CS 130, Final

Solutions

1	2	3	4	5	6	7	8	9	10	11	12	total

Read the entire exam before beginning. **Manage your time carefully.** This exam has 50 points; you need 40 to get full credit. Additional points are extra credit. 50 points → 3.6 min/point. 40 points → 4.5 min/point.

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)

Compute the normal to the triangle with vertices $A = (1, 0, 0)$, $B = (0, 2, 0)$, and $C = (0, 0, 3)$. Don't worry about whether the normal points inside or outside.

$$N = (B - A) \times (C - A) = \begin{pmatrix} -1 \\ 2 \\ 0 \end{pmatrix} \times \begin{pmatrix} -1 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \\ 2 \end{pmatrix}$$

$$\|N\| = 7$$

$$n = \frac{N}{\|N\|} = \begin{pmatrix} 6/7 \\ 3/7 \\ 2/7 \end{pmatrix}$$

Problem 3 (2 points)

The 4×4 transformation matrix

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

expresses a transformation from one 3D point (x, y, z) to another 3D point (x', y', z'). Find x', y', and z'.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 1 & 2 & 3 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ 2y \\ 3z \\ x + 2y + 3z \end{pmatrix}$$

$$x' = \frac{x}{x + 2y + 3z} \qquad y' = \frac{2y}{x + 2y + 3z} \qquad z' = \frac{3z}{x + 2y + 3z}$$

Problem 4 (3 points)

Construct 4×4 transformation matrices that accomplish each of the following:
(a) Rotation about the y axis by 90 degrees (you can rotate clockwise or counterclockwise).
(b) Translate in the y direction by 3.
(c) Scale in the y direction by 3.
(d) Translate in the y direction by 3, followed by a scale in the y direction by 3.
(e) Scale in the y direction by 3, followed by a translation in the y direction by 3.
(f) Swap the x and y components.

$$(a) = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (b) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (c) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(d) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 9 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (e) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (f) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note that (d) and (e) are different.

Problem 5 (2 points)

Why do we use red, green, and blue when producing images? Why not two or four colors instead of three? Why not a different set of three colors?

The human eye uses three types of color-sensitive cells to distinguish colors; these cells are most sensitive to red, green, and blue light. The eye thus perceives other colors as combinations of these colors.

Problem 6 (2 points)

Outdoor temperatures rise throughout the middle of the day due to the energy received from the sun, but outdoor temperatures start cooling off before sunset. Why would temperatures start getting cooler while the sun is still in the sky? Why don't temperatures keep increasing until sunset? (This can be understood in terms of concepts introduced in this course.)

The amount of energy an area gets from the sun depends on the angle between the sun and vertical (by $\cos\theta$ in fact), since the same amount of incoming energy is spread over a larger area of ground. This is exactly the same phenomenon being captured by the Lambertian shading model.

Problem 7 (2 points)

Rotation matrices P and Q satisfy the property $P^TP = I$ and $Q^TQ = I$. Show that $R = PQ$ satisfies the same property.

$R^TR = (PQ)^T(PQ) = Q^T(P^TP)Q = Q^TQ = I.$

Problem 8 (2 points)

Given an equilateral triangle ABC, what are the values of the barycentric weights at (a) the vertex A, (b) the midpoint of segment AB, and (c) the center of the triangle?

The barycentric weights are (a) $\alpha = 1$, $\beta = \gamma = 0$, (b) $\alpha = \beta = \frac{1}{2}$, $\gamma = 0$, and (c) $\alpha = \beta = \gamma = \frac{1}{3}$.

Problem 9 (2 points)

Given the triangle with vertices $A = (0, 0)$, $B = (2, 0)$, and $C = (0, 3)$, what are the values of the barycentric weights at point (x, y)?

The barycentric weights are $\alpha = 1 - x/2 - y/3$, $\beta = x/2$, $\gamma = y/3$.

Problem 10 (2 points)

In our acceleration structure, we needed to compute the distance between consecutive cell crossings along the x, y, and z directions. Given a ray with endpoint $\mathbf{e}$ and direction $\mathbf{u}$, as well as a grid with cell sizes Δx, Δy, and Δz, compute the distance t_x along the ray between cell crossings in the x direction. You may assume the ray is not parallel to any of the cell faces.

$$t_x = \frac{\Delta x}{|u_x|}$$

Problem 11 (3 points)

The equation $2z = x^2 + y^2$ represents a paraboloid. Compute *all* of the intersection *locations* between this paraboloid and the ray with endpoint $\mathbf{e} = \langle 0, -1, 3 \rangle$ and direction $\mathbf{u} = \langle \frac{1}{3}, -\frac{2}{3}, \frac{2}{3} \rangle$.

$$
\begin{aligned}
x &= \frac{t}{3} \\
y &= \frac{-3 - 2t}{3} \\
z &= \frac{9 + 2t}{3} \\
2z &= x^2 + y^2 \\
2\left(\frac{9 + 2t}{3}\right) &= \left(\frac{t}{3}\right)^2 + \left(\frac{-3 - 2t}{3}\right)^2 \\
6(9 + 2t) &= t^2 + (3 + 2t)^2 \\
0 &= 5t^2 - 45 \\
t^2 &= 9 \\
t &= \pm 3
\end{aligned}
$$

The negative solution does not lie on the ray, so $t = 3$. Plugging this in we get the intersection location $(1, -3, 5)$.

Problem 12 (2 points)

The equation $2z = x^2 + y^2$ represents a paraboloid. Given a point (x, y, z) that is on the paraboloid, compute the normal direction at that location.

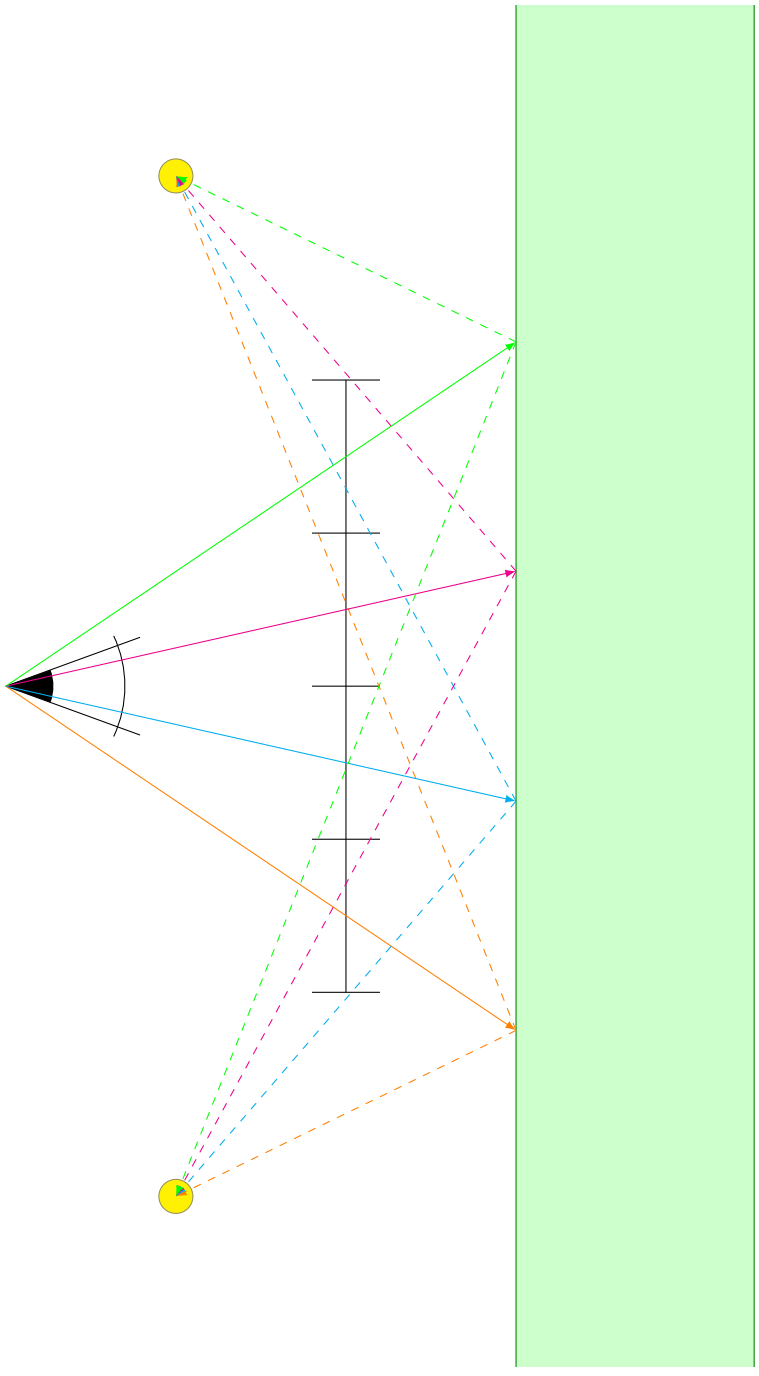
This can be done easily as a parametric equation: $(u, v, (x^2 + y^2)/2)$. It can also be done as an implicit equation: $f(x, y, z) = x^2 + y^2 - 2z = 0$. The latter approach is simpler, so we will do it that way.

$$
\nabla f = \begin{pmatrix} 2x \\ 2y \\ -2 \end{pmatrix}
$$

$$
\mathbf{n} = \frac{\nabla f}{\|\nabla f\|} = \frac{1}{\sqrt{x^2 + y^2 + 1}} \begin{pmatrix} x \\ y \\ -1 \end{pmatrix}
$$

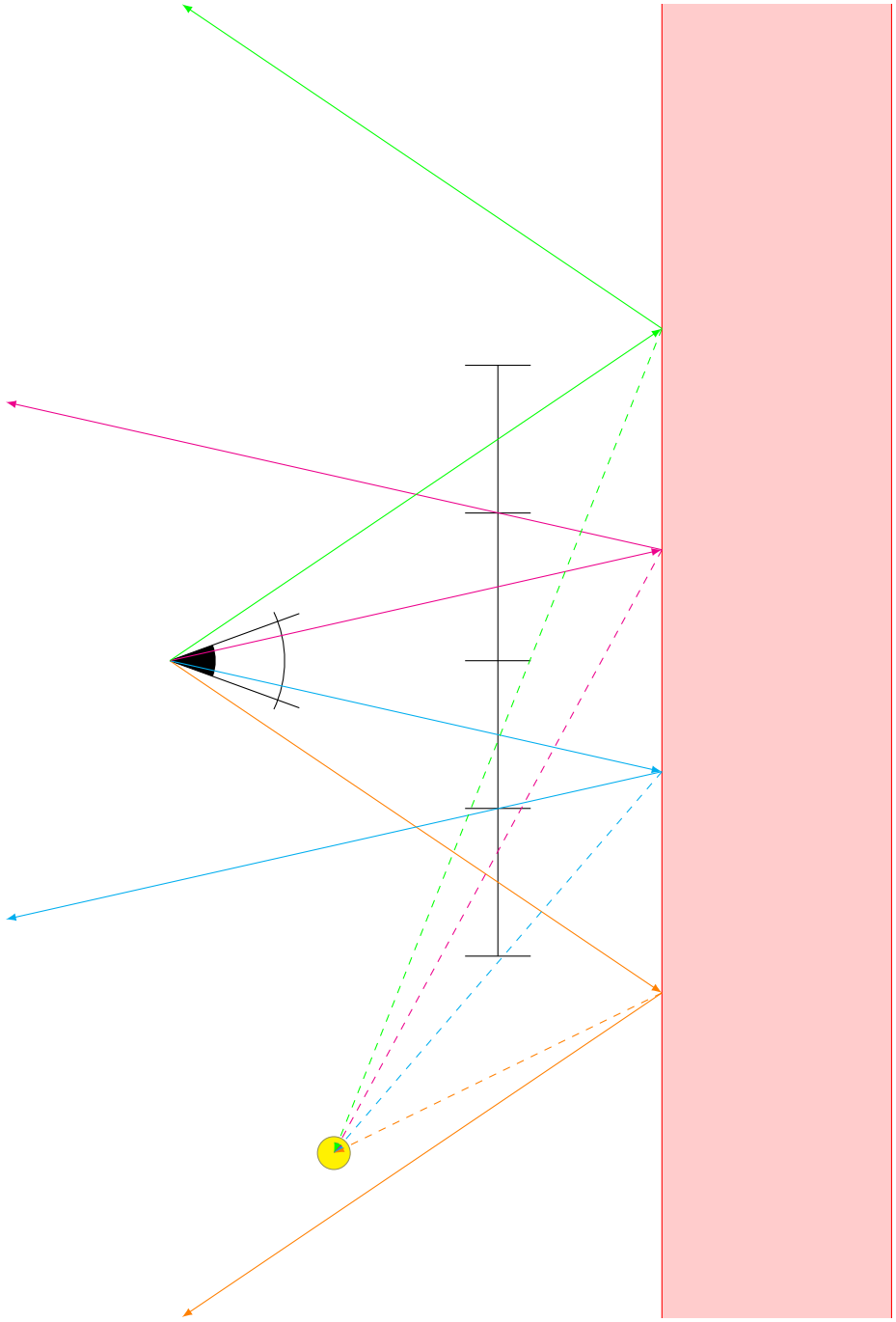
Problem 13 (4 points)

The object in the raytracing problem below is made of **wood**. The scene is in 2D with a 1D image. Each image has four pixels. yellow circles are point lights; the ray tracer supports shadows. Draw all of the rays that would be cast while raytracing this scene. You may assume a recursion depth of 4. (If you have recursion depth 3-5, it is fine.)



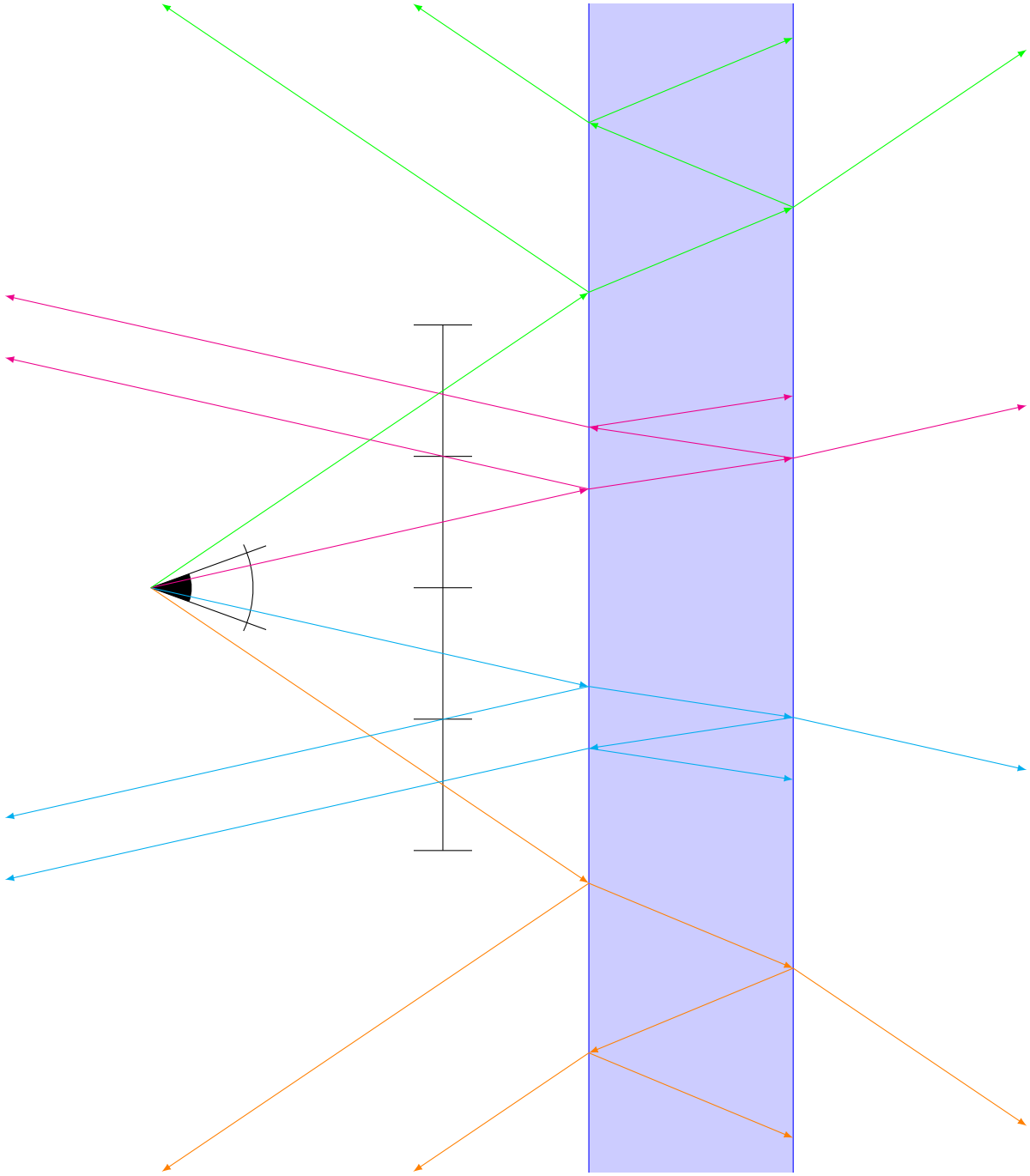
Problem 14 (4 points)

The object in the raytracing problem below is **reflective**. The scene is in scene is in 2D with a 1D image. Each image has four pixels. yellow circles are point lights; the ray tracer supports shadows. Draw all of the rays that would be cast while raytracing this scene. You may assume a recursion depth of 4. (If you have recursion depth 3-5, it is fine.)



Problem 15 (4 points)

The object in the raytracing problem below is **transparent**. The scene is in scene is in 2D with a 1D image. Each image has four pixels. yellow circles are point lights; the ray tracer supports shadows. Draw all of the rays that would be cast while raytracing this scene. You may assume a recursion depth of 4. (If you have recursion depth 3-5, it is fine.)



A degree-n Bezier curve with control points $P_0, \ldots, P_n$ has the form

$$p(t) = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} P_k \qquad \binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad k! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot k$$

Over the next few problems, we will look at algorithms for computing them. You implemented the De Casteljau's algorithm for computing them in the case for $n = 3$. Here, we will consider the general case, focusing on the (not very useful) case when n is large.

Problem 16 (3 points)

Implement the function vec2 Evaluate(const std::vector<vec2>& P, double t); for computing $p(t)$ using the De Casteljau's algorithm. You may allocate $O(n)$ additional space if you like, and your implementation should run in $O(n^2)$ time. You may use `vec2` routines. You should try to stay close to C++ syntax, but don't worry about minor syntax errors.

```
vec2 Evaluate(const std::vector<vec2>& P, double t)
{
    std::vector<vec2> A(P);
    int n=P.size()-1;
    for(int k=n; k>0; k--)
        for(int i=0; i<k; i++)
            A[i] = A[i] * (1-t) + A[i+1] * t;
    return A[0];
}
```

A degree-n Bezier curve with control points $P_0, \ldots, P_n$ has the form

$$p(t) = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} P_k \qquad \binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad k! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot k$$

Over the next few problems, we will look at algorithms for computing them. You implemented the De Casteljau's algorithm for computing them in the case for $n = 3$. Here, we will consider the general case, focusing on the (not very useful) case when n is large.

Problem 17 (2 points)

An alternative implementation would be to compute the blending functions

$$c_k = \binom{n}{k} t^k (1-t)^{n-k}$$

directly, then use these to evaluate the Bezier curve. Computing the blending functions directly takes $O(n)$ for each c_k, resulting in an $O(n^2)$ algorithm, which is no better. If we compute them all at once, however, we can improve on this. First, we break c_k into two pieces r_k and s_k.

$$c_k = r_k s_k \qquad r_k = \binom{n}{k} t^k \qquad s_k = (1-t)^{n-k}.$$

Next, we note that r_i can be easily computed left to right and s_i can be easily computed right to left. Let

$$A = r_0 \qquad R(k) = \frac{r_k}{r_{k-1}} \qquad B(n) = s_n \qquad S(k) = \frac{s_k}{s_{k+1}}$$

Derive expressions for A, $R(k)$, $B(n)$, and $S(k)$. Note that each of these expressions should be $O(1)$ and use only basic arithmetic (`+,-,*,/`).

$$A = r_0 = \binom{n}{0} t^0 = 1$$

$$R(k) = \frac{r_k}{r_{k-1}} = \frac{\binom{n}{k} t^k}{\binom{n}{k-1} t^{k-1}} = \frac{\frac{n!}{k!(n-k)!}}{\frac{n!}{(k-1)!(n-k+1)!}} t = \frac{n!(k-1)!(n-k+1)!}{n!k!(n-k)!} t = \frac{n-k+1}{k} t$$

$$B(n) = s_n = (1-t)^{n-n} = 1$$

$$S(k) = \frac{s_k}{s_{k+1}} = \frac{(1-t)^{n-k}}{(1-t)^{n-k-1}} = 1 - t$$

A degree-n Bezier curve with control points $P_0, \ldots, P_n$ has the form

$$p(t) = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} P_k \qquad \binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad k! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot k$$

Over the next few problems, we will look at algorithms for computing them. You implemented the De Casteljau's algorithm for computing them in the case for $n = 3$. Here, we will consider the general case, focusing on the (not very useful) case when n is large.

Problem 18 (3 points)

Implement the function vec2 Evaluate(const std::vector<vec2>& P, double t); for computing $p(t)$ in $O(n)$ time and $O(n)$ space. (You may use A, $R(k)$, $B(n)$, and $S(k)$ from the previous problem, even if you have not solved it. You do not need to implement them.) You may use `vec2` routines. You should try to stay close to C++ syntax, but don't worry about minor syntax errors.

```
vec2 Evaluate(const std::vector<vec2>& P, double t)
{
    std::vector<double> s(P.size());
    int n=P.size()-1;
    s[n]=B(n);
    for(int k=n-1; k>=0; k--)
        s[k]=s[k+1]*S(k);
    double r=A;
    vec2 X=s[0]*r*P[0];
    for(int k=1;k<=n;k++)
    {
        r*=R(k);
        X+=s[k]*r*P[k];
    }
    return X;
}
```

Problem 19 (2 points)

Clip the segment (in homogeneous coordinates) with endpoints $A = (-4, 1, 3, 5)$ and $B = (-2, 5, 3, 3)$.

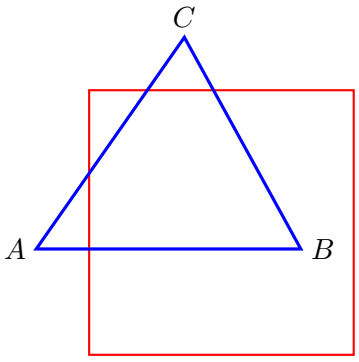
Note that both vertices satisfy all of the clipping constraints except $y \le w$, which B fails. Thus, we only need to clip against the plane $y = w$. Let

$$
\begin{aligned}
P &= (1-\gamma)A + \gamma B \\
0 &= P_y - P_w \\
&= ((1-\gamma)A_y + \gamma B_y) - ((1-\gamma)A_w + \gamma B_w) \\
&= (1 - \gamma + 5\gamma) - (5(1-\gamma) + 3\gamma) \\
&= -4 + 6\gamma \\
\gamma &= \frac{2}{3} \\
P &= \frac{1}{3}A + \frac{2}{3}B \\
P &= \begin{pmatrix} -\frac{8}{3} \\ \frac{11}{3} \\ 3 \\ \frac{11}{3} \end{pmatrix}
\end{aligned}
$$

Then, AP is the clipped segment.

Problem 20 (2 points)

Apply one of the clipping algorithms that we learned in class, step by step, to the triangle ABC. The clipping region is the red square. *Note that points are being awarded for demonstrating the steps of the algorithm. Merely showing what the results might look like does not score points.*



A correct solution for the triangle-based algorithm would clip the triangle against the edges of the square one by one. Triangles that get clipped into a quad should be divided back into two triangles. A correct solution for the polygon-based algorithm would maintain a polygon (initially the triangle) and clip it against the edges of the square one by one.