DATA VISUALIZATION: PIE CHARTS AND OPENGL

Programming assignment on fruit preference survey

GROUP 9

TEAM

- Ian Ndolo Mwau SCT 211-0034/2022
- Maureen Mukami SCT 211-0052/2022
- David Nzambuli SCT 211-00682022
- Pharis Kariuki SCT 211-0033/2022

INTRODUCTION

- Objective:
  - Create a pie chart using OpenGL to visualize fruit preference data.
  - Demonstrate customization by applying fruit-resembling colors for each section.
  - Demonstrate grayscale conversion using OpenGL

PROBLEM 1 – PIE CHART FOR FRUIT PREFERENCES

Data Table:

Fruit	People
Orange	36
Banana	41
Kiwi Fruit	19
Mango	28
Grapes	30
Ovacado	16

PROBLEM1:FRUIT PREFERENCES PIE CHART

Objective

- Draw a piechart with title "Fruit Preferences Survey"
- Calculate percentages from the data.
- Place labels outside each section.

FRUIT PREFERENCES PIE CHART

Code Walkthrough

CODE STRUCTURE OVERVIEW

Main Components:

- display(): Renders the pie chart
- reshape(): Maintains aspect ratio and orthographic projection
- drawBitmapText(): Displays chart and label text
- main(): Initializes GLUT and starts the rendering loop

DRAWING THE PIE CHART

Rendering Approach:

- Calculate total of all values
- For each slice:
  - Calculate percentage and angle
  - Draw slice using GL_TRIANGLE_FAN
  - Label each section outside the circle using trigonometry
  - Draw radial boundaries

IMPLEMENTATION USING C++

1. Slice Calculation
   - float sliceAngle = (values[i] / total) * 360.0f; // Convert % to angle
   - Uses trigonometry (sin/cos) to plot points.

2. Label Positioning

   float labelX = centerX + cos(midAngle) * radius;

   float labelY = centerY + sin(midAngle) * radius;

   - Special handling for “Banana” and “Kiwifruit” labels to improve readability.

3. Rendering Pipeline
   - display() → reshape() → main().
   - Output: Clean pie chart with dynamic resizing.

IMPLEMENTATION USING PYTHON

Code Snippet:

- labels = ['Ovacado', 'Orange', 'Banana', 'Kiwifruit', 'Mangos', 'Grapes']
- sizes = [36, 41, 19, 28, 30, 16]
- colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0','#ffb3e6']
- plt.pie(sizes, labels=[f"{l} ({s/sum(sizes)*100:.1f}%)" for l, s in zip(labels, sizes)],
- colors=colors, startangle=90)
- plt.title("Fruit Consumption Pie Chart")
- plt.axis('equal')

CODE EXPLANATION

Function used:plt.pie() from matplotlib.pyplot

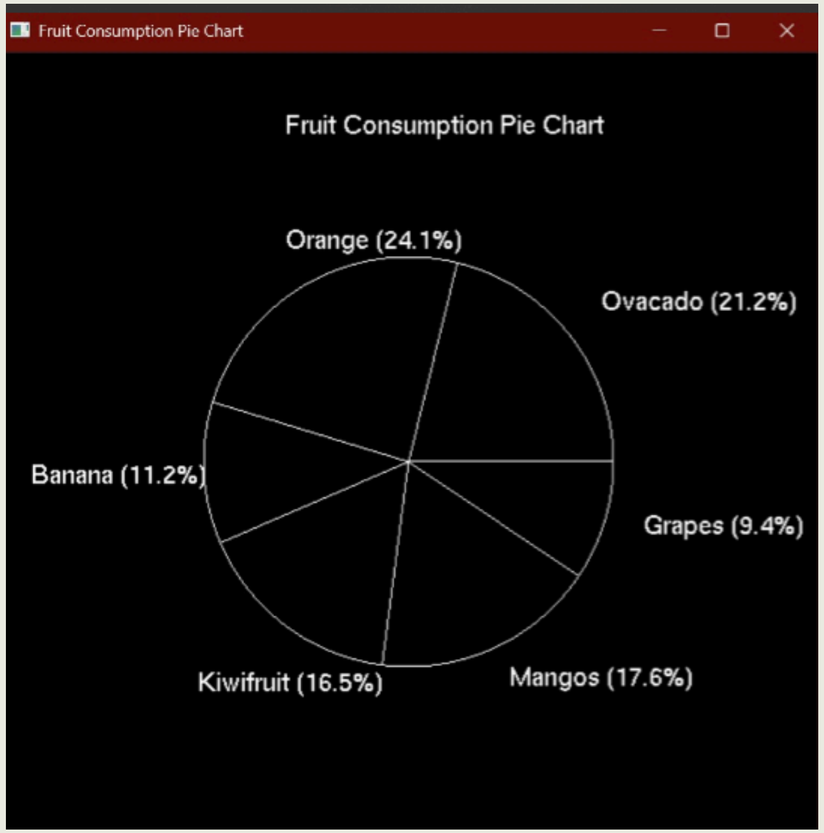
Code Explanation:

- sizes: Values for each fruit.
- labels: Fruit names.
- startangle=90: Rotates chart for better visual start.
- axis('equal'): Ensures pie is circular.
- labels=...: Adds names and percentage outside each slice.

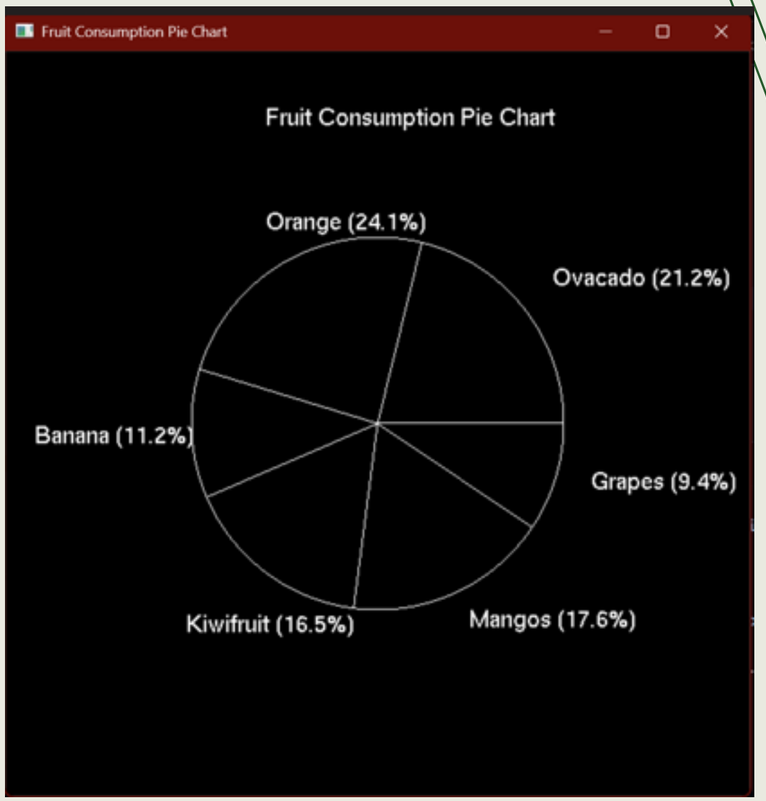
-Python's matplotlib makes it easy to create percentage-labeled visualizations.
-Chart is accurate, readable, and easily customizable.

FRUIT PIE CHART

Output Using C++



Output Using Python



C++ VS. PYTHON COMPARISON

1. Text Rendering

C++

```
// C++ (GLUT)  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,
*c);
```

Python

```
# Python (PyOpenGL)
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, ord(ch))
```

2. Main Loop Structure

C++

```
int  main()  {glutInit(&argc,  argv);glutCreateWindow("Pie
Chart");glutDisplayFunc(display);  // Callback registration
}
```

Python

```
def main():
    glutInit(sys.argv)
    glutCreateWindow(b"Pie Chart")
    glutDisplayFunc(display)  # Same callback

if __name__ == "__main__":
    main()
```

3. Label Repositioning Logic

C++

```
//    Adjust    Banana    label    outward    (extra    space)if
(std::string(labels[i]) == "Banana") {
     labelRadius = radius + 0.42f;   // Push further out}//
Adjust Kiwifruit label leftwardif (std::string(labels[i]) ==
"Kiwifruit") {
    labelRadius = radius + 0.20f;  // Slightly out
    labelX -= 0.09f;               // Shift left}
```

Python

```
# Banana adjustmentif labels[i] == "Banana":
       labelRadius  =  radius  +  0.42    #  Push  further  out#
Kiwifruit adjustmentif labels[i] == "Kiwifruit":
    labelRadius = radius + 0.20  # Slightly out
    labelX -= 0.09               # Shift left
```

Key Takeaways

Same Output: Both versions produce identical visuals.

Syntax Differences:

- C++: Explicit types, manual string formatting.
- Python: Dynamic typing, f-strings.

Workflow: Python is quicker for prototyping; C++ offers more control.

PROBLEM 2 – FRUIT COLORED PIE CHART

Objective

- Redraw the pie chart so that each section corresponds to the color of the fruit represented when ripest.

FRUIT-COLORED PIE CHART

Color Selection Explanation

- Avocado: The greenish color (0.82, 0.81, 0.41) represents the distinctive flesh color of a ripe avocado.
- Orange: A vibrant orange shade (0.93, 0.55, 0.14) that mimics the citrus fruit's characteristic color.
- Banana: A bright yellow (1.0, 0.87, 0.35) selected to match a perfectly ripe banana skin
- Kiwifruit: Deeper green (0.43, 0.51, 0.04) representing the kiwi's flesh once cut open
- Mango: Orange-red shade (1.0, 0.51, 0.26) capturing the warm tones of a ripe mango
- Grapes: Rich purple (0.44, 0.18, 0.66) mimicking the deep color of dark grapes

FRUIT COLORED PIE CHART

Code Walkthrough

FRUIT-COLORED PIE CHART

Data Structure Setup

C++

```
// Data for the pie chart
static float values[] = {36.0f, 41.0f, 19.0f, 28.0f, 30.0f, 16.0f};
static const char *labels[] = {"Ovacado", "Orange", "Banana", "Kiwifruit", "Mangos", "Grapes"};
static const int NUM_SLICES = sizeof(values) / sizeof(values[0]);
```

Python

```
# Survey data
fruits = ['Avocado', 'Orange', 'Banana', 'Kiwifruit', 'Mango', 'Grapes']
people = [36, 41, 19, 28, 30, 16]
total = sum(people)  # Total number of responses
percentages = [round((count / total) * 100, 1) for count in people]
```

Color Array Definition

C++

```
// Fruit colors (R, G, B)
static float colors[][3] = { {0.34f, 0.51f, 0.01f}, // Avocado
{1.0f, 0.65f, 0.0f}, // Orange
{1.0f, 0.87f, 0.35f}, // Banana
{0.65f, 0.89f, 0.18f}, // Kiwifruit
{1.0f, 0.51f, 0.26f}, // Mango
{0.44f, 0.18f, 0.66f} // Grapes;
```

Python

```
# Fruit colors (R, G, B)
fruit_colors = [
    (0.82, 0.81, 0.41),   # Avocado
    (0.93, 0.55, 0.14),   # Orange
    (1.0, 0.87, 0.35),    # Banana
    (0.43, 0.51, 0.04),   # Kiwifruit
    (1.0, 0.51, 0.26),    # Mango
    (0.44, 0.18, 0.66)    # Grapes
]
```

FRUIT-COLORED PIE CHART

Window/Application Setup

C++

```
// Window dimensions
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;

// Program entry point
int main(int argc, char **argv) {
    glutInit(&argc, argv);                              // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);        // Double buffer and
RGB color
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);    // Set window size
    glutCreateWindow("Fruit Preferences Pie Chart");    // Create window
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);               // Set white
background
    glutDisplayFunc(display);                           // Register display
callback
    glutReshapeFunc(reshape);                           // Register reshape
callback
    glutMainLoop();                                     // Enter main event
loop
    return 0;
}
```

Python

```
class PieChartApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Fruit Preferences Pie Chart")

        # Set window size
        self.root.geometry("800x800")

        # Create the pie chart
        fig = create_pie_chart()

        # Create a canvas to display the pie chart
        self.canvas = FigureCanvasTkAgg(fig, master=self.root)
        self.canvas.draw()
        self.canvas.get_tk_widget().pack(expand=True, fill=tk.BOTH)

        # Close button
        close_button = tk.Button(
            self.root,
            text="Close",
            command=self.root.quit,
            font=tkfont.Font(size=12)
        )
        close_button.pack(pady=10)
```

FRUIT-COLORED PIE CHART

Main Drawing Function

C++

```
void display() {
  // Calculate percentages
  float total = 0.0f;
  for (int i = 0; i < NUM_SLICES; ++i) {
    total += values[I];
  }
  // Draw pie slices with colors
  for (int i = 0; i < NUM_SLICES; ++i){
    float sliceAngle = 360.0f * values[i] / total;
    // Draw filled slices using fruit colors
    glColor3f(colors[i][0], colors[i][1], colors[i][2]);
    glBegin(GL_TRIANGLE_FAN);
    // [Drawing code...]
    glEnd();
  }
}
```

Python

```
def create_pie_chart():
    # Create a figure and axis
    fig, ax = plt.subplots(figsize=(6, 6), dpi=100)

    # Create the pie chart
    wedges, texts, autotexts = ax.pie(
        people,
        labels=[f"{f} ({p}%)" for f, p in zip(fruits, percentages)],
        colors=fruit_colors,
        autopct='',
        startangle=90,
        textprops={'fontsize': 10}
    )

    # Equal aspect ratio ensures the pie chart is circular
    ax.axis('equal')

    # Set the title
    ax.set_title("Youth Fruit Preferences in Gachororo", pad=20)

    return fig
```

FRUIT-COLORED PIE CHART

Label Positioning Logic

C++

```
// Calculate label position
float midAngle = currentAngle + sliceAngle / 2.0f;
float midRad = midAngle * PI / 180.0f;
float labelRadius = radius + 0.15f;

// Adjust specific label positions if needed
if (i == 2) labelRadius += 0.1f;  // Banana
if (i == 3) labelRadius += 0.05f; // Kiwi

float labelX = centerX + cos(midRad) * labelRadius;
float labelY = centerY + sin(midRad) * labelRadius;
```

Python

```
# Label formatting is handled automatically
by Matplotlib in the pie() function:
labels=[f"{f} ({p}%)" for f, p in zip(fruits,
percentages)]

# This creates formatted labels like "Avocado
(21.2%)" that are
# automatically positioned around the pie
chart
```

FRUIT COLORED PIE CHART

C++ vs Python Implementation

FRUIT COLORED PIE CHART

Differences

- **Rendering Approach:**

C++: Low-level rendering using triangle fans and manual angle calculations
Python(Matplotlib): High-level API with built-in pie chart function

- **UI Framework:**

C++: GLUT (OpenGL Utility Toolkit) for window management
Python: Tkinter for window management with Matplotlib embedded

- **Text Rendering:**

C++: Custom bitmap text rendering function
Python: Automatic text rendering with formatting options

- **Label Positioning:**

C++: Manual positioning with angle calculations
Python(Matplotlib): Automatic label placement

FRUIT COLORED PIE CHART

Advantages and Disadvantages of C++(OpenGL) Implementation

Advantages

- Performance: Generally faster for real-time graphics and animations
- Fine-grained control: More precise control over drawing and positioning
- Portability: OpenGL is supported across multiple platforms
- Learning value: Demonstrates fundamental graphics principles
- Customization: Every aspect of the rendering can be customized

Disadvantages

- Complexity: Requires more lines of code for basic functionality
- Manual calculations: Must manually calculate angles, positions, and text placement
- Development time: Takes longer to implement
- Maintenance: More complex code is harder to maintain
- Lack of built-in features: No automatic label positioning or formatting

FRUIT COLORED PIE CHART

Advantages and Disadvantages of Python Implementation

Advantages

- Simplicity: Much shorter, more readable code
- High-level API: Built-in pie chart function handles most details
- Automatic formatting: Automatic label positioning and text formatting
- Extensibility: Easy to add more features like legends, annotations, etc.
- Interactive elements: Simple to add buttons and other UI controls
- Export options: Built-in support for saving charts in various formats

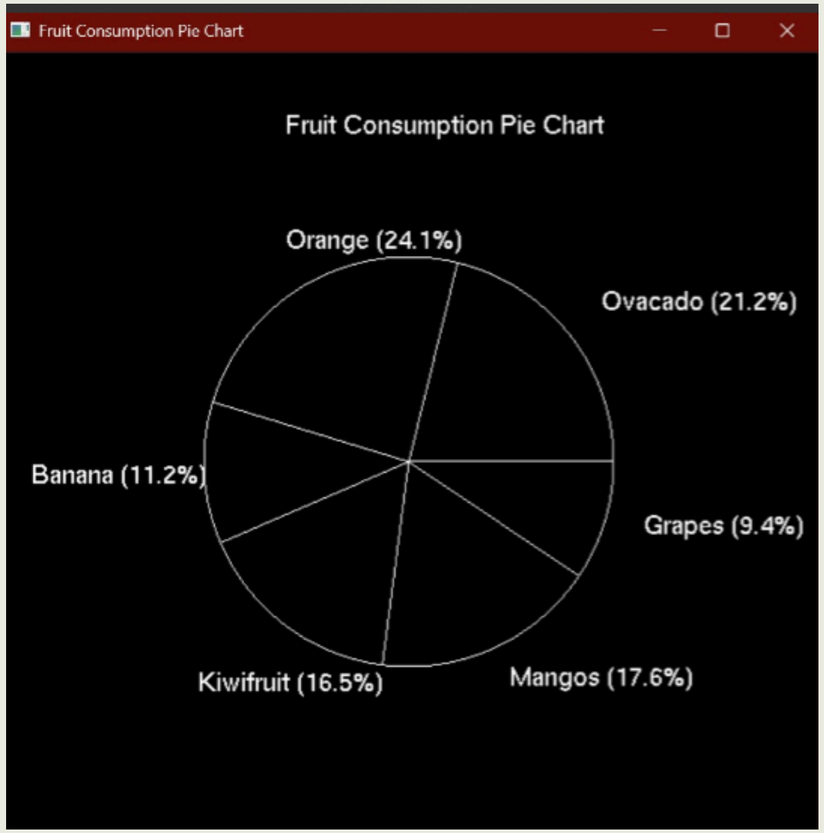
Disadvantages

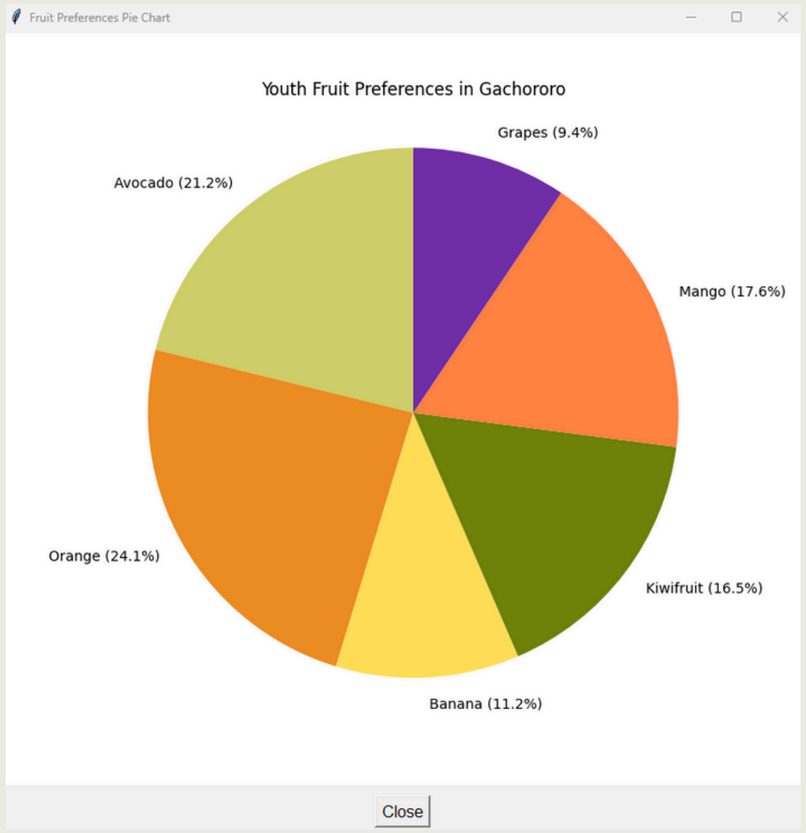
- Less control: Fewer options for customizing low-level details
- Performance: Generally slower for real-time or animated visualizations
- Dependencies: Requires multiple libraries (Matplotlib, Tkinter)
- Learning curve: Requires understanding Matplotlib's specific API
- Styling limitations: Some visual customizations may be difficult to achieve

FRUIT COLORED PIE CHART

Results & Comparison

FRUIT-COLORED PIE CHART





FRUIT-COLORED PIE CHART

Visual Changes and Additions

- Fruit-resembling colors create an intuitive association with data.
- Maintained sufficient contrast between adjacent sections.
- Adjusted label placement to prevent overlapping and improve readability.
- Added slice borders in the C++ version for better visual separation

FRUIT-COLORED PIE CHART

Challenges

Label Positioning Challenges

- Labels for adjacent small slices (Banana and Kiwifruit) were overlapping.

Solution

- Implemented custom offset adjustments for specific slices.

Color Selection Challenges

- Finding precise RGB values that closely match each fruit was difficult.
- "Some fruits (like avocado) have different colors inside and out.

Solution

- Researched RGB values representing each fruit's most recognizable color.
- Tested multiple color options to ensure sufficient contrast between adjacent sections.

PROBLEM 3 – GRAYSCALE BACKGROUND IN OPENGL

- **Objective:**
  Convert chart background to grayscale using OpenGL.
- **Approach:**

- Use luminance method to set grayscale values.
- Render the pie chart on a grayscale background.

THEORY – LUMINANCE PRESERVATION

Color Space Conversion Theory

- **Use the luminance formula to convert RGB to grayscale:**
  Gray = 0.299 * R + 0.587 * G + 0.114 * B

- This maintains perceived brightness across colors.

STEP-BY-STEP PROCESS

How Grayscale Conversion Works

- Extract RGB values of each original color.
- Apply the luminance formula to calculate the grayscale value.
- Create new color where R = G = B = Gray.

OPENGL IMPLEMENTATION OVERVIEW

Converting Colors in OpenGL

- For each color:
  float gray = 0.299 * R + 0.587 * G + 0.114 * B;
  glColor3f(gray, gray, gray);
- Use grayscale values when rendering the chart background.

GRAYSCAL CONVERSION TABLE

Fruit	(R,G,B)	Gray Calculation	Grayscale Value
Avocado	(0.34, 0.51, 0.01)	0.299×0.34 + 0.587×0.51 + 0.114×0.01	0.40
Orange	(1.0, 0.5, 0.0)	0.299×1.0 + 0.587×0.5 + 0.114×0.0	0.59
Banana	(1.0, 1.0, 0.0)	0.299×1.0 + 0.587×1.0 + 0.114×0.0	0.89
Kiwifruit	(0.45, 0.76, 0.23)	0.299×0.45 + 0.587×0.76 + 0.114×0.23	0.61
Mangos	(1.0, 0.8, 0.0)	0.299×1.0 + 0.587×0.8 + 0.114×0.0	0.77
Grapes	(0.5, 0.0, 0.5)	0.299×0.5 + 0.587×0.0 + 0.114×0.5	0.21

Code Walk Through

1. Grayscale Colors Definition

C++

```
// Grayscale colors using luminance formula: 0.299*R +
0.587*G + 0.114*B
static float grayscaleColors[][3] = {
    {0.40f, 0.40f, 0.40f},  // Avocado
    {0.59f, 0.59f, 0.59f},  // Orange
    {0.89f, 0.89f, 0.89f},  // Banana
    {0.61f, 0.61f, 0.61f},  // Kiwifruit
    {0.77f, 0.77f, 0.77f},  // Mangos
    {0.21f, 0.21f, 0.21f}   // Grapes
};
```

Python

```
# Convert colors to grayscale using luminance formula:
0.299*R + 0.587*G + 0.114*B
# Multiply by 3 to create RGB tuple (gray, gray, gray)
for matplotlib compatibility
fruit_colors = [(0.299*r + 0.587*g + 0.114*b,) * 3 for
r, g, b in original_colors]
```

2. Slice Coloring

C++

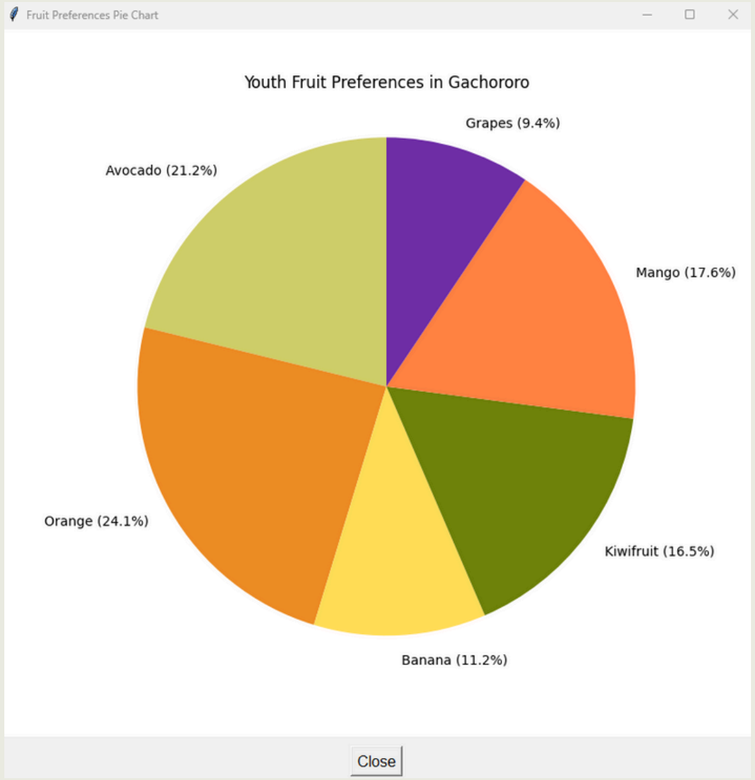
```
// Set grayscale color for this slice
glColor3fv(grayscaleColors[i]);  // Array pointeres
};
```

Python

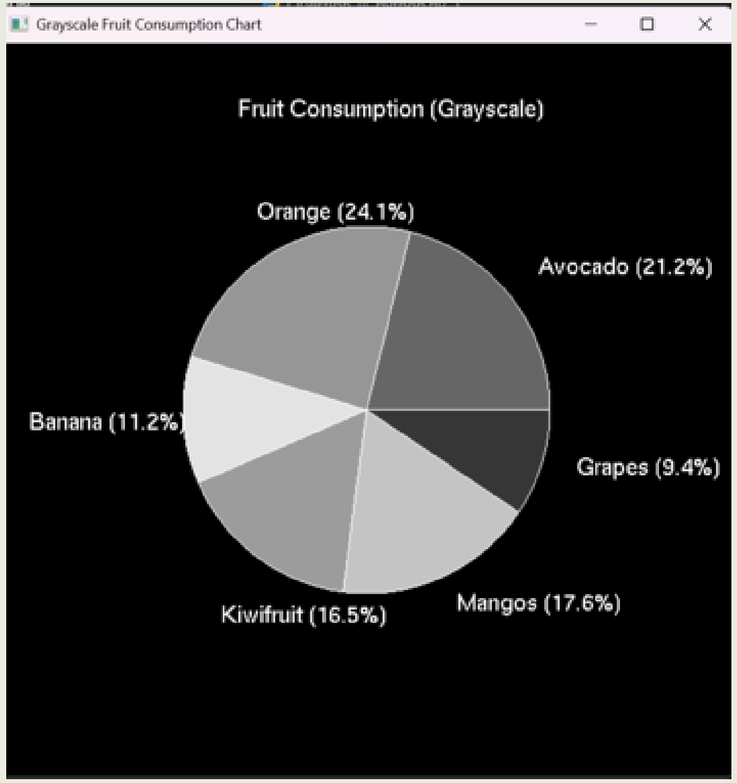
```
# Create pie chart with custom parameters
  wedges, texts = ax.pie(
    people,
    colors=fruit_colors,
.....
```

BEFORE AND AFTER VISUAL

Original colored chart background



Converted grayscale version



Takeaways:

Same Visual Output:

- Both versions produce identical grayscale rendering:
- Avocado = 0.4, Banana = 0.89, etc.

Maintenance:

- Python version is easier to modify (e.g., adding new fruits)
- C++ version offers better performance for complex visualizations

CONCLUSION

Summary of tasks achieved:

- Pie chart representation of fruit preferences with percentages and external labels.
- Color customization for realism.
- OpenGL grayscale implementation.

Q&A

Thank you.