CS 184 Summer 2025
Exam 1

July 14th, 2025
Time Limit: 80 minutes

Name: Mr. Solutions

SID Number: _____

Room: _____

Instructions:

- Please wait to look past this page until official start time.
- You must carefully read and sign the Honor Code below.
- Initial the top of every page, in case pages separate while scanning.
- This exam is closed book, except for one 8.5×11 page of notes (double sided), handwritten.
- This exam is: 80 minutes long, 100 points, 14 pages (including this cover page) and 5 parts.
- Problem difficulty varies, so move on if feeling stuck on a problem.
- There is one coding subpart for extra credit. This will be worth 5 points.
- Answer each question in the space provided. Partial credit may be given on certain problems.
- Please detach the last page of scratch paper, which will not be graded.
- Raise your hand if you require any clarifications. If we decide to make a clarification, we will show it on the projector.

Question	Points
1	20
2	25
3	15
4	15
5	25
Total:	100

Honor Code and Academic Integrity Certification

The Honor Code is the commitment and work of students individually and as a community to uphold honesty and integrity. This commitment is comprehensive. Specifically on exams, it means not giving or receiving any help, or using any resources that are not permitted. The Honor Code requires that students take an active role in seeing to it that others as well as themselves uphold the letter and spirit of the Honor Code.

By signing I certify that:

- All the work submitted in my name for this exam is my work alone.
- I have not given or received any help during this exam.
- I have not used any resources during this exam aside from my own page of notes.
- I will inform course staff immediately about any Honor Code violations I become aware of.

I understand that a false statement on this declaration is a violation of the Honor Code, and would be subject to the highest level of disciplinary sanction under the University Code of Conduct.

Signature: _____

Date: _____

1. (Total : 20 points) True / False

   Mark each statement True or False. (1 point each)

   (1a) T ◯ F ◯ **[Soln: F]** Matrix multiplication is commutative in general.

   (1b) T ◯ F ◯ **[Soln: T]** You need exactly four linearly independent constraints to guarantee a fit to a unique cubic polynomial.

   (1c) T ◯ F ◯ **[Soln: F]** Bézier and Catmull-Rom curves interpolate through all of their control points.

   (1d) T ◯ F ◯ **[Soln: F]** The Nyquist frequency is the highest frequency that can be sampled without losing information.

   (1e) T ◯ F ◯ **[Soln: T]** Bounding Volume Hierarchies are not especially useful acceleration structures for rendering repetitive structures such as a grass field.

   (1f) T ◯ F ◯ **[Soln: F]** The Painters Algorithm does not require you to rasterize objects in any specific order.

   (1g) T ◯ F ◯ **[Soln: T]** Recording a video at 60 fps of a faucet dripping at a frequency of 40 Hz will result in temporal aliasing.

   (1h) T ◯ F ◯ **[Soln: T]** Gaussian convolution kernel is a form of low pass filtering over the frequency domain.

   (1i) T ◯ F ◯ **[Soln: T]** Jaggies, the wagon-wheel effect, and Moiré patterns are all artifacts of undersampling.

   (1j) T ◯ F ◯ **[Soln: F]** If you have not implemented mipmaps, you can antialias your textures with the same computational performance by increasing the supersampling rate.

   (1k) T ◯ F ◯ **[Soln: F]** If all three barycentric coordinates sum to 1, i.e. $\alpha + \beta + \gamma = 1$, then the point is guaranteed to be inside the triangle.

   (1l) T ◯ F ◯ **[Soln: F]** Mesh subdivision preserves mesh topology.

   (1m) T ◯ F ◯ **[Soln: T]** Increasing the p parameter in the Blinn-Phong model shrinks specular highlights.

   (1n) T ◯ F ◯ **[Soln: F]** Fragment shaders pass their output to vertex shaders as input as part of the graphics pipeline.

   (1o) T ◯ F ◯ **[Soln: F]** All affine transformations have an inverse.

   (1p) T ◯ F ◯ **[Soln: T]** In OpenGL, when a sample point falls on an edge, the sample is classified as within triangle if the edge is a "top edge" or "left edge".

   (1q) T ◯ F ◯ **[Soln: F]** In texture mapping, there always exists a 1:1 mapping between pixel samples and texel samples.

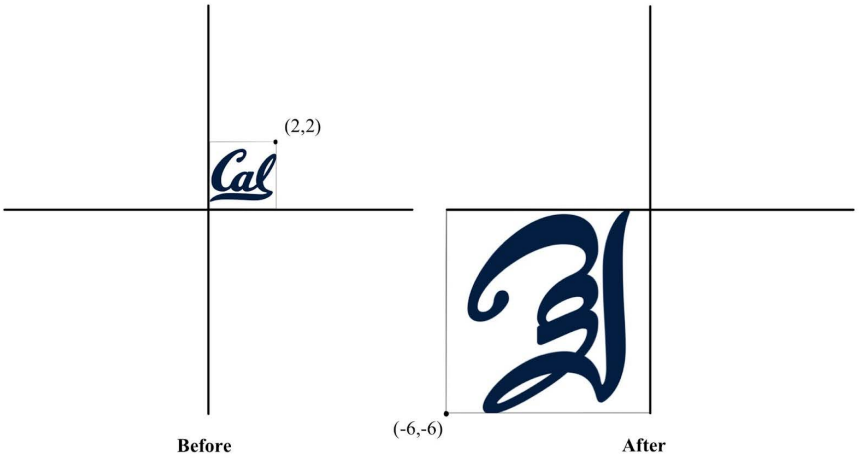
   (1r) T ◯ F ◯ **[Soln: T]** For a regular quad mesh, Catmull-Clark subdivision creates $2N \cdot 2N$ mesh from a $N \cdot N$ mesh, where N is the number of edges.

   (1s) T ◯ F ◯ **[Soln: T]** When computing the appropriate mipmap level for texture mapping, a larger L value generally corresponds to a lower resolution level.

   (1t) T ◯ F ◯ **[Soln: T]** Convolution in the spatial domain is equivalent to multiplication in the frequency domain, and vice versa.

2. (Total : 25 points) Transforms

(2a) [12 points] Simple Transformations



Following the conventions in lecture, which of the following 2D transforms will produce the result depicted in the image above. [Scratch graph-paper is attached at the back of the exam. Regardless of whether you use it, please detach that page from the exam.] Mark each option as True or False (3 points each).

2a.i. T ◯ F ◯ **[Soln: F]** Scale(3, 3) · Rotate(180°)

2a.ii. T ◯ F ◯ **[Soln: T/F]** ReflectVertical · Rotate(270°) · Scale(3, 3)

We gave credit for True or False depending on the direction you applied the transformations (left to right as in text or right to left as a matrix)

2a.iii. T ◯ F ◯ **[Soln: F]** $\begin{bmatrix} -3 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

2a.iv. T ◯ F ◯ **[Soln: T]** $\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

(2b) Camera to World to Camera

Consider a camera in a 2D world located at the point (2, 1), looking at a 2D ant located at the point (3, 2).

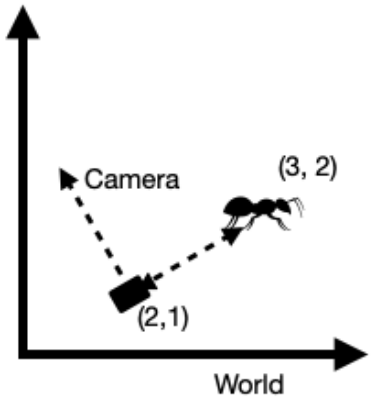
2b.i. [3 points] Draw and label the world and camera reference frames. Label the viewing direction vector, the ant, and the up vector.

2b.ii. [7 points] In homogeneous coordinates, compute the transform matrix from standard camera to world space. We will use a right-handed coordinate system.

2b.iii. [3 points] Given a coordinate system in camera space, what step(s) would you apply to obtain the corresponding coordinate in world space? Describe the steps, don't do any manual computation.

Solution:

- 2b.i



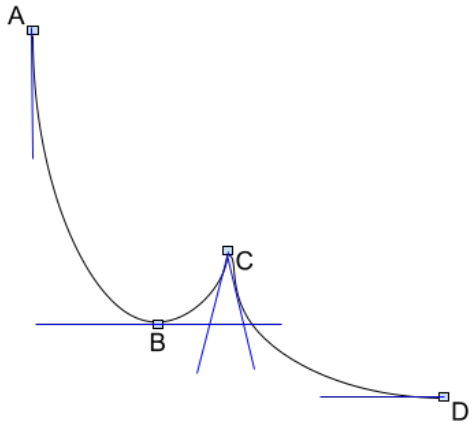
- 2b.ii $\begin{bmatrix} \frac{\sqrt{1}}{2} & -\frac{\sqrt{1}}{2} & 2 \\ \frac{\sqrt{1}}{2} & \frac{\sqrt{1}}{2} & 1 \\ 0 & 0 & 1 \end{bmatrix}$

- 2b.iii Multiply the coordinates by the above transform matrix

3. (Total : 15 points) Geometry

(3a) [3 points] Cubic Bezier Splines

The following diagram shows a spline composed of three cubic Bezier curves. The spline is C^1 continuous at point B. Draw the missing control points for the three cubic Bezier curves at their approximate locations.



Solution:

In blue

(3b) [6 points] Cubic Hermite Interpolation

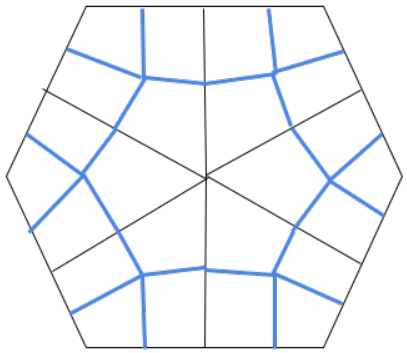
Assume you have two points at $t = 0$, $t = 1$ with $P(0) = 5$, $P'(0) = 3$, $P(1) = -2$, $P'(1) = 3$. Use Cubic Hermite Interpolation to derive the formula for a cubic polynomial function $f(t)$ that interpolates as t varies from 0 to 1. Show your work.

Solution:

$$f(t) = 20t^3 - 30t^2 + 3t + 5$$

(3c) Catmull-Clark Subdivision

3c.i. [3 points] Let's consider applying Catmull-Clark Subdivision to the mesh below. Draw one level of subdivision, clearly showing the new vertices and edges on the illustration below.



Solution: Above in blue

3c.ii. [3 points] How many faces, F_N will there be after 3 rounds of subdivision.

Solution: 384

4. (Total : 15 points) [Coding] de Casteljau

(4a) [15 points] In 3D space, Bèzier curves can be evaluated using de Casteljau's algorithm. Write a C++ function that implements de Casteljau's algorithm for Bézier curves in 3D using arrays of length 3 to represent 3D points.
Note the following.

- $0 <$ `delta_t` < 1, which represents the sampling interval.
- Return a `std::vector<Vector3D>` containing the sampled points on the Bèzier curve at uniform intervals of `delta_t`, including both $t = 0$ and $t = 1$.
- We only require you to fill in `de_casteljau_curve()`, but feel free to organize your code in `lerp()` and `de_casteljau_single()`

```
Vector3D lerp(const float a[3], const float b[3], float t) {
    Vector3D x(a[0], a[1], a[2]);
    Vector3D y(b[0], b[1], b[2]);
    return x * (1 - t) + y * t;
}

Vector3D de_casteljau_single(
    const float control_points[][3],
    int num_points,
    float t
) {
    float new_points[num_points][3];
    while (num_points > 1) {
        for (int i = 0; i < num_points - 1; i++) {
            Vector3D temp = lerp(new_points[i], new_points[i + 1], t);
            new_points[i][0] = temp.x;
            new_points[i][1] = temp.y;
            new_points[i][2] = temp.z;
        }
        num_points--;
    }

    return Vector3D(new_points[0][0], new_points[0][1], new_points[0][2]);
}
```

```
vector<Vector3D> de_casteljau_curve(
    const float control_points[][3],
    int num_points,
    float delta_t
) {

    vector<Vector3D> result;
    float i = 0;
    for (; i < 1; i+=delta_t)
        result.push_back(de_casteljau_single(control_points, num_points, i));

    result.push_back(de_casteljau_single(control_points, num_points, 1.0f));

    return result;
}}


// Example Usage
float control_points[][3] = {
    {0.0f, 0.0f, 0.0f},
    {1.0f, 2.0f, 0.0f},
    {2.0f, 0.0f, 0.0f}
};
int num_points = 3;
float delta_t = 0.1f;

auto curve_points = de_casteljau_curve(control_points, num_points, delta_t);
```

5. (Total : 25 points) Z-Buffer with Opacity

(5a) You are asked to modify the traditional Z-buffer algorithm to support opacity-aware compositing of semi-transparent triangles in a 3D scene rendered in camera space.

Each triangle has a 2D screen-space representation (simplified for this problem), a color, depth value, and an opacity (between 0–100). Instead of storing only the closest triangle, the Z-buffer should now retain a list of triangles sorted by depth per pixel, but only up to a total opacity of 100. Once the sum of opacities reaches or exceeds 100, any remaining triangles must be ignored for that pixel.

You will implement:

- A blend function that blends the final pixel color using weighted color contributions up to 100 opacity.
- A modified Z-buffer algorithm that stores only necessary triangles per pixel.

A simple driver that processes a framebuffer using these utilities can be completed for **Extra Credit**.

```
// Definitions
struct Triangle {
    Vector3D vertices[3];    // In camera space. All z-values will be the same
    Vector3D color;          // RGB values
    float opacity;           // In range [0, 100]
};
```

5a.i. [10 points] Implement the `blend` function. Each of the `fragments` is a triangle with color and opacity. This list is sorted front-to-back in terms of visibility, although the sum of opacities may exceed 100.

The function should compute final_color = color1 · opacity1 + color2 · opacity2 + ... If adding a fragment's full opacity would exceed 100, truncate it so the total is capped. For example, if we have two triangles with opacity 75, the resulting equation would be final_color = color1 · 75 + color2 · 25.

Ideally, opacity should sum to 1, but we overlooked this when writing the exam

```
Vector3D blend(const std::vector<Triangle>& fragments) {
    Vector3D blended;
    Vector3D remaining_opacity = 100.0f;

    for (Triangle t : fragments) {
        if (remaining_opacity <= 0.0f) {
            break;
        }
        blended += std::min(t.opacity, remaining_opacity) * t.color;
        remaining_opacity -= t.opacity;
    }

    return blended;
}
```

5a.ii. [15 points] Implement the `renderToZBuffer` algorithm. the Z-buffer should be a 2D grid of pixels, each pixel holding a sorted list of Triangles from closest to furthest to the camera. Use the provided `insertOrdered` function to insert a triangle into a pixel's list, however, **you must prune the list if the cumulative opacity exceeds 100.** excluding the final triangle that puts the list over the threshold. Do not blend colors in this function.

```
// Implementation Hidden
void insertOrdered(std::vector<Triangle> pixelBuffer, Triangle triangle);
// Remove last element of a vector
void std::vector::pop_back()

void renderToZBuffer(
    const std::vector<Triangle>& triangles,
    int screenWidth, int screenHeight,
    std::vector<std::vector<Triangle>>& zBuffer
) {
    for (int x = 0; x < screenWidth; x++) {
        for (int y = 0; y < screenHeight; y++) {
            Vector2D p(x + 0.5f, y + 0.5f);
            for (auto &t : triangles) {
                // We did not grade for winding order
                Vector2D p0(t.vertices[0].x, t.vertices[0].y);
                Vector2D p1(t.vertices[1].x, t.vertices[1].y);
                Vector2D p2(t.vertices[2].x, t.vertices[2].y);
                Vector2D n0_1(p0.y - p1.y, p1.x - p0.x);
                Vector2D n1_2(p1.y - p2.y, p2.x - p1.x);
                Vector2D n2_0(p2.y - p0.y, p0.x - p2.x);

                // Inside the triangle test
                if ((dot(p - p0, n0_1) >= 0)
                    && (dot(p - p1, n1_2) >= 0)
                    && (dot(p - p2, n2_0) >= 0)) {
                    insert_ordered(zBuffer[y * width + x], t);
                }
            }

            float total_opac = 0;
            int total_triangles = zBuffer[y * width + x].size();
            for (int i = 0; i < total_triangles; i++) {
                if (total_opac > 100) {
                    for (int j = total_triangles - i; j > 0; j--) {
                        zBuffer[y * width + x].pop_back();
                    }
                    break;
                }
                total_opac += zBuffer[y * width + x].opacity;
            }
        }
    }
}
```

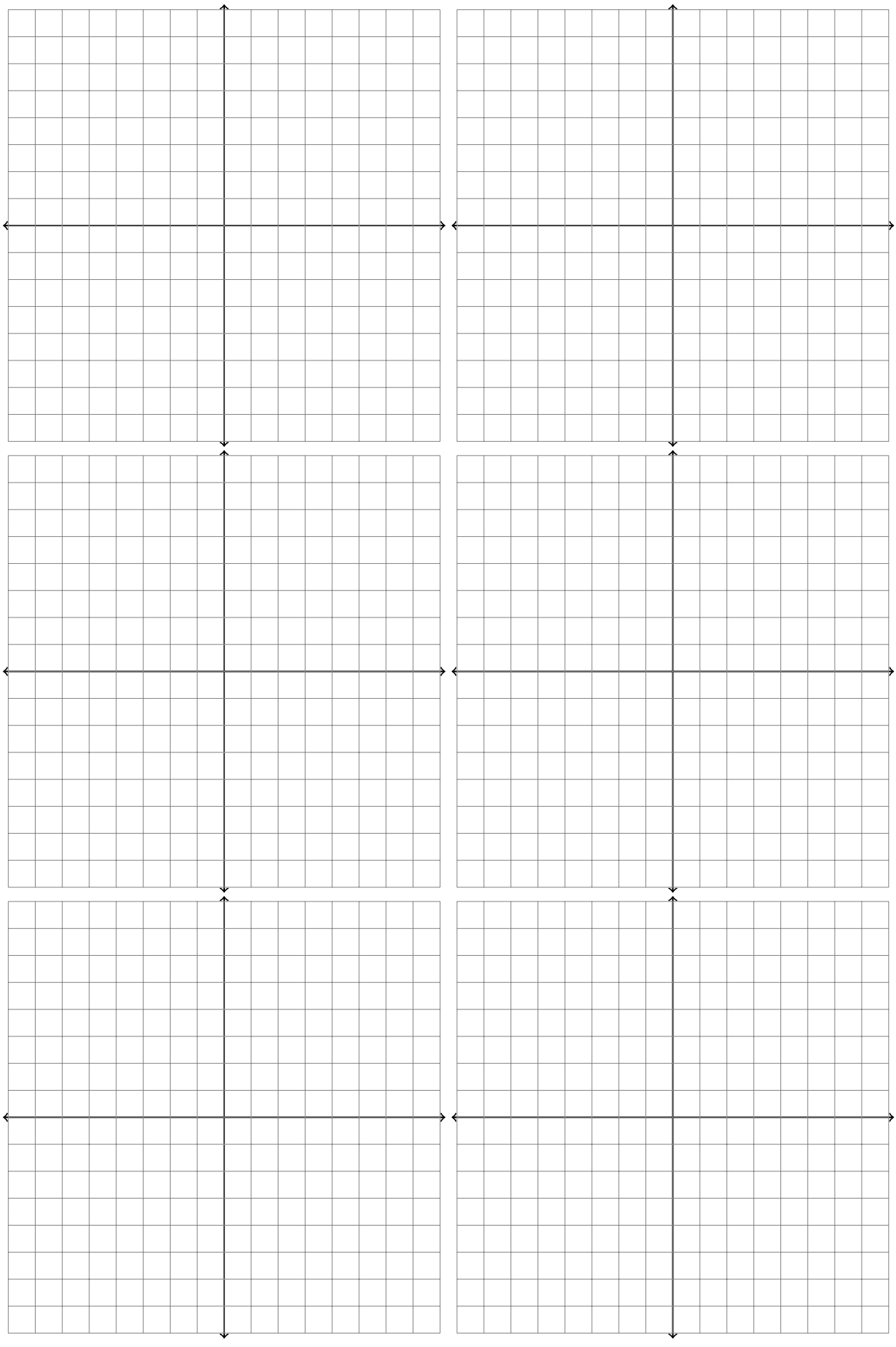
Congratulations for making it to the end of the exam! This extra credit question is optional.

Extra Credit (5 Points): Put together the `blend` and `renderToZBuffer` functions!

```
int rasterize_triangles(
    std::vector<Triangle>& sceneTriangles,
    std::vector<Vector3D>& framebuffer, // size = width * height
    int width,
    int height
) {
    std::vector<std::vector<Triangle>> zBuffer(
        width * height
    );

    renderToZBuffer(sceneTriangles, width, height, zBuffer);

    for (int i = 0; i < width * height; i++) {
        framebuffer[i] = blend(zBuffer[i]);
    }

}
```

Assumptions Box

If there's anything else you want us to know, or you feel like there was an ambiguity in the exam, please put it in the box below. For ambiguities, you must qualify your answer and provide an answer for both interpretations. You will only receive credit if it is a genuine ambiguity and both of your answers are correct. We will only look at ambiguities if you request a regrade.

For example, "If question X is asking about A, then my answer is Y, but if the question is asking about B, then my answer is Z".

Blank page – this is scratch space for you.