CS184-2025 Spring-Homework1

Shuang Liu

Overview

I followed all the instructions for Homework 1 and successfully completed all the basic assignments. I implemented the required functions step by step, making modifications to some earlier functions as needed while working on later tasks.

In summary, I implemented triangle rasterization, supersampling, image transformations, barycentric coordinate computation, texture sampling using nearest-neighbor and bilinear interpolation, and mipmap level selection.

Through this process, I learned how to apply existing knowledge and use the provided code framework to complete the missing parts. I also improved my debugging skills by analyzing visual outputs. Ultimately, I gained a deep understanding of how a rasterizer works.

Task 1

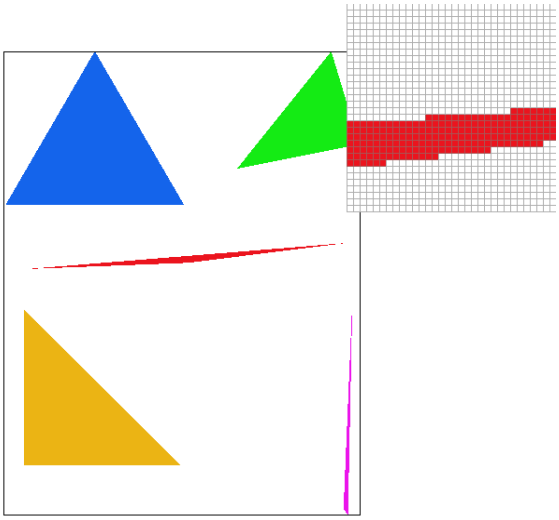
Q1: Walk through how you rasterize triangles in your own words.

A1: At first, I sort the vertices (x0, y0), (x1, y1), (x2, y2) to ensure that y0 <= y1 <= y2. Then, I compute the slopes of the edges to determine how x changes as y increases. Starting from the lowest point (x0, y0), I iterate over each scanline by incrementing y by 1 and calculating the corresponding x range using linear interpolation. For each pixel within this range, I use barycentric coordinates to check whether it lies inside the triangle. Finally, I call the rasterize_point function to color the valid pixels.

Q2: Explain how your algorithm is no worse than one that checks each sample within the bounding box of the triangle. The bounding box of the triangle is defined as the smallest rectangle that can be drawn whilst ensuring that the entire triangle is within it.

A2: My algorithm avoids checking every point within the bounding box. Instead, it only evaluates pixels within the valid x-range for each scanline. By incrementing y and computing the corresponding x-range using edge slopes, the algorithm efficiently skips unnecessary checks. Thus, this approach is significantly more efficient than iterating over every pixel in the bounding box.

Q3: Show a png screenshot of basic/test4.svg with the default viewing parameters and with the pixel inspector centered on an interesting part of the scene.

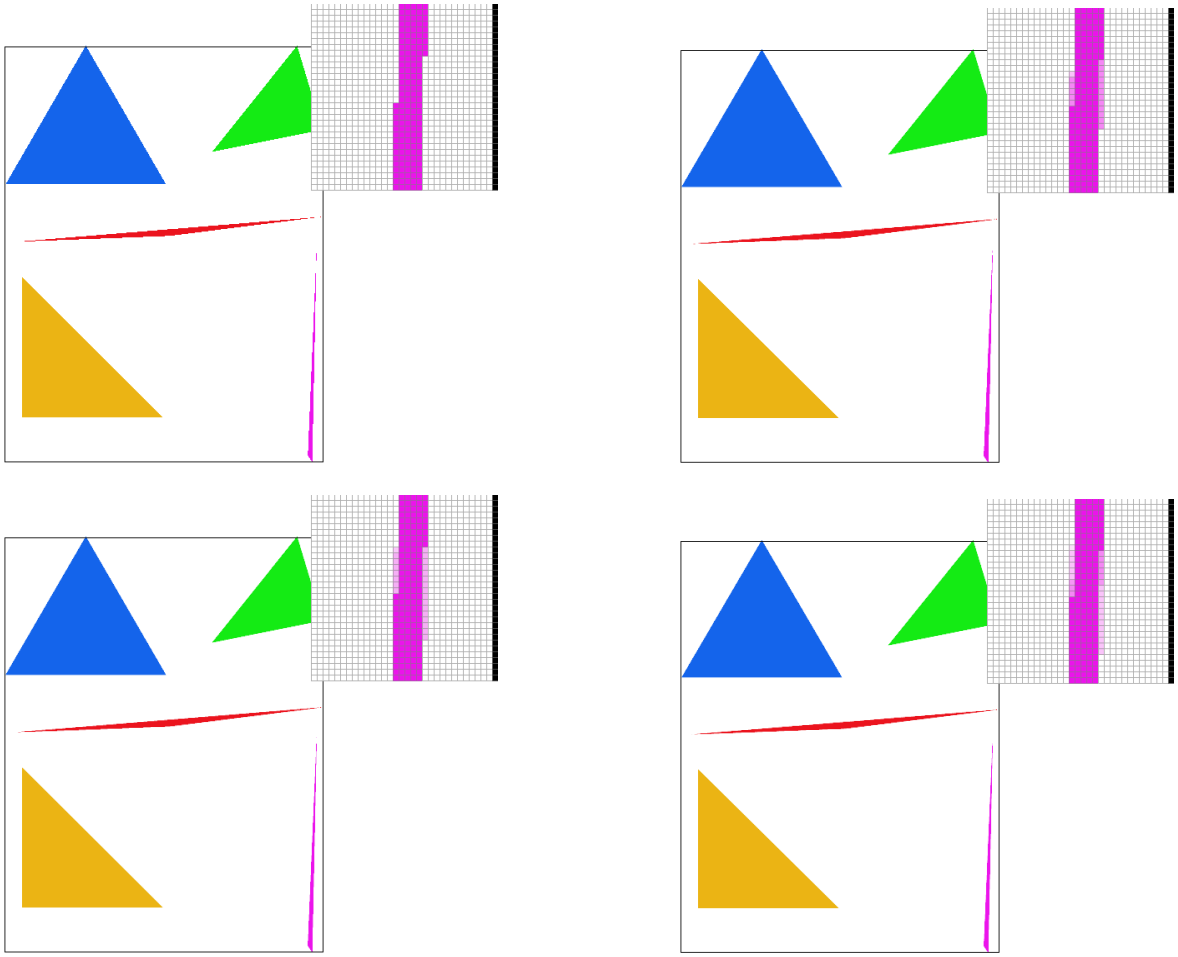


Task 2

Q1: Walk through your supersampling algorithm and data structures. Why is supersampling useful? What modifications did you make to the rasterization pipeline in the process? Explain how you used supersampling to antialias your triangles.

A1: Supersampling is useful because relying solely on (x+0.5f, y+0.5f) to determine pixel coverage causes abrupt transitions along triangle edges, leading to aliasing. By dividing each pixel into smaller sub-pixels (e.g., 4 or 16), we can sample multiple points within a pixel and average the results to produce smoother transitions. This reduces jagged edges and improves visual quality. To implement supersampling, I calculate sub-pixel positions (sub_x, sub_y) and store color samples in a sample_buffer. After rasterization, the final pixel color is obtained by averaging the sampled values, effectively applying antialiasing.

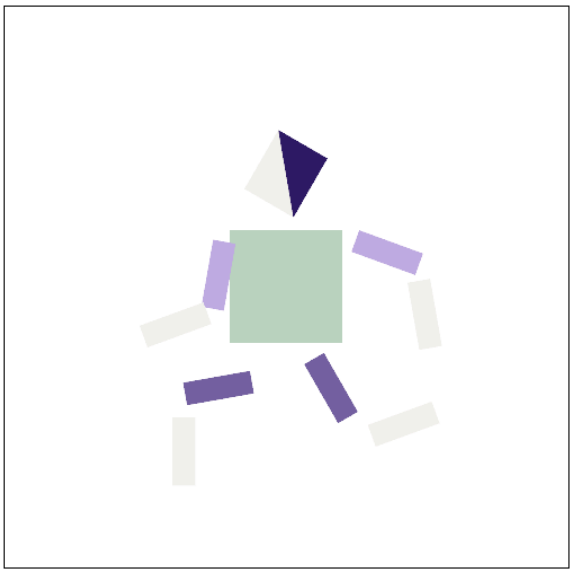
Q2: Show png screenshots of basic/test4.svg with the default viewing parameters and sample rates 1, 4, and 16 to compare them side-by-side. Position the pixel inspector over an area that showcases the effect dramatically; for example, a very skinny triangle corner. Explain why these results are observed.



A2: The observed results occur because the chosen triangle is very thin, causing y to span multiple pixels as x increases, leading to aliasing. Supersampling improves accuracy by taking multiple samples per pixel, allowing for a more precise determination of whether a point is inside the triangle, which helps reduce aliasing effects.

Task 3

Q1: Create an updated version of svg/transforms/robot.svg with cubeman doing something more interesting, like waving or running. Feel free to change his colors or proportions to suit your creativity. Save your svg file as my_robot.svg in your docs/ directory and show a png screenshot of your rendered drawing in your write-up. Explain what you were trying to do with cubeman in words.

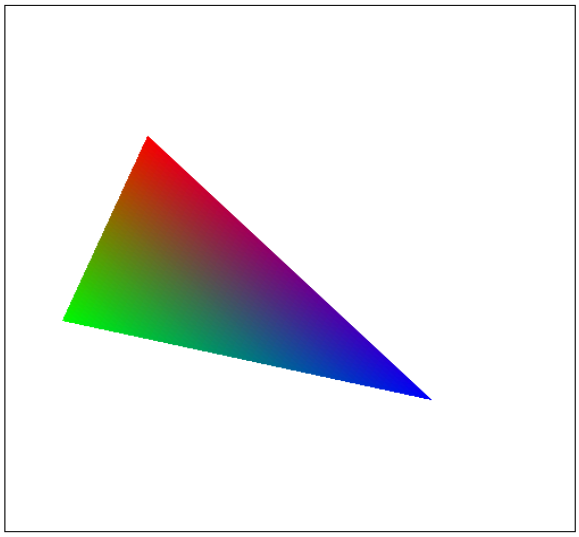


A1: I created a running cubeman inspired by the colors of the Paris Olympics, reflecting a sense of art and beauty—though the cubeman itself remains somewhat simple and crude. To achieve this, I rotated and scaled its head, as well as transformed and translated its arms and legs to create a dynamic running pose.

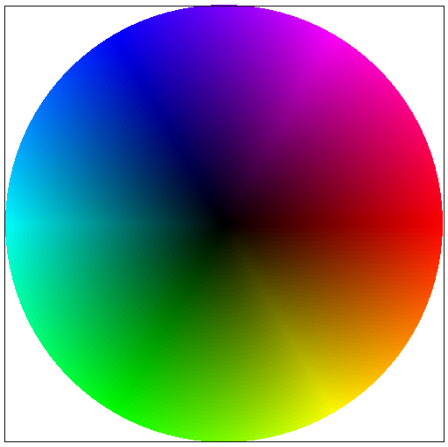
Task 4

Q1: Explain barycentric coordinates in your own words and use an image to aid you in your explanation. One idea is to use a svg file that plots a single triangle with one red, one green, and one blue vertex, which should produce a smoothly blended color triangle.

A1: Barycentric coordinates use three weights—α (alpha), β (beta), and γ (gamma)—to express the position of a point relative to the vertices of a triangle. These weights represent the proportion of the area formed by the point and two triangle vertices compared to the total area of the triangle. This provides a natural way to interpolate values, such as color, across the triangle.



Q2: Show a png screenshot of svg/basic/test7.svg with default viewing parameters and sample rate 1. If you make any additional images with color gradients, include them.



Task 5

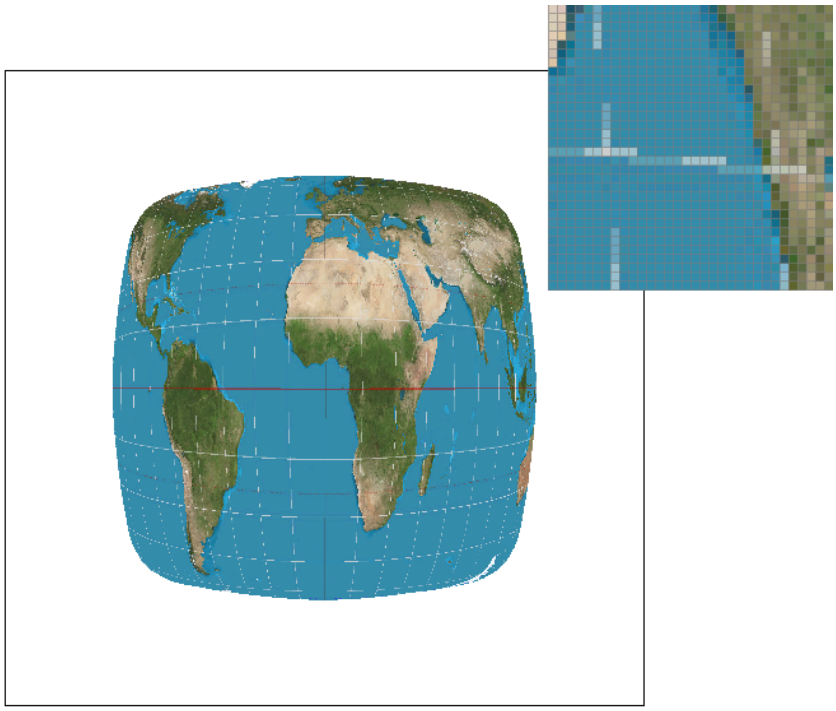
Q1: Explain pixel sampling in your own words and describe how you implemented it to perform texture mapping. Briefly discuss the two different pixel sampling methods, nearest and bilinear.

A1: Pixel sampling is the process of determining the color of a pixel by referencing a texture image. In texture mapping, both the rendered image and the texture image consist of pixels, and our task is to establish a correspondence between them. This is achieved by barycentric coordinates (α, β, γ), which

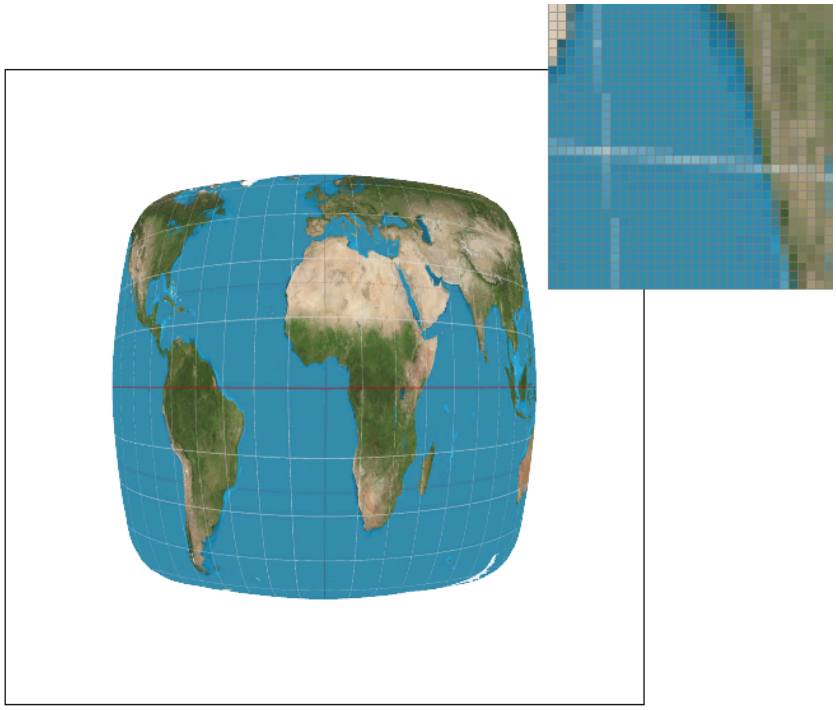
is used to get the texture coordinates (u, v) for a given pixel. Once we obtain the texture coordinates, we sample the texture image to determine the color to be applied to the rendered pixel.
Nearest neighbor sampling selects the color of the closest texel to (u, v) and then uses it directly. Bilinear interpolation considers the four closest texels surrounding (u, v), and then computes a weighted average of these four texels based on their relative distances to the sampled point. The final color is a smooth blend of the four texels, leading to softer transitions and reducing aliasing effects.

Q2: Check out the svg files in the svg/texmap/ directory. Use the pixel inspector to find a good example of where bilinear sampling clearly defeats nearest sampling. Show and compare four png screenshots using nearest sampling at 1 sample per pixel, nearest sampling at 16 samples per pixel, bilinear sampling at 1 sample per pixel, and bilinear sampling at 16 samples per pixel.

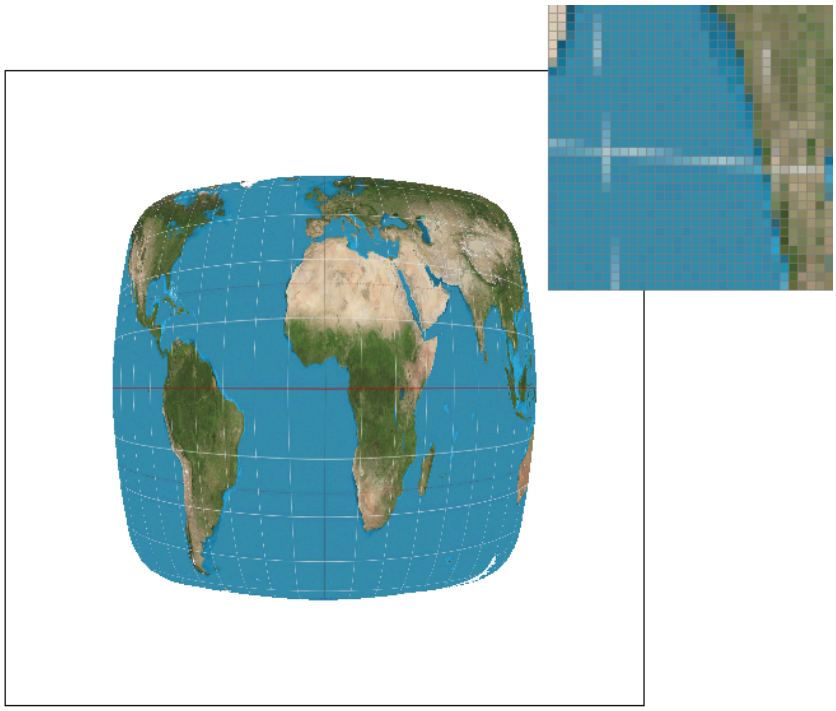
A2:
nearest sampling at 1 sample per pixel



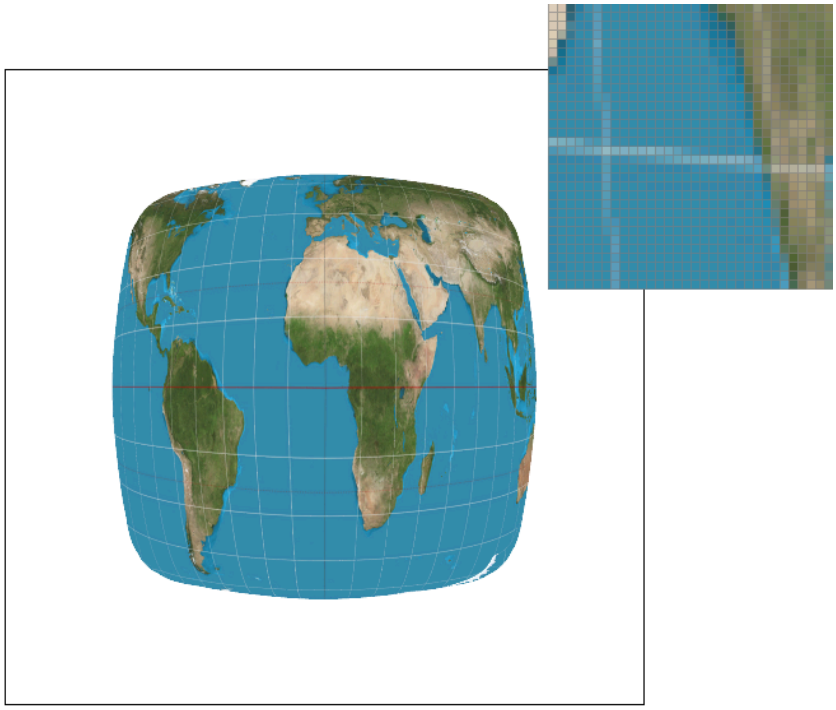
nearest sampling at 16 sample per pixel



bilinear sampling at 1 sample per pixel



bilinear sampling at 16 sample per pixel



Q3: Comment on the relative differences. Discuss when there will be a large difference between the two methods and why.

A3: Nearest neighbor sampling is the simpler of the two methods. While this method is computationally efficient, it often results in aliasing artifacts, causing the texture to appear blocky or pixelated. Bilinear interpolation, on the other hand, improves visual quality. However, this method requires more computation than nearest neighbor sampling, which may slightly impact performance.

Task 6

Q1: Explain level sampling in your own words and describe how you implemented it for texture mapping.

A1: Level sampling is used to determine the appropriate mipmap level for texture mapping based on the rate of change of texture coordinates in screen space. To achieve this, we compute not only the texture coordinates p_uv at a given pixel but also the differentials pdx_uv and pdy_uv, which represent how the texture coordinates change when moving one pixel in the x or y direction. From these, we calculate Dx and Dy, which measure the rate of change of texture coordinates in terms of pixel units. A larger D indicates that the texture is being magnified in screen space and needs to be sampled from a lower-resolution mipmap level to avoid aliasing. Mipmap levels are defined such that level 0

corresponds to the original texture resolution, level 1 corresponds to a version of the texture with half the resolution, and so on. Higher mipmap levels represent progressively lower-resolution textures, which help reduce aliasing artifacts and improve rendering performance, especially for distant objects. In summary, level sampling is used to adjust the texture resolution dynamically, ensuring that appropriate detail is preserved while mitigating rendering artifacts.

Q2: You can now adjust your sampling technique by selecting pixel sampling, level sampling, or the number of samples per pixel. Describe the tradeoffs between speed, memory usage, and antialiasing power between the three various techniques.

A2:

	Number of samples per pixel	Pixel sampling	Level samping
Speed	Increases rendering time significantly, as multiple samples must be computed and averaged.	Nearest-neighbor is fast, as it simply picks the closest texel. Bilinear sampling is slightly slower, as it requires interpolating between four texels.	L_ZERO is fast. L_NEAREST selects the closest mipmap level, slightly increasing computation time. L_LINEAR interpolates between two mipmap levels, increasing computation time more.
Memory usage	Does not require additional texture memory but increases framebuffer memory usage if storing multiple samples per pixel.	No extra memory is required beyond the texture itself.	Mipmap storage requires more memory to store texture image of different mipmap levels.
Antialiasing power	More samples lead to significantly better antialiasing, especially on the border.	Nearest-neighbor can cause aliasing artifacts. Bilinear sampling smooths transitions.	Helps reduce aliasing, especially for textures at oblique angles or small screen sizes or images with different depth.

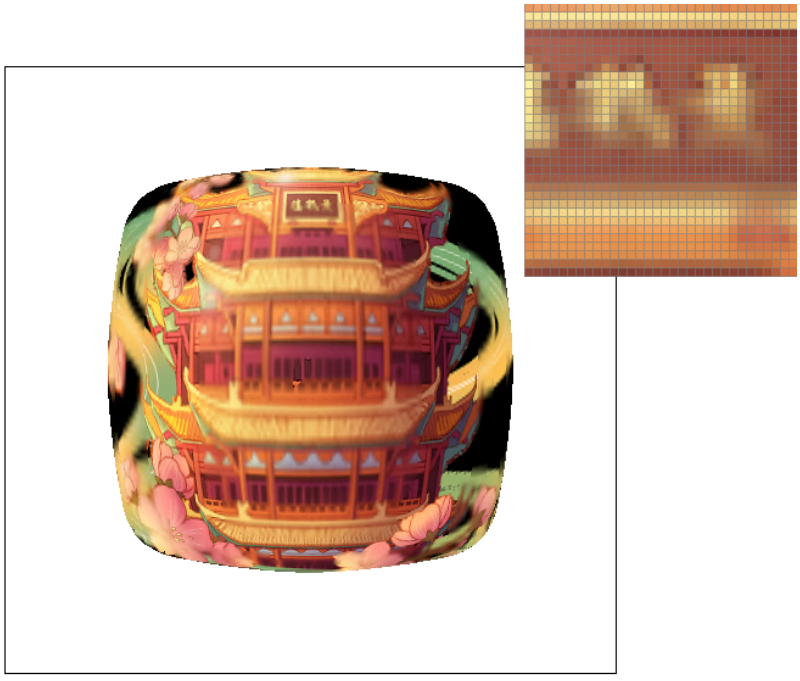
Q3: Using a png file you find yourself, show us four versions of the image, using the combinations of L_ZERO and P_NEAREST, L_ZERO and P_LINEAR, L_NEAREST and P_NEAREST, as well as L_NEAREST and P_LINEAR.

L_ZERO and P_NEAREST:

L_ZERO and P_LINEAR:

L_NEAREST and P_NEAREST

L_NEAREST and P_LINEAR



Report of use of AI

I use ChatGPT and Copilot to help debug my code, as they can catch mistakes that I might overlook. However, AI doesn’t always find all the bugs, so I often correct some issues myself. I also use AI to refine my reports and improve the clarity of my writing.