CS 352

Computer Graphics & Visualization

Assignment - 5

Name – Anjani kumar

Roll No - 210001004

Question – 1:

Code:

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

// Structure to represent a point
struct Dot {
    int x, y;
    Dot(int a = 0, int b = 0) : x(a), y(b) {}
};

// Structure to represent a color in RGB
struct Shade {
    GLfloat red;
    GLfloat green;
    GLfloat blue;
};

// Vector to store points for drawing
vector<Dot> dots;

// Function to get the color of a pixel at given coordinates
Shade acquirePixelShade(GLint x, GLint y) {
    Shade shade;
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &shade);
```

```
    return shade;
}

// Function to set the color of a pixel at given coordinates
void adjustPixelShade(GLint x, GLint y, Shade shade) {
    glColor3f(shade.red, shade.green, shade.blue);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

// Function to check if a neighboring pixel has the same color
bool confirmDirection(GLint x, GLint y, Shade oldShade, int
total_change_in_x, int total_change_in_y) {
    Shade shade1 = acquirePixelShade(x + total_change_in_x,
y);
    Shade shade2 = acquirePixelShade(x, y +
total_change_in_y);
    return (shade1.red == oldShade.red && shade1.green ==
oldShade.green && shade1.blue == oldShade.blue) &&
           (shade2.red == oldShade.red && shade2.green ==
oldShade.green && shade2.blue == oldShade.blue);
}

// Flood fill algorithm to fill a bounded area with a new
color
void performFloodFill(GLint x, GLint y, Shade oldShade, Shade
newShade) {
    Shade shade = acquirePixelShade(x, y);

    // If the color at the given coordinates is different from
the old color, fill it with the new color
```

```
    if (shade.red != oldShade.red && shade.green !=
oldShade.green && shade.blue != oldShade.blue) {
        adjustPixelShade(x, y, newShade);

        // Recursively fill adjacent pixels with the new color
        performFloodFill(x + 1, y, oldShade, newShade);
        performFloodFill(x, y + 1, oldShade, newShade);
        performFloodFill(x - 1, y, oldShade, newShade);
        performFloodFill(x, y - 1, oldShade, newShade);

        // Check diagonal pixels and fill them if they have
the old color
        if (!confirmDirection(x, y, oldShade, 1, -1))
performFloodFill(x + 1, y - 1, oldShade, newShade);
        if (!confirmDirection(x, y, oldShade, 1, 1))
performFloodFill(x + 1, y + 1, oldShade, newShade);
        if (!confirmDirection(x, y, oldShade, -1, 1))
performFloodFill(x - 1, y + 1, oldShade, newShade);
        if (!confirmDirection(x, y, oldShade, -1, -1))
performFloodFill(x - 1, y - 1, oldShade, newShade);
    }
}

// Function to draw a circle using Bresenham's algorithm
void Circle_drawing(int x, int y, int radius) {
    int current_x = 0, current_y = radius;
    int decision_parameter = 3 - 2 * radius;
    while (current_x <= current_y) {
        dots.push_back(Dot(current_x, current_y));
        dots.push_back(Dot(current_y, current_x));
        dots.push_back(Dot(-current_x, current_y));
        dots.push_back(Dot(-current_y, current_x));
        dots.push_back(Dot(current_x, -current_y));
```

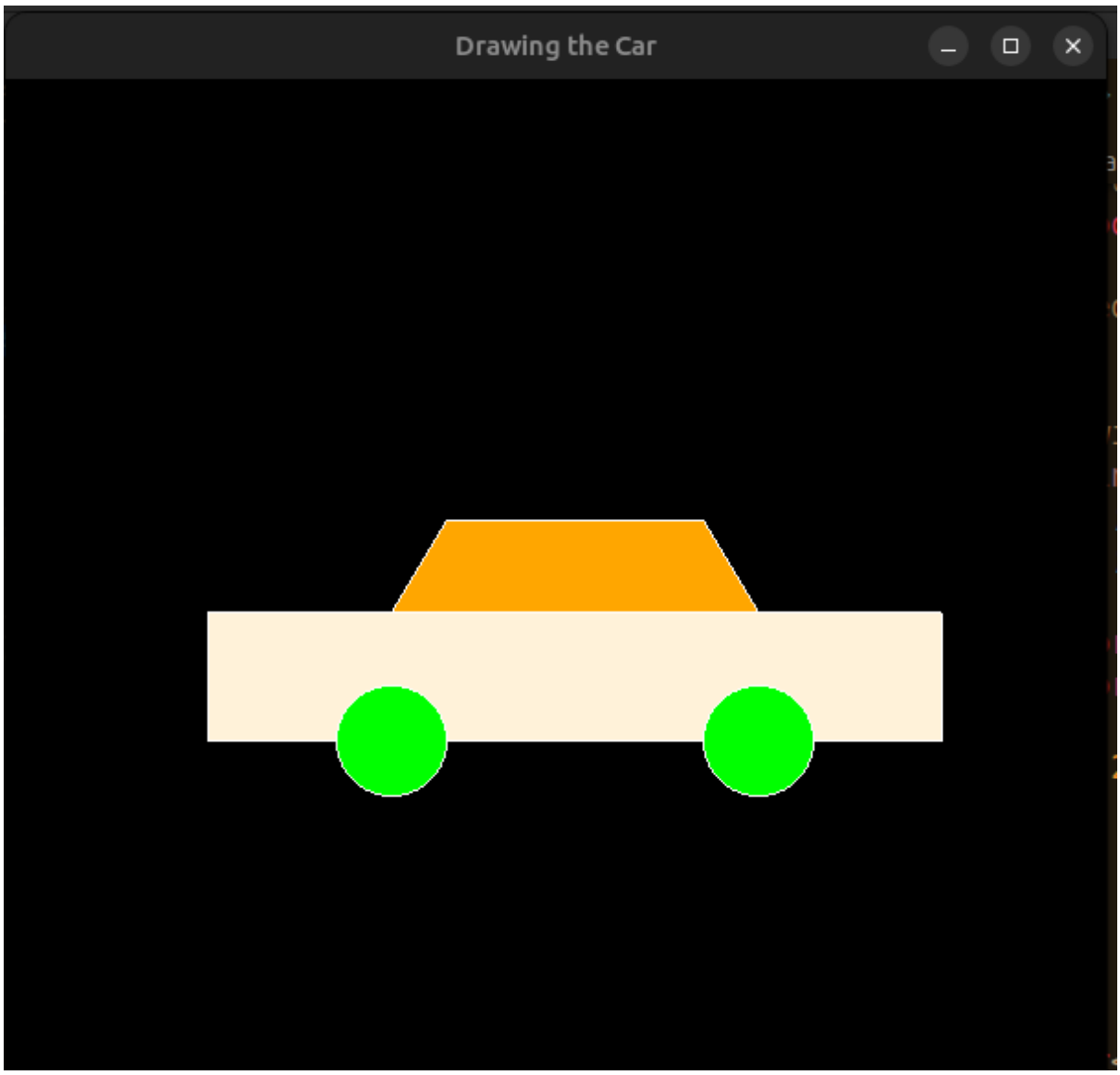
```
        dots.push_back(Dot(current_y, -current_x));
        dots.push_back(Dot(-current_x, -current_y));
        dots.push_back(Dot(-current_y, -current_x));

        if (decision_parameter < 0) {
            decision_parameter = decision_parameter + 4 *
current_x + 6;
        } else {
            decision_parameter = decision_parameter + 4 *
(current_x - current_y) + 10;
            current_y--;
        }
        current_x++;
    }

    glBegin(GL_POINTS);
    for (auto dot : dots) {
        glVertex2f(round(dot.x + x), round(dot.y + y));
    }
    glEnd();
}

// Function to draw a line using Bresenham's algorithm
void Line_drawing(int x1, int x2, int y1, int y2) {
    float total_change_in_x = x2 - x1;
    float total_change_in_y = y2 - y1;
    float increment_in_x, increment_in_y;

    if (total_change_in_x < 0) total_change_in_x = -
total_change_in_x;
    if (total_change_in_y < 0) total_change_in_y = -
total_change_in_y;
```

```
    increment_in_x = (x2 < x1) ? -1 : 1;
    increment_in_y = (y2 < y1) ? -1 : 1;

    float current_x = x1, current_y = y1;
    glBegin(GL_POINTS);

    if (total_change_in_x > total_change_in_y) {
        float decision_parameter = 2 * total_change_in_y -
total_change_in_x;
        for (int i = 0; i < total_change_in_x; i++) {
            dots.push_back(Dot(round(current_x),
round(current_y)));
            if (decision_parameter >= 0) {
                current_y += increment_in_y;
                decision_parameter += 2 * (total_change_in_y -
total_change_in_x);
            } else {
                decision_parameter += 2 * total_change_in_y;
            }
            current_x += increment_in_x;
        }
    } else {
        float decision_parameter = 2 * total_change_in_x -
total_change_in_y;
        for (int i = 0; i < total_change_in_y; i++) {
            dots.push_back(Dot(round(current_x),
round(current_y)));
            if (decision_parameter >= 0) {
                current_x += increment_in_x;
                decision_parameter += 2 * (total_change_in_x -
total_change_in_y);
            } else {
                decision_parameter += 2 * total_change_in_x;
```

```
            }
            current_y += increment_in_y;
        }
    }
    glEnd();
}

// Display callback function
void displayCB() {
    glClear(GL_COLOR_BUFFER_BIT);
    Circle_drawing(210, 240, 30);
    Circle_drawing(410, 240, 30);
    Line_drawing(110, 510, 310, 310);
    Line_drawing(110, 180, 240, 240);
    Line_drawing(240, 380, 240, 240);
    Line_drawing(440, 510, 240, 240);
    Line_drawing(510, 510, 240, 310);
    Line_drawing(110, 110, 240, 310);
    Line_drawing(210, 240, 310, 360);
    Line_drawing(240, 380, 360, 360);
    Line_drawing(380, 410, 360, 310);
    Line_drawing(410, 210, 310, 310);

    glBegin(GL_POINTS);
    for (int i = 0; i < dots.size(); i++) {
        glVertex2f(dots[i].x, dots[i].y);
    }
    glEnd();
    glFlush();

    // Define colors
```

```
    Shade newShade = {1.0f, 0.95f, 0.85f}; // Cream color
    Shade oldShade = {1.0f, 1.0f, 1.0f};   // White color
    performFloodFill(310, 280, oldShade, newShade);

    Shade newShade1 = {0.0f, 1.0f, 0.0f};  // Green color
    performFloodFill(210, 240, oldShade, newShade1);
    performFloodFill(410, 240, oldShade, newShade1);

    Shade newShade2 = {0.0f, 0.0f, 0.8f};   // Dark blue color
    performFloodFill(260, 335, oldShade, newShade2);

    glFlush();
}

// Main function
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Drawing the Car");

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 600.0, 0.0, 600.0);

    glutDisplayFunc(displayCB);
    glutMainLoop();
    return 0;
}
```

Screenshot of the Output:



Question 2:

Code:

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

// Structure to represent a point
struct Dot {
    int x, y;
    Dot(int a = 0, int b = 0) : x(a), y(b) {}
};

// Structure to represent a color in RGB
struct Shade {
    GLfloat red;
    GLfloat green;
    GLfloat blue;
};

// Vector to store points for drawing
vector<Dot> dots;

// Function to get the color of a pixel at given
coordinates
Shade acquirePixelShade(GLint x, GLint y) {
    Shade shade;
```

```
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &shade);
    return shade;
}

// Function to set the color of a pixel at given coordinates
void adjustPixelShade(GLint x, GLint y, Shade shade) {
    glColor3f(shade.red, shade.green, shade.blue);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

// Function to check if a neighboring pixel has the same color
bool confirmDirection(GLint x, GLint y, Shade oldShade, int total_change_in_x, int total_change_in_y) {
    Shade shade1 = acquirePixelShade(x + total_change_in_x, y);
    Shade shade2 = acquirePixelShade(x, y + total_change_in_y);
    return (shade1.red == oldShade.red && shade1.green == oldShade.green && shade1.blue == oldShade.blue) &&
           (shade2.red == oldShade.red && shade2.green == oldShade.green && shade2.blue == oldShade.blue);
}

// Flood fill algorithm to fill a bounded area with a new color
void performFloodFill(GLint x, GLint y, Shade oldShade, Shade newShade) {
```

```
    Shade shade = acquirePixelShade(x, y);


    // If the color at the given coordinates is different
from the old color, fill it with the new color
    if (shade.red != oldShade.red && shade.green !=
oldShade.green && shade.blue != oldShade.blue) {
        adjustPixelShade(x, y, newShade);


        // Recursively fill adjacent pixels with the new
color
        performFloodFill(x + 1, y, oldShade, newShade);
        performFloodFill(x, y + 1, oldShade, newShade);
        performFloodFill(x - 1, y, oldShade, newShade);
        performFloodFill(x, y - 1, oldShade, newShade);


        // Check diagonal pixels and fill them if they
have the old color
        if (!confirmDirection(x, y, oldShade, 1, -1))
performFloodFill(x + 1, y - 1, oldShade, newShade);
        if (!confirmDirection(x, y, oldShade, 1, 1))
performFloodFill(x + 1, y + 1, oldShade, newShade);
        if (!confirmDirection(x, y, oldShade, -1, 1))
performFloodFill(x - 1, y + 1, oldShade, newShade);
        if (!confirmDirection(x, y, oldShade, -1, -1))
performFloodFill(x - 1, y - 1, oldShade, newShade);
    }
}


// Function to draw a circle using Bresenham's algorithm
void Circle_drawing(int x, int y, int radius) {
    int current_x = 0, current_y = radius;
    int decision_parameter = 3 - 2 * radius;
```

```
    while (current_x <= current_y) {
        dots.push_back(Dot(current_x, current_y));
        dots.push_back(Dot(current_y, current_x));
        dots.push_back(Dot(-current_x, current_y));
        dots.push_back(Dot(-current_y, current_x));
        dots.push_back(Dot(current_x, -current_y));
        dots.push_back(Dot(current_y, -current_x));
        dots.push_back(Dot(-current_x, -current_y));
        dots.push_back(Dot(-current_y, -current_x));

        if (decision_parameter < 0) {
            decision_parameter = decision_parameter + 4 *
current_x + 6;
        } else {
            decision_parameter = decision_parameter + 4 *
(current_x - current_y) + 10;
            current_y--;
        }
        current_x++;
    }

    glBegin(GL_POINTS);
    for (auto dot : dots) {
        glVertex2f(round(dot.x + x), round(dot.y + y));
    }
    glEnd();
}

// Function to draw a line using Bresenham's algorithm
void Line_drawing(int x1, int x2, int y1, int y2) {
```

```
    float total_change_in_x = x2 - x1;
    float total_change_in_y = y2 - y1;
    float increment_in_x, increment_in_y;

    if (total_change_in_x < 0) total_change_in_x = -
total_change_in_x;
    if (total_change_in_y < 0) total_change_in_y = -
total_change_in_y;
    increment_in_x = (x2 < x1) ? -1 : 1;
    increment_in_y = (y2 < y1) ? -1 : 1;

    float current_x = x1, current_y = y1;
    glBegin(GL_POINTS);

    if (total_change_in_x > total_change_in_y) {
        float decision_parameter = 2 * total_change_in_y
- total_change_in_x;
        for (int i = 0; i < total_change_in_x; i++) {
            dots.push_back(Dot(round(current_x),
round(current_y)));
            if (decision_parameter >= 0) {
                current_y += increment_in_y;
                decision_parameter += 2 *
(total_change_in_y - total_change_in_x);
            } else {
                decision_parameter += 2 *
total_change_in_y;
            }
            current_x += increment_in_x;
        }
    } else {
```

```
        float decision_parameter = 2 * total_change_in_x
- total_change_in_y;
        for (int i = 0; i < total_change_in_y; i++) {
            dots.push_back(Dot(round(current_x),
round(current_y)));
            if (decision_parameter >= 0) {
                current_x += increment_in_x;
                decision_parameter += 2 *
(total_change_in_x - total_change_in_y);
            } else {
                decision_parameter += 2 *
total_change_in_x;
            }
            current_y += increment_in_y;
        }
    }
    glEnd();
}

// Display callback function
void displayCB() {
    glClear(GL_COLOR_BUFFER_BIT);
    Circle_drawing(210, 240, 30);
    Circle_drawing(410, 240, 30);
    Line_drawing(110, 510, 310, 310);
    Line_drawing(110, 180, 240, 240);
    Line_drawing(240, 380, 240, 240);
    Line_drawing(440, 510, 240, 240);
    Line_drawing(510, 510, 240, 310);
    Line_drawing(110, 110, 240, 310);
```

```
    Line_drawing(210, 240, 310, 360);
    Line_drawing(240, 380, 360, 360);
    Line_drawing(380, 410, 360, 310);
    Line_drawing(410, 210, 310, 310);

    glBegin(GL_POINTS);
    for (int i = 0; i < dots.size(); i++) {
        glVertex2f(dots[i].x, dots[i].y);
    }
    glEnd();
    glFlush();

    // Define colors
    Shade newShade = {1.0f, 0.95f, 0.85f}; // Cream color
    Shade oldShade = {1.0f, 1.0f, 1.0f};   // White color
    performFloodFill(310, 280, oldShade, newShade);

    Shade newShade1 = {0.0f, 1.0f, 0.0f};  // Green color
    performFloodFill(210, 240, oldShade, newShade1);
    performFloodFill(410, 240, oldShade, newShade1);

    Shade newShade2 = {0.0f, 0.0f, 0.8f};   // Dark blue
color
    performFloodFill(260, 335, oldShade, newShade2);

    glFlush();
}

// Main function
```

```
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Drawing the Car");

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 600.0, 0.0, 600.0);

    glutDisplayFunc(displayCB);
    glutMainLoop();
    return 0;
}
```