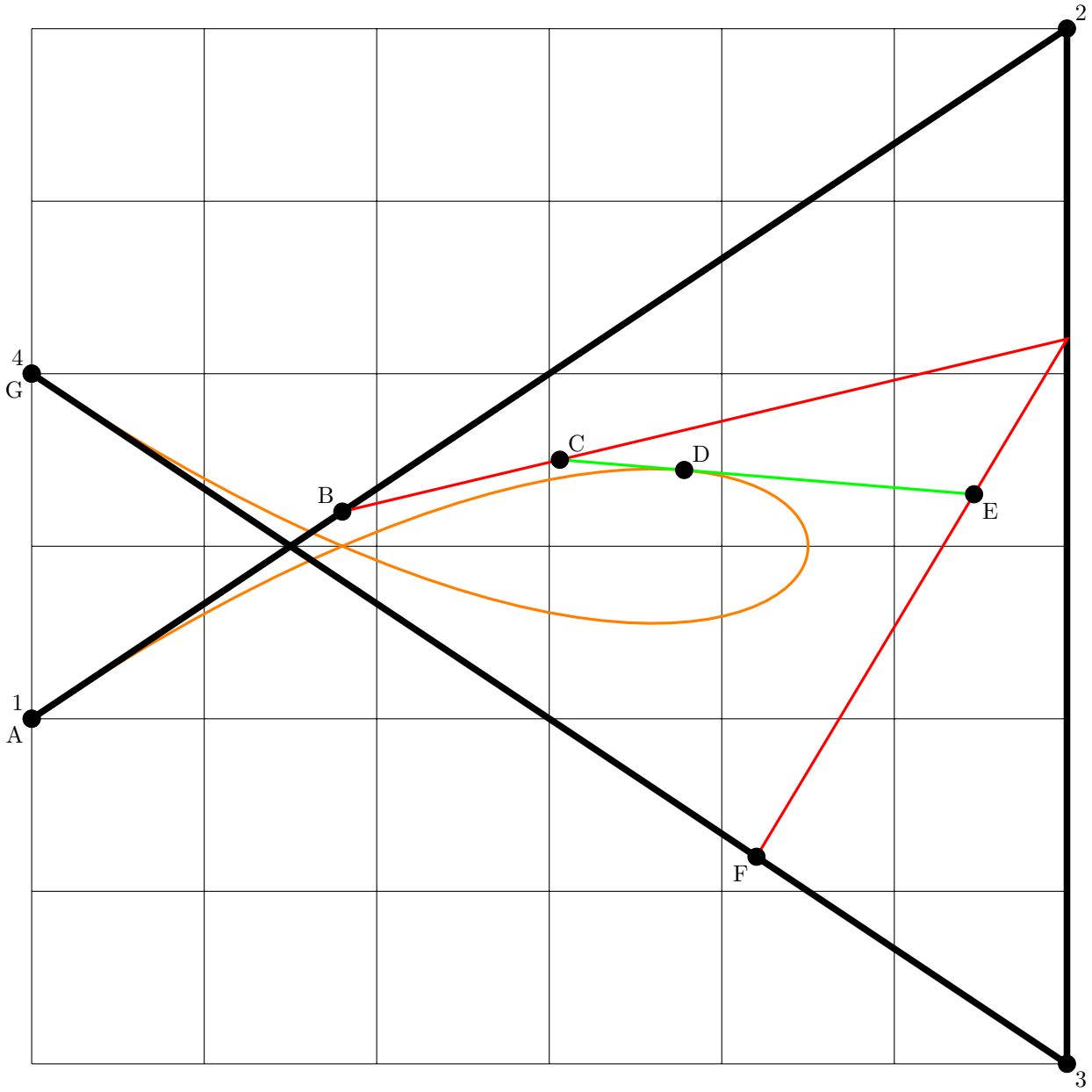
Problem 15 (4 points)

In each of the four examples below, control points are shown for a cubic Bezier curve. Sketch out approximately what these curves will look like.



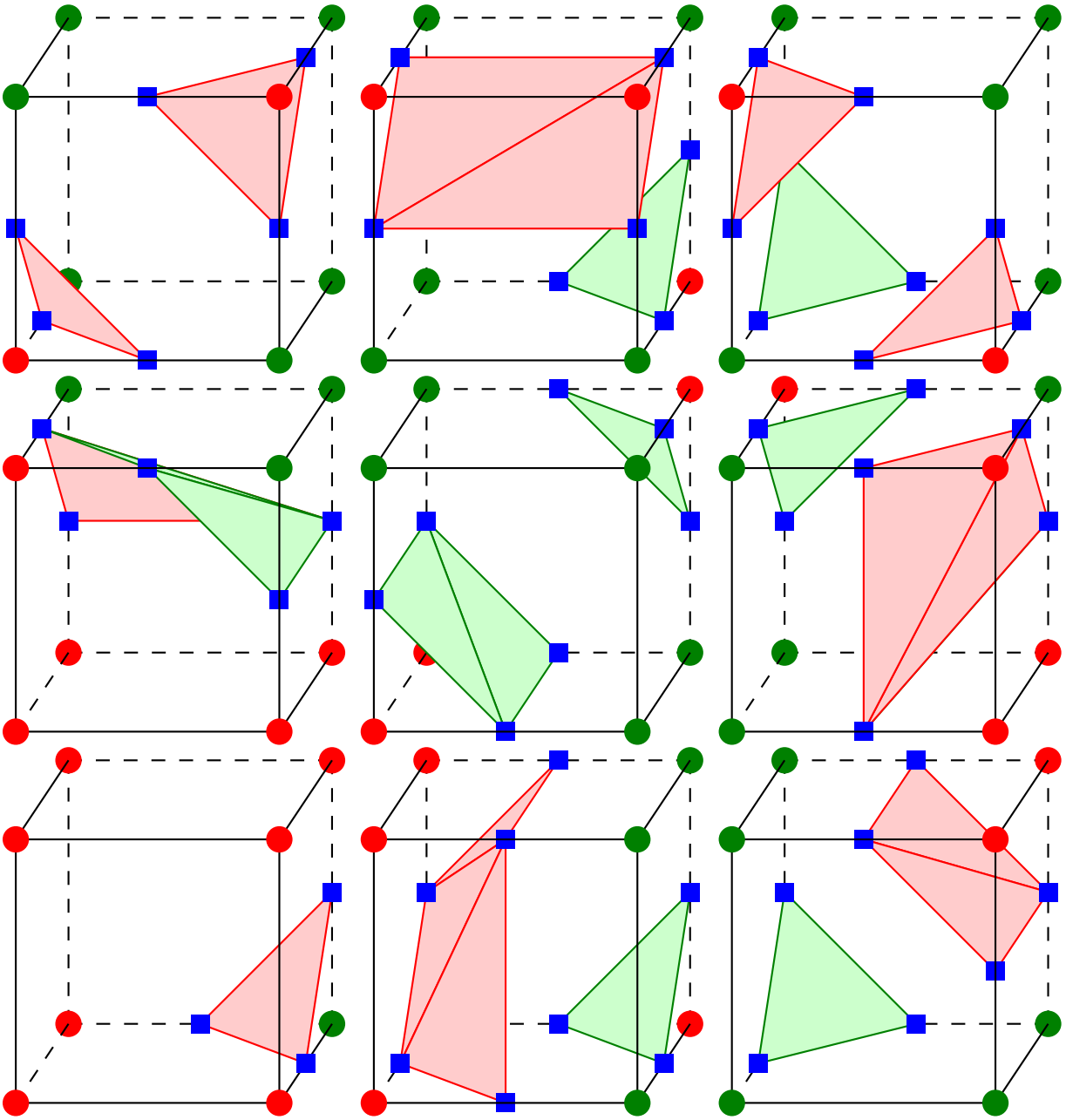
Problem 16 (4 points)

Geometrically subdivide the Bézier curve given by the control points below and at $t = 0.3$ along the curve. Label the new control points A, B, C, ..., G. (The two resulting Bézier curves should share their common point, so only 7 control points are required.)



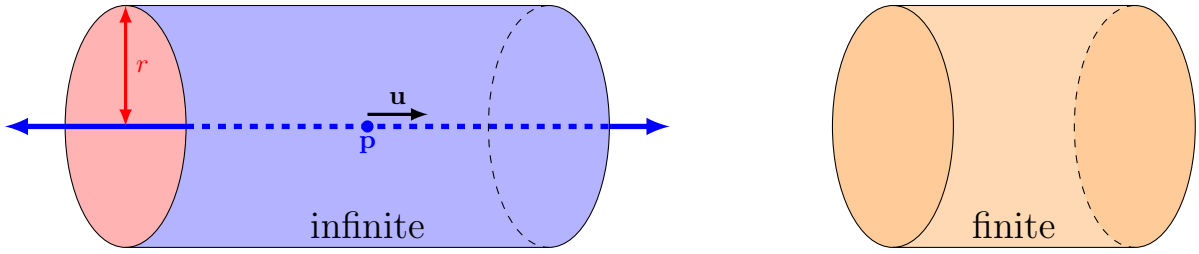
Problem 17 (9 points)

Construct a consistent marching cubes triangulation of the cubes shown below. The cubes should actually be touching, but they have been separated apart for clarity. Draw squares at the vertices of your triangles.



Problem 18 (12 points)

In this problem, we go through the implementation of a cylinder ray tracing primitive, in much the form that it might be implemented in your ray tracing project. An *infinite* cylinder is defined by a point $\mathbf{p}$, a direction $\mathbf{u}$ (with $\|\mathbf{u}\| = 1$), and a distance r. The cylinder's axis is the line with point $\mathbf{p}$ and direction $\mathbf{u}$. The cylinder consists of all points whose distance to the axis is r. All sub-parts are independent and can be solved in any order, even if earlier parts have not been solved.



(a) Show that an implicit equation for the cylinder is given by $f(\mathbf{x}) = 0$, $f(\mathbf{x}) = \|(\mathbf{x}-\mathbf{p}) - ((\mathbf{x}-\mathbf{p})\cdot\mathbf{u})\mathbf{u}\|^2 - r^2$.

The vector $\mathbf{y} = \mathbf{x}-\mathbf{p}$ points from the axis to the query point. Projecting this onto the axis we get a component $(\mathbf{y}\cdot\mathbf{u})\mathbf{u}$ along the axis and $\mathbf{y} - (\mathbf{y}\cdot\mathbf{u})\mathbf{u}$ orthogonal to it. The length of the latter piece is the distance to the axis. The cylinder formula is obtained by requiring the square of the distance to be r^2.

(b) Does $f(\mathbf{x}) < 0$ correspond to inside or outside? You must justify your answer.

A simple way to the point $\mathbf{p}$ on the axis, which is inside. $f(\mathbf{p}) = -r^2 < 0$, so negative is inside.

(c) Write pseudocode to calculate *all* intersections t between the *line* given by $g(t) = \mathbf{e} + t\mathbf{w}$ (with $\|\mathbf{w}\| = 1$) and the infinite cylinder. Note that we are not intersecting with a *ray* here. We will deal with that later. You should compute a sorted list of the intersections (not just the closest one). You do not need to use C++ syntax (math syntax is okay, and you need not use C++ data structures), but your pseudocode must contain all of the formulas and logic needed calculate the results, and your code must handle all of the (non-degenerate) cases. You may assume the line given by $g(t)$ is not parallel to the axis of the cylinder.

$$\mathbf{x} = \mathbf{e} + t\mathbf{w}$$
$$\|(\mathbf{x}-\mathbf{p}) - ((\mathbf{x}-\mathbf{p})\cdot\mathbf{u})\mathbf{u}\|^2 = r^2$$
$$\|(\mathbf{e}+t\mathbf{w}-\mathbf{p}) - ((\mathbf{e}+t\mathbf{w}-\mathbf{p})\cdot\mathbf{u})\mathbf{u}\|^2 = r^2$$
$$\mathbf{a} = (\mathbf{e}-\mathbf{p}) - ((\mathbf{e}-\mathbf{p})\cdot\mathbf{u})\mathbf{u}$$
$$\mathbf{b} = \mathbf{w} - (\mathbf{w}\cdot\mathbf{u})\mathbf{u}$$
$$\|\mathbf{a}+t\mathbf{b}\|^2 = r^2$$
$$\mathbf{a}\cdot\mathbf{a} - r^2 + 2\mathbf{a}\cdot\mathbf{b}t + \mathbf{b}\cdot\mathbf{b}t^2 = 0$$
$$t = \frac{-2\mathbf{a}\cdot\mathbf{b} \pm \sqrt{(2\mathbf{a}\cdot\mathbf{b})^2 - 4(\mathbf{a}\cdot\mathbf{a} - r^2)(\mathbf{b}\cdot\mathbf{b})}}{2\mathbf{b}\cdot\mathbf{b}}$$
$$d = (\mathbf{a}\cdot\mathbf{b})^2 - (\mathbf{a}\cdot\mathbf{a} - r^2)(\mathbf{b}\cdot\mathbf{b})$$
$$t_0 = \frac{-\mathbf{a}\cdot\mathbf{b} - \sqrt{d}}{\mathbf{b}\cdot\mathbf{b}} \qquad t_1 = \frac{-\mathbf{a}\cdot\mathbf{b} + \sqrt{d}}{\mathbf{b}\cdot\mathbf{b}}$$

The algorithm is to compute $\mathbf{a}$, $\mathbf{b}$, and d. If $d < 0$, return the empty list. Otherwise, return (t_0, t_1).
Note that we do not need to worry about the line being entirely inside the cylinder, since that is only possible if the line is parallel to the axis of the cylinder.

(d) In the previous step, we computed an intersection list $(a, b, c, \ldots, d)$ for the intersection of a cylinder with a *line*. In this problem, we will start with the intersection list for a *line*. The input list will always contain an even number of entries in sorted order, possibly with duplicates. The first entry of the list might be $-\infty$, and the last entry might be ∞. Thus, $(-\infty, 1, 3, \infty)$, $(-3, \infty)$, $(-1, -2, 1, 4, 4, 7)$, $(-\infty, \infty)$, $(-\infty, -3)$, and $()$ are all valid inputs. $-\infty$ and ∞ are never duplicated. Construct an algorithm that returns the corresponding intersection list for the *ray* corresponding to the same line but with $t \geq 0$. Your algorithm must work for any (even) size list.

Visit the list one pair (r, s) of entries at a time in order, noting that the full list is sorted and has an even number of entries. If $s < 0$, discard the pair. Otherwise, if $r < 0$ then replace the pair with $(0, s)$. Otherwise, leave the pair unchanged.
An alternative solution is to remove the negative entries and then prepend 0 if the number of entries removed was odd.

(e) Compute the normal direction for an intersection that occurs at location $\mathbf{x}$.

Following the original derivation of the formula, let $\mathbf{y} = \mathbf{x} - \mathbf{p}$ and $\mathbf{z} = \mathbf{y} - (\mathbf{y} \cdot \mathbf{u})\mathbf{u}$. Then, $\mathbf{n} = \frac{\mathbf{z}}{\|\mathbf{z}\|} = \frac{\mathbf{z}}{r}$. Remember that the normal direction must be normalized.

(f) Normally, one is interested in rendering finite cylinders (with a top and bottom cap). Explain how the infinite cylinder primitive developed above, along with other features and primitives available in your ray tracer, can be used to render finite cylinders.

A finite cylinder can be obtained as the intersection of an infinite cylinder and two planes (half-spaces) for the end caps.

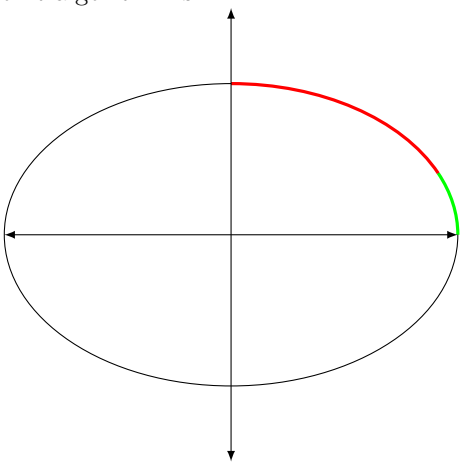
The next problem is an extension of a similar problem on the final from the Fall 2018 offering of this course. You may want to review the solutions to that final (available for download on the website for that course) before attempting this problem. This problem is self-contained and can be solved on its own merits. All sub-parts are independent and can be solved in any order, even if earlier parts have not been solved.

Problem 19 (18 points)

In this problem, we will adapt the midpoint algorithm to rasterize the outline of an ellipse. The ellipse is given by the equation $(x/a)^2 + (y/b)^2 = 1$, where a, b are positive integers. We will handle the red and green parts separately. The general shell for an algorithm like the midpoint algorithm is

```
void rasterize_ellipse(int a, int b)
{
  // loop A
  for(int i = x_A, j = y_A; stop_A; i++)
  {
    draw(i,j);
    if(update_criterion_A)
      j++; // or j--;
  }

  // loop B
  // TODO
}
```



(a) Let's start with loop A, which corresponds to the red portion of the ellipse. What values should be used for `x_A` and `y_A`?

`x_A=0;` and `y_A=b;`

(b) In part A, should we use `j++` or `j--`? Why?

`j--;` As we move right (`i++;`) the curve is decreasing, so `j` must decrease.

(c) Show that the slope of the ellipse at the point (x, y) is $m = -\frac{b^2 x}{a^2 y}$. [Hint: use implicit differentiation.]

$$\begin{aligned} \frac{x^2}{a^2} + \frac{y^2}{b^2} &= 1 \\ \frac{2x}{a^2} + \frac{2yy'}{b^2} &= 0 \qquad \text{implicit differentiation} \\ \frac{2yy'}{b^2} &= -\frac{2x}{a^2} \\ y' &= -\frac{b^2 x}{a^2 y} \end{aligned}$$

(d) At what slope of the ellipse should we stop loop A? Using the formula for the slope from the previous problem ($m = -\frac{b^2 x}{a^2 y}$), find a reasonable stopping criterion for `stop_A`. You are not allowed to use division in

your criterion. [Hint: express the stopping criterion first as a criterion on m. Then, use the formula for m and simplify the criterion.]

$$
\begin{aligned}
m &\geq -1 \\
-\frac{b^2 x}{a^2 y} &\geq -1 \\
\frac{b^2 x}{a^2 y} &\leq 1 \\
b^2 i &\leq a^2 j \qquad \texttt{stop\_A}
\end{aligned}
$$

(e) For `update_criterion_A`, we need a function $g(x, y)$ such that $g(x, y) \leq 0$ if and only if (x, y) is inside the ellipse. Suggest a suitable function $g(x, y)$. (You are not allowed to use divisions or square roots.)

This is obtained from the equation for the ellipse by clearing the denominators. We can check the overall sign by testing $g(0, 0) < 0$. $g(x, y) = b^2x^2 + a^2y^2 - a^2b^2$

(f) For `update_criterion_A`, we want to test $g(i + a, j + b)$ at one point $(i + a, j + b)$. What should we use for the constants a and b?

$a = 1$, $b = -\frac{1}{2}$. Recall that we are testing the point halfway between $(i + 1, j)$ and $(i + 1, j - 1)$.

(g) Finally, use $g(x, y)$ to construct `update_criterion_A`.

We want $g(i + 1, j - \frac{1}{2}) > 0$.

(h) Loop B (for the green portion of the ellipse) can be constructed by copying Loop A and modifying it. What modifications must be made to Loop A to construct Loop B?

Swap a, b everywhere, and replace `draw(i,j)` with `draw(j,i)`. Note that swapping a, b results in the same ellipse, but mirrored (swap i, j). The green portion of the original ellipse looks exactly like the red portion of the mirrored ellipse. So we can get the second loop by rasterizing the red portion of the mirrored ellipse and then swapping the coordinates before drawing them.
Alternative: Continue from where the first loop left off. Stop criterion is $j \geq 0$. `j--;` unconditionally; `i++` conditionally. Condition stays the same. *This approach will produce lower-quality asymmetrical ellipses.*
Alternative: Start from $i = a, j = 0$. Stop condition should be reversed. `j++;` unconditionally; `i--` conditionally. Condition becomes $g(i - \frac{1}{2}, j + 1) > 0$.

(i) The algorithm now draws only the red and green portions of the ellipse. Suggest a simple (and very efficient) modification that can be made to the algorithm above so that it will draw rest of the ellipse as well.

A pixel location in each of the four portions of the circle can be deduced from the one pixel computed: `draw(i,j);draw(i,-j);draw(-i,j);draw(-i,-j);`. A similar modification is needed for Loop B as well.