Homework 1

Manyara Bonface Baraka - mbaraka

February 2, 2025

Part I: Pinhole Camera

Step

- Defined the function to **perform ray intersection with a standard plane:**

  The function `ray_intersection_standard(C, P, Z0)` computed the intersection of a ray with a standard image plane at $Z = Z_0$.

  Given:

  - **Camera position:** $C = (c_x, c_y, c_z)$
  - **3D point:** $P = (X, Y, Z)$
  - **Standard image plane:** $Z = Z_0$

  The equation for the ray from C through P is:

  $$(x, y, z) = (c_x, c_y, c_z) + t \cdot (X - c_x, Y - c_y, Z - c_z)$$

  where t is the control movement along the ray.

  To find the intersection with the plane $Z = Z_0$, we solve for t:

  $$t = \frac{Z_0 - c_z}{Z - c_z}$$

  Using this t, we find the intersection coordinates:

  $$x_{\text{intersect}} = c_x + t(X - c_x)$$

  $$y_{\text{intersect}} = c_y + t(Y - c_y)$$

  Thus, the intersection is:

  $$\left(c_x + \frac{Z_0 - c_z}{Z - c_z}(X - c_x), \quad c_y + \frac{Z_0 - c_z}{Z - c_z}(Y - c_y) \right)$$

  This intersection give us the **2D projection** of P onto the standard image plane at Z_0.

- Defined a function to **perform ray intersection on tilted planes:**

  The function `ray_intersection_tilted(C, P, a, b, c, d)` computed the intersection of a ray with a tilted image plane defined by:

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

  Given:

  - **Camera position:** $C = (c_x, c_y, c_z)$
  - **3D point:** $P = (X, Y, Z)$
  - **Plane equation:** $ax + by + cz + d = 0$

  The equation for the ray from C through P is:

$$(x, y, z) = (c_x, c_y, c_z) + t \cdot (X - c_x, Y - c_y, Z - c_z)$$

  where t controls movement along the ray just as the one on the standard plane.

  Substitute x, y, z into the plane equation to find the intersection of the ray on the tilted plane:

$$= a(c_x + t(X - c_x)) + b(c_y + t(Y - c_y)) + c(c_z + t(Z - c_z)) + d = 0$$

$$= ac_x + at(X - c_x) + bc_y + bt(Y - c_y) + cc_z + ct(Z - c_z) + d = 0$$

  Therefore t:

$$t = \frac{-(ac_x + bc_y + cc_z + d)}{a(X - c_x) + b(Y - c_y) + c(Z - c_z)}$$

  Using$t$, the intersection coordinates:

$$x_{\text{intersect}} = c_x + t(X - c_x)$$

$$y_{\text{intersect}} = c_y + t(Y - c_y)$$

$$z_{\text{intersect}} = c_z + t(Z - c_z)$$

  Thus, the intersection point on the tilted plane is:

$$(c_x + t(X - c_x), \quad c_y + t(Y - c_y), \quad c_z + t(Z - c_z))$$

  This gives the **3D intersection** of the ray with the tilted image plane.

- Define a function to **Project the image to a plane:**

  The function `project_to_image(C, P, Z0, f, sx, sy, cx, cy, plane_type, a, b, c, d)` projects a 3D point $P = (X, Y, Z)$ onto a 2D image plane, either standard or tilted.

  The function performed in this format:

  – If the plane type is **standard**, use the function:

  $$(x_{\text{intersect}}, y_{\text{intersect}}) = \text{ray\_intersection\_standard}(C, P, Z_0)$$

  where the intersection with the standard image plane at $Z = Z_0$ is computed in step 1.

  – If the plane is **tilted**, use:

  $$(x_{\text{intersect}}, y_{\text{intersect}}, z_{\text{intersect}}) = \text{ray\_intersection\_tilted}(C, P, a, b, c, d)$$

  where the intersection with the tilted plane $ax + by + cz + d = 0$ is computed in step 2.

The function **Convert to Image Coordinates** The projected coordinates are normalized:

$$u' = \frac{x_{\text{intersect}} - c_x}{f}$$

$$v' = \frac{y_{\text{intersect}} - c_y}{f}$$

where f is the focal length.
The function then **Applies Scaling and Centering**:

$$u = \lfloor s_x u' + c_x \rfloor$$

$$v = \lfloor s_y v' + c_y \rfloor$$

where:

- s_x, s_y are scaling factors,
- c_x, c_y define the image center.

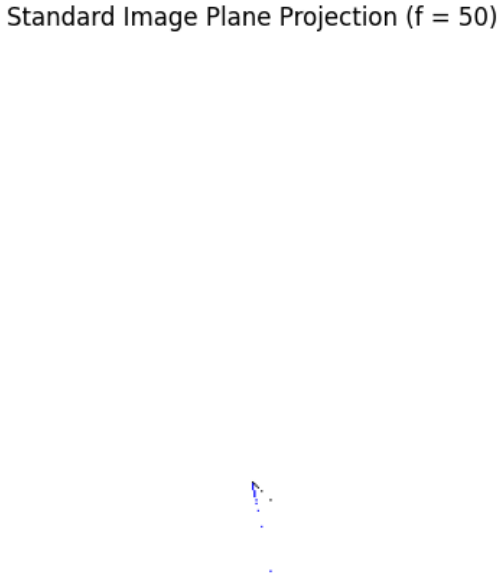
Then finally the pixel coordinates on the image plane are:

$$(u, v) = \left(\lfloor s_x \cdot \frac{x_{\text{intersect}} - c_x}{f} + c_x \rfloor, \quad \lfloor s_y \cdot \frac{y_{\text{intersect}} - c_y}{f} + c_y \rfloor \right)$$

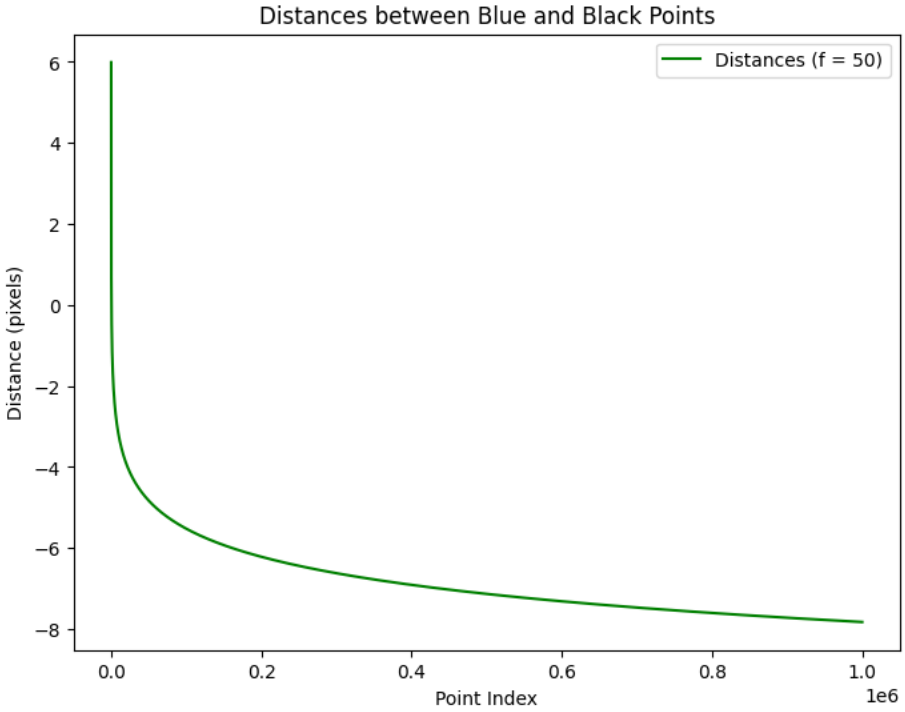
The functions were then applied in subsequent test to test the image formation

Observation

- **With the standard plane - [Focal length of 50]**

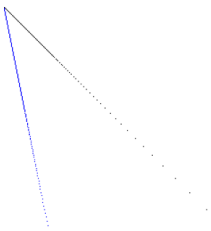


- **Plotted the Distance between the black and blue points** - the distance was so small to be noticed. I used log to have a better vision and clarity of the plot:

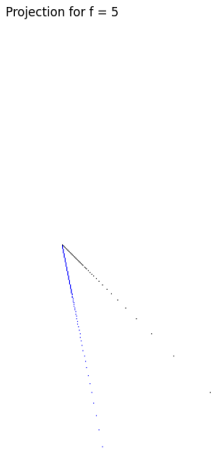


Experimented with different Focal lengths:

- **Focal length of 1**

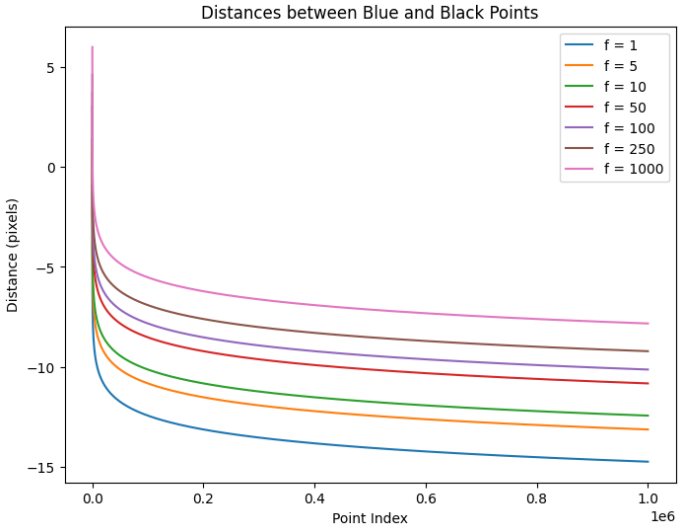


- **focal length of 5**

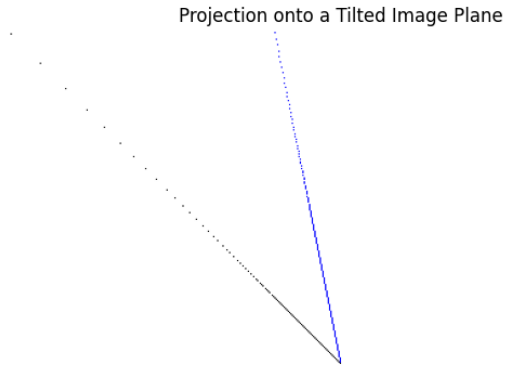


- **focal length of 1000**

- **Plotted distance between the black and blue points**



Experimentation with tilted image plane



Distance Between 3D and 2D Points

Points	Distance (3D)	Distance (2D)
First black and first blue point	4.0000	14.7020
Last black and last blue point	4.0000	0.0000

Insights

- **Images with different focal length**
  - As the focal length incresed the image became smaller this justified the theoretical concept of the focal length affect the zoom, an increase in focal length leads to the reduction of the size of the image.
  - For different focal lengths the distances are more negative as the focal length became smaller this means that the projections are significantly displaces, and for larger focal lengths the distances are closer to zero these means that the points are lining more closely.
- **Tilted Image plane:** the tilted image plane wraps the image percepective this cause the point to appear stretched compared to the original image.
- **Distance between 3D and 2D first and last points**
  - The distance between the fist and the last for the 3D is consiten this indicates the spatial relationship that remains unchanged before prjections.
  - For 2D the distance changes significantly for the first points but reduces to 0 for the last points.
  - This indicates the **perspective effect on image formation** the 2D values change due to the projection, and merges back to zero because of the image perception / projection to the ideal point.

Part II (Image Demosaicing)

Steps

Bayer to RGB Conversion Using Bilinear Interpolation

The function `bayer2rgb` converts a Bayer-patterned grayscale image to an RGB image using bilinear interpolation. The image is processed pixel by pixel, and the color channels (Red, Green, Blue) are estimated based on the surrounding pixels:

- This function takes three inputs:
  - `image`: The Bayer-patterned grayscale image to be converted.
  - `height`: The height (number of rows) of the image.

– `width`: The width (number of columns) of the image.

The output is an RGB image where the Red, Green, and Blue channels are computed based on the Bayer pattern.

- Initialize RGB ImageThe new image `im_color` is initialized the same size as the input `image`, but with 3 channels (for Red, Green, and Blue). The RGB channels are initialized to zero, and later filled with computed values.
- Processing Red and Green Pixels The Bayer pattern is assumed to alternate between Green and Red pixels, with Green pixels being on the even rows and columns. The Red channel is processed first:

```
if (y % 2 == 0) and (x % 2 == 0):  # Get the red channel
    im_color[y, x, 0] = image[y, x]
```

The condition `(y % 2 == 0) && (x % 2 == 0)` checks for the Red pixels. The Red channel (`im_color[y, x, 0]`) is assigned the pixel value from the Bayer image.

- Estimating Green for Red Pixels The Green channel is estimated using the surrounding pixels. Green pixels are either directly above or below (for vertical neighbors) or left and right (for horizontal neighbors). The average of these neighboring pixels is taken.

```
if y > 0 and y < height - 1:
    im_color[y, x, 1] = (image[y-1, x] + image[y+1, x]) / 2
if x > 0 and x < width - 1:
    im_color[y, x, 1] += (image[y, x-1] + image[y, x+1]) / 2
    im_color[y, x, 1] /= 2
```

- *Estimating Blue for Red Pixels The Blue channel is estimated diagonally from the neighboring pixels. These diagonally placed Green pixels are averaged to estimate the Blue channel.

```
if y > 0 and y < height - 1 and x > 0 and x < width - 1:
    im_color[y, x, 2] = (image[y-1, x-1] + image[y-1, x+1] +
                         image[y+1, x-1] + image[y+1, x+1]) / 4
```

- Processing Blue and Green Pixels Next, the Blue pixels are processed:

```
elif (y % 2 == 1) and (x % 2 == 1):  # Get blue channel
    im_color[y, x, 2] = image[y, x]  # take the blue color
```

Here, the condition `(y % 2 == 1) && (x % 2 == 1)` checks for the Blue pixels. The Blue channel is directly taken from the input image.

- Estimating Green for Blue Pixels Similar to the Red pixels, the Green channel is estimated using the surrounding pixels. This is done both vertically and horizontally, and the average is taken.

```
if y > 0 and y < height - 1:
    im_color[y, x, 1] = (image[y-1, x] + image[y+1, x]) / 2
if x > 0 and x < width - 1:
    im_color[y, x, 1] += (image[y, x-1] + image[y, x+1]) / 2
    im_color[y, x, 1] /= 2
```

- Estimating Red for Blue Pixels The Red channel is estimated diagonally from the neighboring Green pixels. The average of the surrounding Red pixels is taken.

```
if y > 0 and y < height - 1 and x > 0 and x < width - 1:
    im_color[y, x, 0] = (image[y-1, x-1] + image[y-1, x+1] +
                         image[y+1, x-1] + image[y+1, x+1]) / 4
```

- Processing Green Pixels The Green pixels are handled separately as they are already present in the Bayer pattern. The Green values are directly taken from the input image and the neighboring pixels are used for interpolation.

```
else:  # Green pixel (G)
    im_color[y, x, 1] = image[y, x]
```

- Estimating Red and Blue for Green Pixels Green pixels on Red rows and Green pixels on Blue rows are handled separately. The Red and Blue channels are estimated based on the surrounding pixels as follows:

  For Green pixels on Red rows:

```
if y % 2 == 0:  # Green on Red row
    if x > 0 and x < width - 1:
        im_color[y, x, 0] = (image[y, x-1] + image[y, x+1]) / 2
    if y > 0 and y < height - 1:
        im_color[y, x, 2] = (image[y-1, x] + image[y+1, x]) / 2
```

  For Green pixels on Blue rows:

```
else:  # Green on Blue row
    if x > 0 and x < width - 1:
        im_color[y, x, 2] = (image[y, x-1] + image[y, x+1]) / 2
    if y > 0 and y < height - 1:
        im_color[y, x, 0] = (image[y-1, x] + image[y+1, x]) / 2
```

  Finally, the function returns the RGB image with the estimated colors for each pixel.

```
return im_color
```

Observation

Image missing pixels

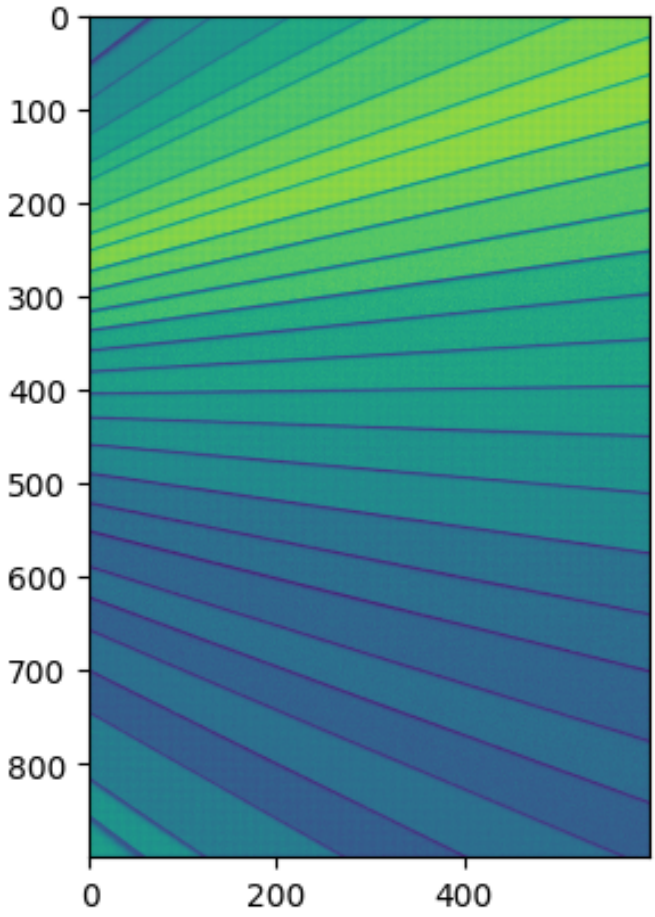
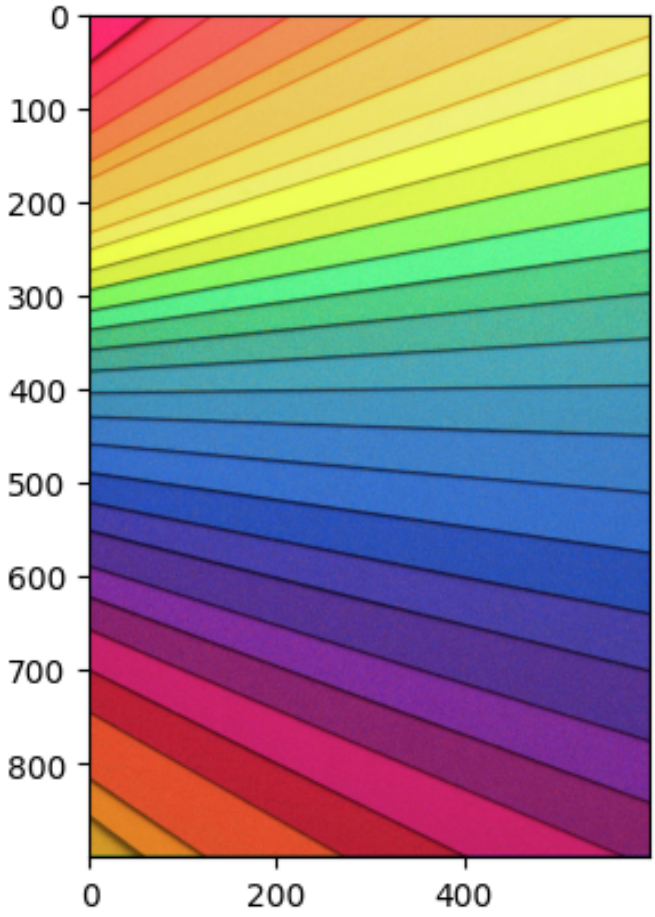


Image After Bilinear interpolation

Insights

Demosaicing is the process of filling in missing color information using the bayer filter.

- **Limitation of Bilinear Aligo**
  - **Loss of high frequency details in images:** Bilinear assumes that color information caries smoothly this leads to blur of the sharp edges and loss of textures.


0
100
200
300
400
500
600
700
800
0
200
400

  - **Color artifacts:** The algorithm treats all pixels equal without differentiating the edge and the smooth region, for instance a boundary with red-blue boundary might get an unwanted green which will cause a visible color fridge.

- **Potential improvements**
  - **Frequency filtering** the algorithm could analyze the image using frequency and filter out the unwanted artifacts this will increase the color accuracy.
  - **Weightig in interpolation** instead of having fixed weight interpolation, assigning weights based on the color similarity will preserve the colors of the image.

- **Alternative Models**
  - **Deep learning** use of CNNS to learn complex demosaicing patterns
  - **Hamilton-Adams interpolation** interpolates the missing colors along the edge rather than across them, this reduces the color artifacts and improves the edge sharpness.