Carnegie Mellon University Africa

Name: Martin Koome

Andrew ID: mkoome

Course: Applied Computer Vision

Assignment 1

Image Formation

Part I: Pinhole Camera Model (50 Points)

Step 1

Let $C = (0,0,0)$ and $f = 50$.

Standard plane $Z = Z_0$

$$Z = 50$$

A general plane

$$ax + by + cz + d = 0$$

Given:

$$a = 0, \quad b = 0, \quad c = 1, \quad d = -f$$

Substituting into the equation:

$$0x + 0y + z - 50 = 0$$
$$z = 50$$

Example:

Consider a plane $f = 50$ and $f = 100$. Let $P = (10,5,70)$ be a point in 3D.
For $f = Z_0 = 50$ (standard plane):

$$I_x = f\frac{x}{Z}, \quad I_y = f\frac{y}{Z}$$

Substituting values:

$$I_x = 50 \cdot \frac{10}{70} = \frac{500}{70} = \frac{50}{7}$$

$$I_y = 50 \cdot \frac{5}{70} = \frac{250}{70} = \frac{25}{7}$$

Image Coordinates:

$$(I_x, I_y) = \left(\frac{50}{7}, \frac{25}{7}\right)$$

For $f = 100$ standard plane

$$I_x = \frac{100}{70}(10) \qquad I_y = \frac{100}{70}(5)$$
$$(I_x, I_y) = \left(\frac{100}{7}, \frac{50}{7}\right)$$

We note that for a fixed object position, a larger focal length results in a magnified image, whereas objects farther from the camera appear smaller due to the inverse dependence on depth.

For a rotated image plane

Tilting on the x-axis: $a = 0.3 \quad b = 0 \quad c = 1 \quad f = 50$

$$0.3x + 0y + z - 50 = 0$$
$$0.3x + z = 50$$

Tilting on both the x and y axis:

$$a = 0.2 \quad b = 0.1 \quad c = 1 \quad f = 100$$

$$0.2x + 0.1y + z - 100 = 0$$

Step 2

Given a 3D point P and camera center C

$$P = (x, y, z)$$

The ray originating from camera center C and passing through a 3D point P can be written:

$$R(t) = C + t(P - C)$$

Where $P - C = \vec{d}$ is the direction of the ray.

For $C = (0,0,0)$, the intersection between the ray and image plane is:

$$R(t) = t(x, y, z)$$
$$x(t) = tx, \quad y(t) = ty, \quad z(t) = tz$$

Step 3

For a standard image plane $Z = Z_0$

$$R(t) = t(x, y, z)$$
$$z(t) = tz = Z_0 \implies t = \frac{Z_0}{z}$$
$$\text{Intersection point} = \left(\frac{Z_0}{z}x, \frac{Z_0}{z}y, Z_0\right)$$

For a tilted image plane $ax + by + cz + d = 0$

$$a(tx) + b(ty) + c(tz) + d = 0$$
$$t(ax + by + cz) = -d$$
$$t = \frac{-d}{ax + by + cz}$$
$$\text{Intersection point} = t(x, y, z)$$

Step 4

Principal point $= (C_x, C_y)$

Pixel scaling factors S_x, S_y

$$U = S_x(I_x - O_x) + C_x$$
$$V = S_y(I_y - O_y) + C_y$$

Assuming $O_x = O_y = 0$ and $S_x = S_y = 1$:

$$U = I_x + C_x$$
$$V = I_y + C_y$$

A 3D point and a standard plane

Given:

$$D = (5, 2, 10), \quad f = 50, \quad \text{Image size} = (644, 424)$$

The principal point :

$$C_x = \frac{644}{2} = 322$$
$$C_y = \frac{424}{2} = 212$$

Projection I:

$$I = \left[\frac{50}{10} \cdot 5, \quad \frac{50}{10} \cdot 2, \quad 1\right]$$
$$I = (25, 10, 1)$$

Pixel coordinates:

$$U = (25 + 322) = 347$$
$$V = (10 + 212) = 222$$

Pixel location:

$$(U, V) = (347, 222)$$

For step 3 and 4, I sampled the first 200 points of the black points provided and demonstrated how to find the intersection point between the image plane and the ray in both a standard plane and a tilted plane. Using an image size of 644 x 424, I calculated the pixel coordinates of the projected images (u,v)

Experimentation Tasks

This section describes the experimental analysis conducted using the provided blue and black point pairs, which were projected onto multiple image planes to study the effects of camera geometry. Figure 1 illustrates how varying the focal length influences the projected configuration of points on

a standard image plane perpendicular to the optical axis. In contrast, Figure 2 presents the results obtained when projecting the same points onto an image plane tilted along both the x- and y-axes, highlighting the additional geometric distortion introduced by plane orientation.

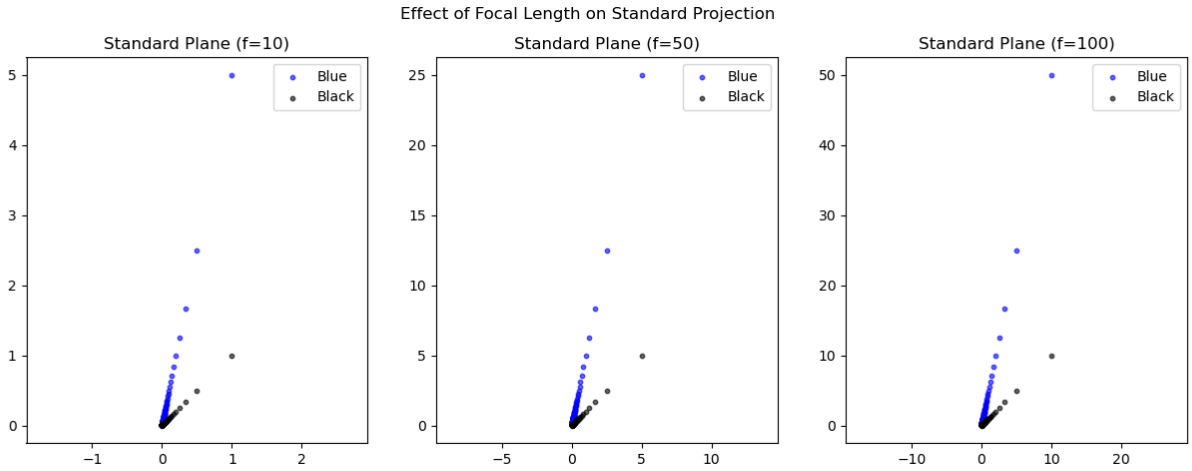

Figure 1: Standard Plane at different focal lengths

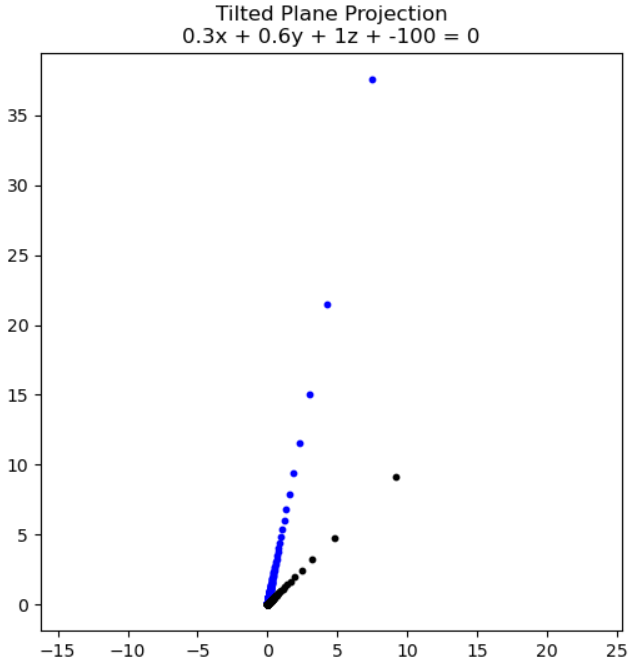

Figure 2: Tilted Plane

Analysis of parameters

i) I notice that each increment in the focal length increases the coordinates of the projected image. This is mainly because of f being a multiplicative scalar in the perspective projection $I_{x,y} = \left(\frac{f}{Z}x, \frac{f}{Z}y\right)$; therefore, an increase in f results in a corresponding increase in the image coordinates.

ii) In an image plane $0.3x + 0.6y + z - 100 = 0$, the arrangement of points is geometrically different from the standard $Z = 100$. The points are stretched significantly in the y-direction compared to the x-direction.

Position d or Z_0: Moving the plane further from the camera makes the image larger because the rays travel further apart before hitting the screen.

Orientation (a,b,c): Tilting the plane introduces perspective distortion. I see that parts of the plane closer to the camera intercept the rays early, making the points appear closer together, while parts tilted away intercept the rays late, making the points spread further apart.

iii) **Comparison**

	Standard plane $Z = Z_0$	**Tilted plane**
Symmetry	Preserves symmetry of the projection around the optical axis	Projection is skewed and distorted
Scaling	The image is scaled uniformly by f/Z	Scale varies across the image because distance from camera to the plane is different for every pixel.

Ideal Points Analysis

Table 1: Comparison of Euclidean Distances Before and After Projection

Projection Type	**Point Pair**	**3D Distance**	**Projected Distance**	**Ratio**
Standard Plane	First Black – First Blue	4.0000	20.0000	5.0000
Standard Plane	Last Black – Last Blue	4.0000	0.0000	0.0000
Tilted Plane	First Black – First Blue	4.0000	32.9316	8.2329
Tilted Plane	Last Black – Last Blue	4.0000	0.0000	0.0000

The results in Table 1 show that Euclidean distances between corresponding 3D point pairs are not preserved after projection onto the image plane. Although the original 3D distances between the selected black and blue points are identical in all cases, the projected distances differ substantially across projection configurations. For the standard image plane, the first point pair undergoes a moderate magnification, while the second pair collapses to a zero projected distance, indicating alignment along the viewing direction. When the image plane is tilted, the effect becomes more pronounced: the same first point pair experiences a larger magnification due to depth variation across the plane, whereas the last point pair again projects to a single location. These observations

confirm that perspective projection distorts distances in a depth-dependent manner and that the orientation of the image plane plays a significant role in shaping the resulting geometric relationships.

The observed changes in distance occur because the pinhole camera performs a perspective projection, which is a nonlinear transformation. In this model, points are projected onto the image plane by intersecting rays from the camera center with the plane. As a result, distances in the image depend on the depth of the points relative to the camera. Points that are closer to the camera appear magnified, while points farther away appear compressed. Consequently, Euclidean distances measured in 3D space are generally not preserved after projection.

Part II: Rendering Views from a 3D Point Cloud (30 Points)

In this part, the pinhole camera model developed in Part I is extended to a full 3D point cloud. The point cloud is treated as a fixed world coordinate system, while a virtual pinhole camera is moved to different positions and orientations to generate multiple 2D views of the same static scene. No rotation or transformation is applied to the point cloud itself meaning all variations in the rendered images result solely from changes in the camera pose. I used plyfile to load the 3D cloud points in memory for further experimentation.

World Coordinate System

The world coordinate system is defined directly by the input point cloud. All 3D points are used in their original coordinates and are represented as

$$P_{\text{world}} = (X, Y, Z).$$

The center of the point cloud is computed and used as a reference point toward which all camera views are oriented.

Camera Centers and Orientations

Six camera poses are defined to generate canonical views of the scene: front, back, left, right, top, and bottom. Each camera pose is specified by a camera center

$$C = (C_x, C_y, C_z),$$

placed at a fixed distance from the point cloud center along one of the principal axes.
For each camera position, a rotation matrix R is computed, orienting the camera toward the center of the scene while maintaining a consistent up direction.

The transformation from world coordinates to camera coordinates follows:

$$P_{\text{cam}} = R(P_{\text{world}} - C).$$

Perspective Projection of the Point Cloud

For each camera pose, all 3D points are transformed into the camera coordinate frame using the defined rotation matrix and camera center. Perspective projection is then applied using the pinhole camera equations:

$$x = f\frac{X_c}{Z_c}, \quad y = f\frac{Y_c}{Z_c},$$

where (X_c, Y_c, Z_c) are the coordinates of a point in the camera frame and f is the focal length.

The projected points are then mapped to 2D image coordinates using fixed camera intrinsics, ensuring consistency across all views.

Image Formation

For each viewpoint, a 2D image is generated by plotting the projected points onto an image grid with a fixed resolution. The same focal length and principal point are used for all views so that differences between images arise solely from changes in the camera pose. Each pixel may represent one or more projected points, resulting in a point-based rendering of the scene.

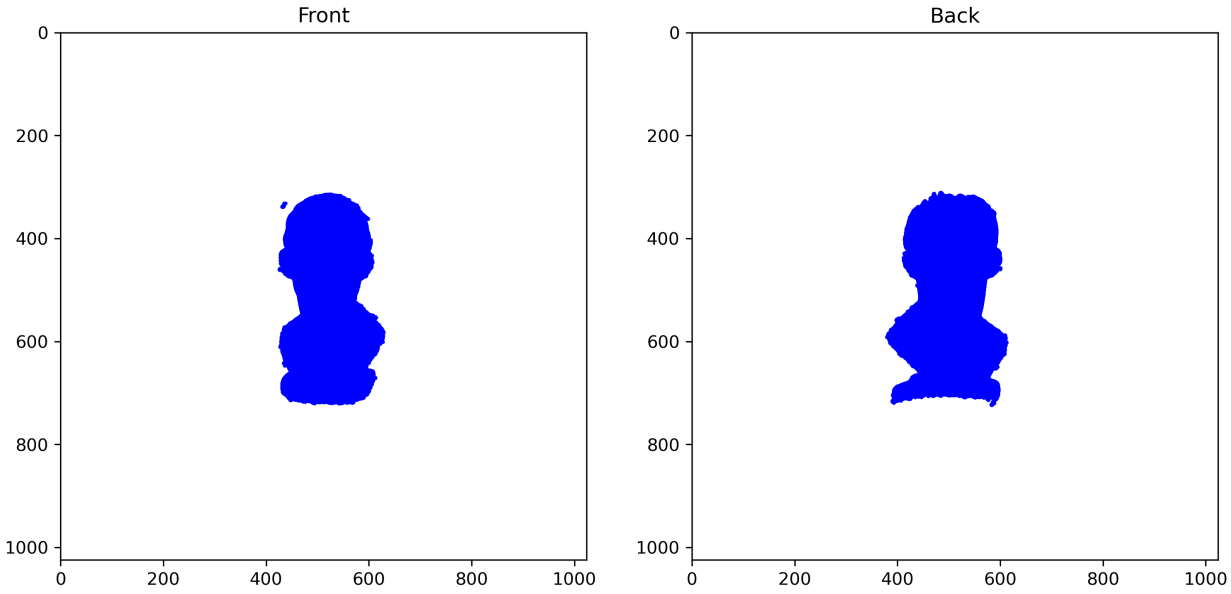

Figure 3: Top and Bottom view

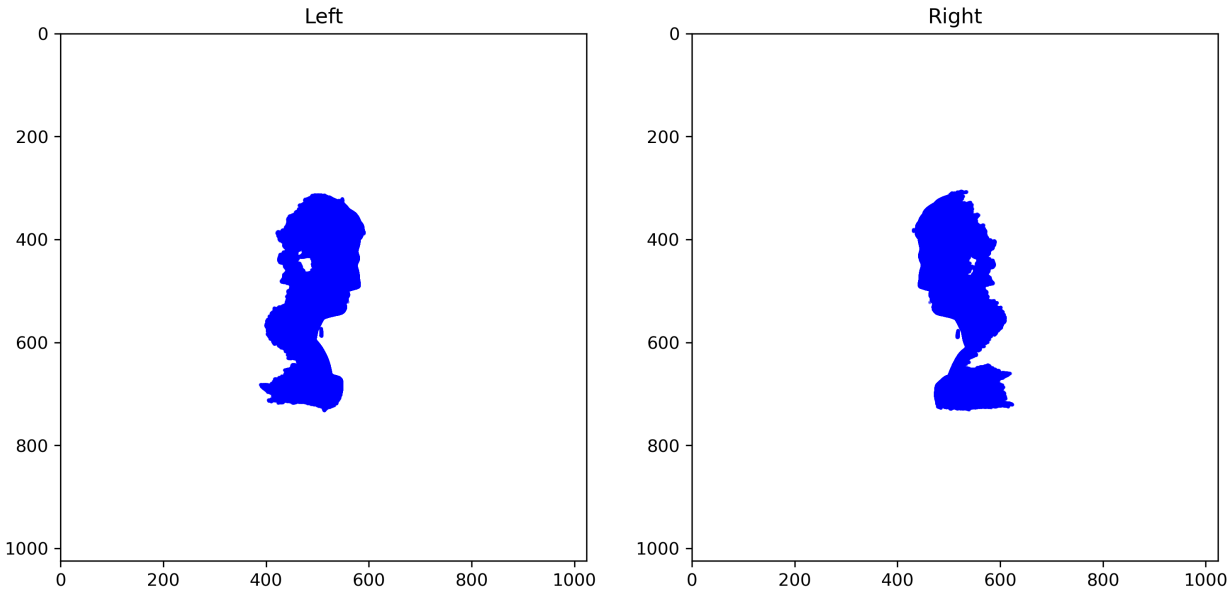

Figure 4: Left and Right View

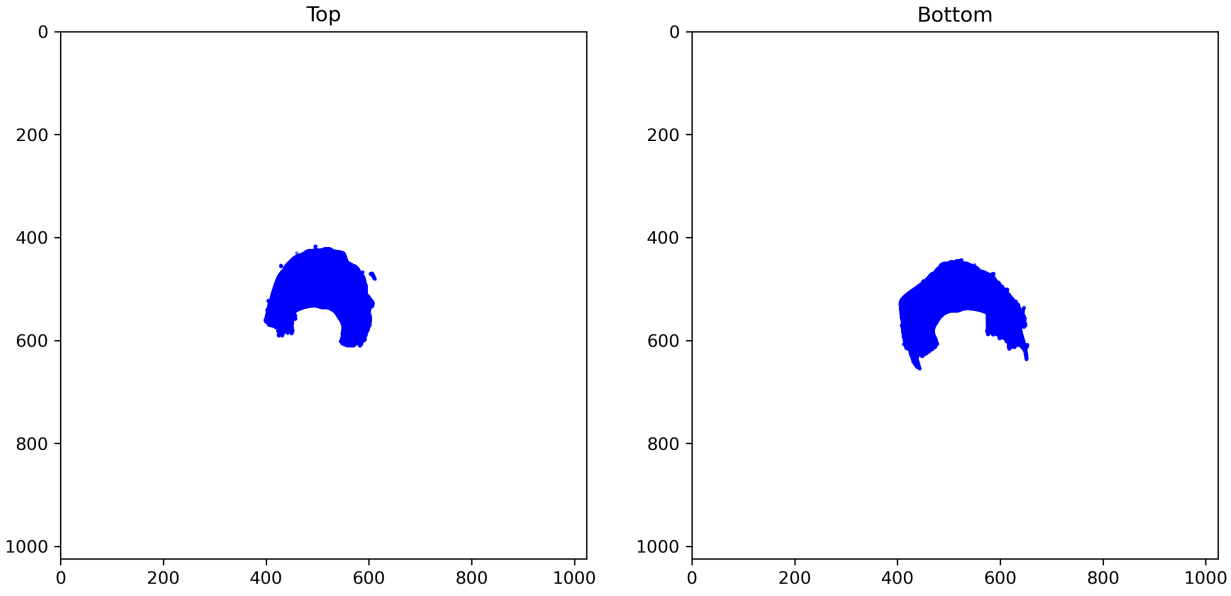

Figure 5: Front and Back view

Camera Motion Interpretation

Changing the camera pose alters the relative positions of 3D points with respect to the camera coordinate system. Although the scene itself remains static, variations in camera position and orientation modify the depth and angular relationships of points, producing different 2D projections. This demonstrates how image sequences in graphics engines and simulators are formed by moving a virtual camera through a fixed 3D environment.

Analysis Questions

How does camera orientation affect the perceived structure of the scene? Camera orientation determines which surfaces and spatial relationships are visible. Different orientations emphasize different depth cues and occlusions, leading to changes in the perceived structure and layout of the scene.

Why does perspective distortion vary across views? Perspective distortion depends on the relative depth of points along the viewing direction. As the camera orientation changes, different regions of the scene move closer to or farther from the camera, causing variations in apparent scale and distortion across views.

How does this experiment relate to camera navigation in video games or simulators? This experiment mirrors camera navigation in virtual environments, where a fixed 3D world is rendered from continuously changing camera poses. Video games and simulators use the same principles of camera motion and perspective projection to generate dynamic visual experiences.

Additional Camera Simulation (Spectacles Casing)

Having simulated camera movement on the 3D cloud point provided, I applied the same function on new 3D cloud points portraying a spectacle casing. These 3D cloud points were captured by Revo Scan software and a 3D scanner and loaded into my environment using pyfile for further experimentation. These are the results from the experimentation:

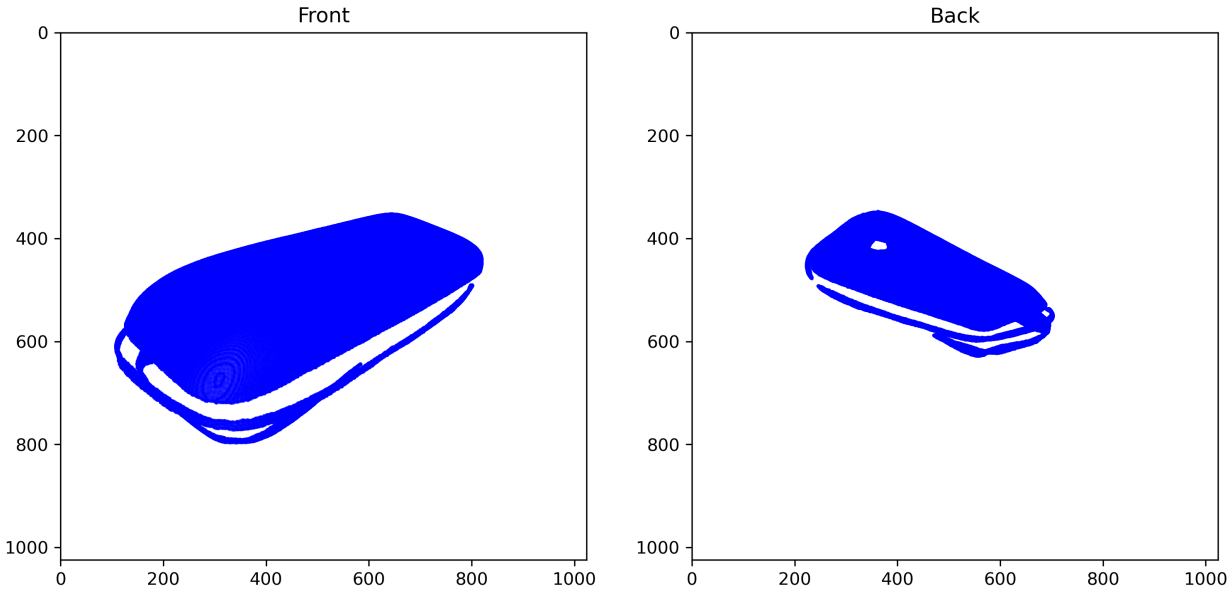

Figure 6: Front and Back view

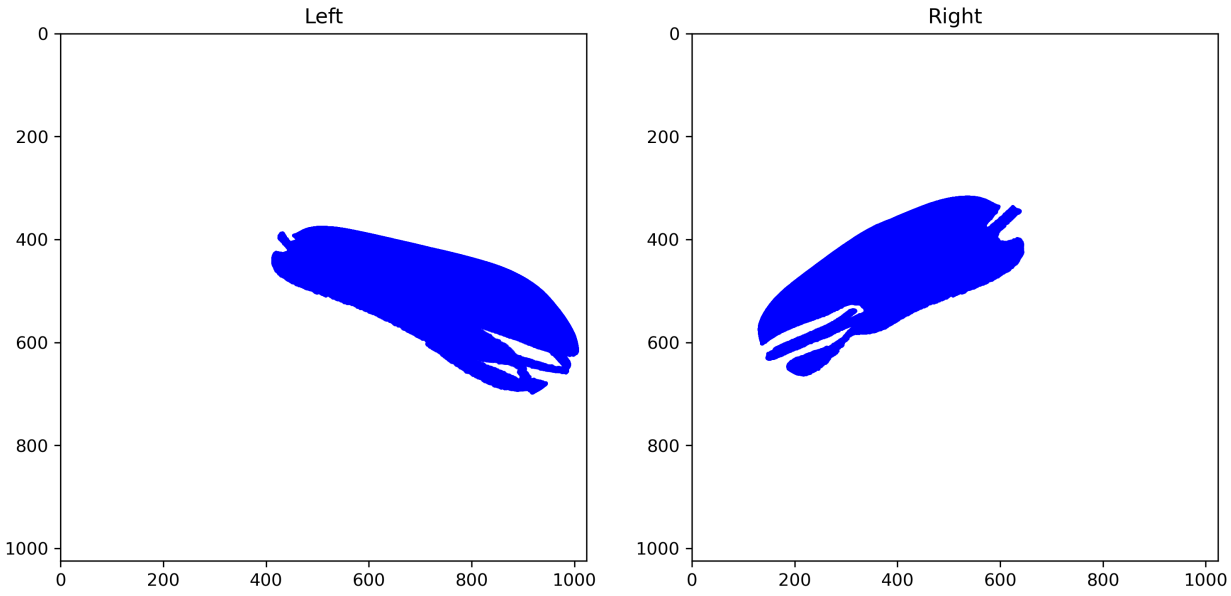

Figure 7: Left and Right view

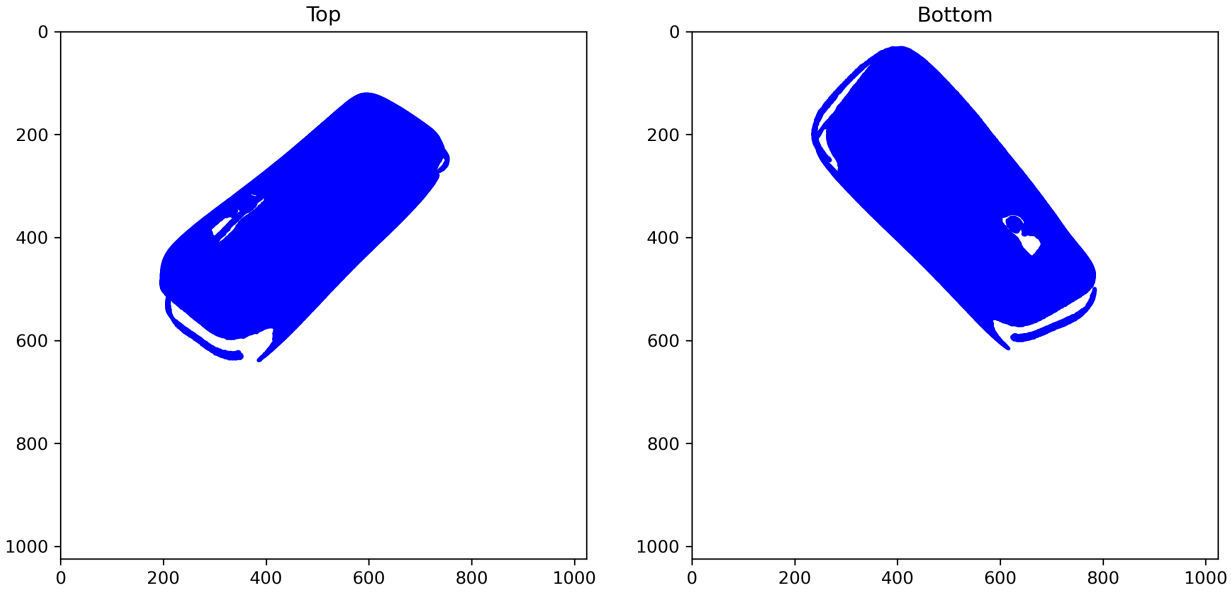

Figure 8: Top and Bottom view

Extension: Depth Visualization and Camera Motion

To further explore the geometric behavior of perspective projection, I conducted additional experiments using the provided `scene.ply` full 3D point cloud. First, the projected points were color-coded based on their depth relative to the camera. Points closer to the camera were assigned warmer colors, while points farther away were assigned cooler colors. This visualization provides an intuitive representation of depth variation within the scene and makes the three-dimensional

structure more perceptible in the resulting 2D images. This was the result of the depth color code experiment.:

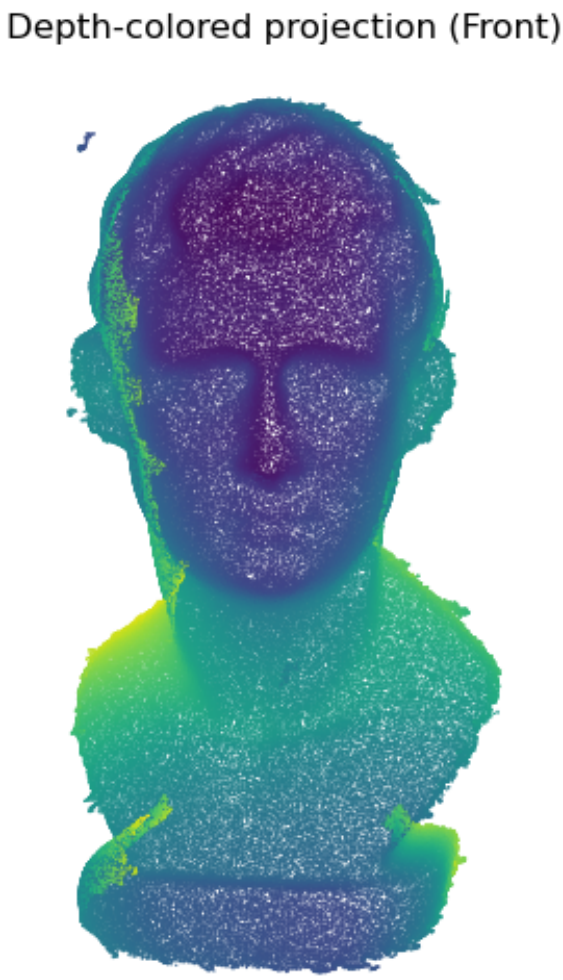

Figure 9: Depth-Color Projection Front View

Next, a smooth camera orbit around the static point cloud was simulated by continuously varying the camera center along a circular trajectory while maintaining a fixed viewing target. For each camera position, the point cloud was projected using the same camera intrinsics, and the resulting images were saved as sequential frames. These frames were subsequently combined to form a video using an online platform VEED, illustrating how continuous camera motion through a fixed 3D environment produces changing image appearances. This experiment closely mirrors how virtual cameras are used in video games and simulation environments to navigate three-dimensional

scenes. The results of this task have been uploaded to the following Google Drive Folder

Finally, a comparison was performed between perspective and orthographic projection using identical camera poses. Under perspective projection, points exhibited depth-dependent scaling, with distant regions appearing smaller. In contrast, orthographic projection preserved relative sizes regardless of depth, resulting in a visually flatter representation of the scene. This comparison highlights the role of perspective projection in conveying depth cues and visual realism, as well as the suitability of orthographic projection for applications requiring metric accuracy rather than perceptual depth. Here is a visual difference between the two:

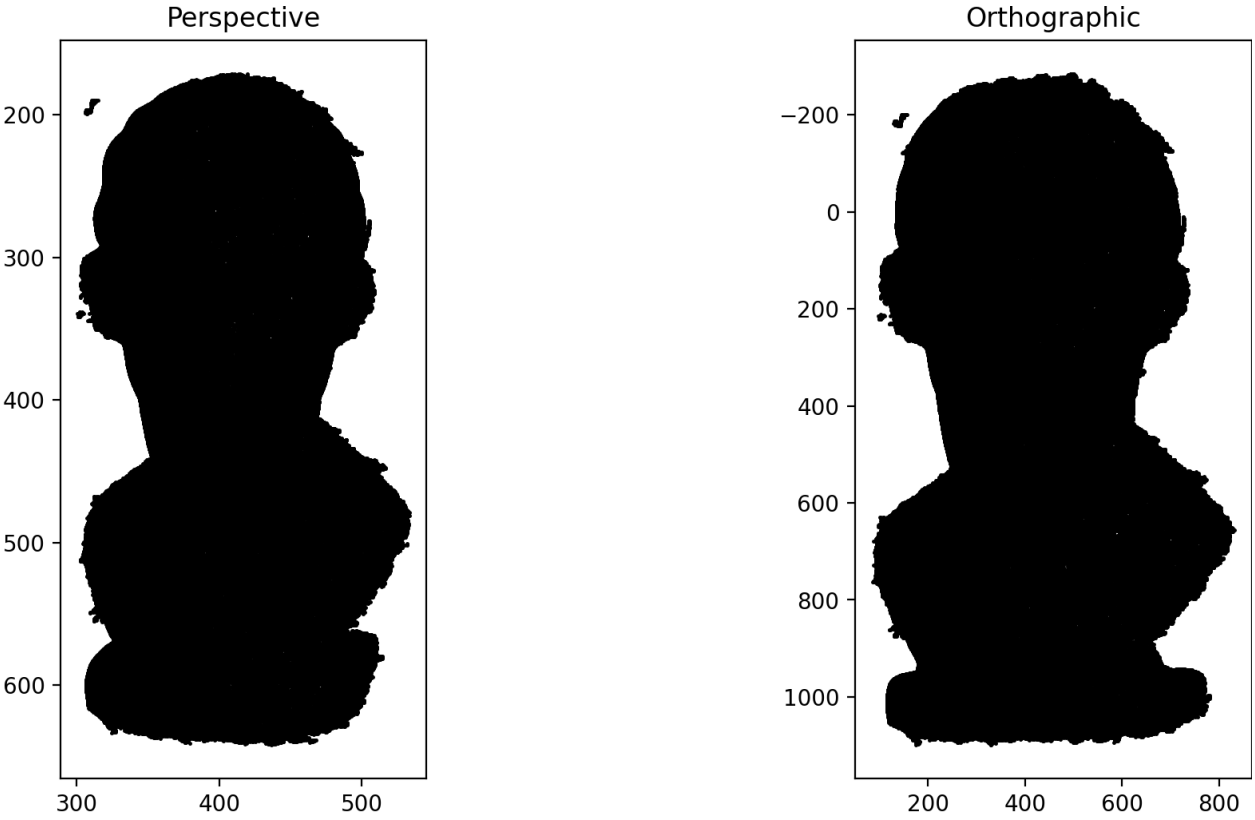

Figure 10: Perspective vs orthographic projection

Part III: Image Demosaicing

Figure 11 presents a qualitative comparison between the original raw Bayer image and the reconstructed RGB image obtained using bilinear interpolation. The demosaicing process successfully recovers full-color information across the image, producing visually coherent results in smooth regions and areas with gradual intensity variations.

(a) Original raw Bayer image

(b) Demosaiced RGB image

Figure 11: Comparison between the raw Bayer input and the reconstructed RGB image obtained using bilinear demosaicing.

Observations and Limitations

The implemented bilinear interpolation method relies exclusively on local neighborhood averaging to estimate missing color values. As a result, the reconstructed image exhibits smooth color transitions in homogeneous regions. However, this simplicity also introduces several limitations.

In regions containing sharp edges or fine textures, noticeable artifacts may appear. These include color bleeding across edges, loss of high-frequency detail, and zippering effects along strong intensity transitions. Such artifacts arise because the interpolation process does not consider edge direction or local image structure, treating all neighborhoods uniformly regardless of underlying content.

Additionally, bilinear interpolation assumes linear variation of color values, which is often violated in natural scenes, particularly at object boundaries and textured regions.

Potential Improvements

The demosaicing process could be enhanced by incorporating image structure awareness into the interpolation strategy. One improvement involves edge-directed interpolation, where interpolation is performed preferentially along edge directions rather than across them. This approach helps preserve edge sharpness and reduces color artifacts.
Another enhancement is high-quality linear interpolation, which introduces correction terms based on local intensity gradients. Such methods reduce color misalignment while maintaining relatively low computational complexity. These improvements offer a better balance between visual quality and efficiency, especially for real-time or low-power imaging systems.

Other Demosaicing Algorithms

Beyond bilinear interpolation, several advanced demosaicing algorithms have been developed. Frequency-domain approaches exploit correlations between color channels to improve reconstruction accuracy, particularly in textured regions. More recently, learning-based demosaicing methods, including convolutional neural networks, have demonstrated superior performance in preserving fine details and minimizing artifacts.

Although these advanced methods produce higher-quality results, they typically require more computational resources and larger memory footprints. Consequently, bilinear interpolation remains a practical baseline for low-cost cameras and real-time applications, where simplicity and efficiency are critical constraints.

Overall, the implemented bilinear demosaicing method provides an effective and computationally efficient solution for reconstructing color images from raw Bayer data. Although it exhibits limitations in edge preservation and detail reconstruction, it serves as a strong baseline for understanding the trade-offs involved in image demosaicing and motivates the use of more advanced techniques in quality-critical applications.