Universidad Nacional de San Agustín de Arequipa

Escuela Profesional de Ciencia de la Computación
Computación Gráfica

Laboratorio 06

Presentado por:
Merisabel Ruelas Quenaya

Docente :
Rosa Paccotacya Yanque

1. Implementación de funciones DDA y Bresenham

Se han implementado las funciones DDA y Bresenham para dibujar líneas. Ambas funciones han sido extendidas para crear líneas con puntos, guiones y sólidas.

1.1. Función DDA

Esta función implementa el algoritmo DDA para dibujar líneas. Toma como entrada las coordenadas de dos puntos (x1, y1) y (x2, y2), así como un parámetro de estilo. El algoritmo calcula incrementos en x e y, y utiliza estos para trazar la línea punto por punto. Dependiendo del estilo seleccionado, puede dibujar una línea sólida, punteada o con guiones.

```
void DDA(int x1, int y1, int x2, int y2, int style) {
    float dx = x2 - x1;
    float dy = y2 - y1;
    float steps = std::max(abs(dx), abs(dy));
    float xIncrement = dx / steps;
    float yIncrement = dy / steps;
    float x = x1;
    float y = y1;

    glBegin(GL_POINTS);
    for (int i = 0; i <= steps; i++) {
        if (style == 0) { // Dibujar puntos
            if (i % 2 == 0) {
                glVertex2i(round(x), round(y));
            }
        }
        else if (style == 1) { // Dibujar guiones
            if (i % 10 < 5) {
                glVertex2i(round(x), round(y));
            }
        }
        else { // Dibujar l nea s lida
            glVertex2i(round(x), round(y));
        }
        x += xIncrement;
        y += yIncrement;
    }
    glEnd();
    glFlush();
}
```

1.2. Función Bresenham

Esta función implementa el algoritmo de Bresenham para dibujar líneas. Al igual que DDA, toma las coordenadas de dos puntos y un parámetro de estilo. El algoritmo de

Bresenham utiliza solo aritmética entera, lo que lo hace más eficiente que DDA. También puede dibujar líneas sólidas, punteadas o con guiones según el estilo seleccionado.

```
void Bresenham(int x1, int y1, int x2, int y2, int style) {
    int dx = std::abs(x2 - x1);
    int dy = std::abs(y2 - y1);
    int sx = x1 < x2 ? 1 : -1;
    int sy = y1 < y2 ? 1 : -1;
    int err = dx - dy;
    int e2;

    glBegin(GL_POINTS);
    while (true) {
        if (style == 0) { // Dibujar puntos
            glVertex2i(x1, y1);
        }
        else if (style == 1) { // Dibujar guiones
            if ((x1 + y1) % 10 < 5) {
                glVertex2i(x1, y1);
            }
        }
        else { // Dibujar l nea s lida
            glVertex2i(x1, y1);
        }

        if (x1 == x2 && y1 == y2) break;
        e2 = err * 2;
        if (e2 > -dy) {
            err -= dy;
            x1 += sx;
        }
        if (e2 < dx) {
            err += dx;
            y1 += sy;
        }
    }
    glEnd();
    glFlush();
}
```

2. Estructura para dibujar una casa y un carro

Se han creado estructuras para representar una casa y un carro, junto con funciones para dibujarlos.

2.1. Función crearCasa

Esta función toma como entrada una estructura Casa y dibuja una representación gráfica de la casa utilizando las funciones DDA. Dibuja el contorno de la casa, el techo y una puerta, utilizando diferentes colores para cada parte.

2.2. Estructura y función para dibujar una casa

```
struct Casa {
    int x; // origen en x
    int y; // origen en y
    int altura;
    int ancho;
};

void crearCasa(Casa& C) {
    // Dibujar base de la casa
    glColor3f(1.0, 0.0, 0.0);
    DDA(C.x, C.y, C.x + C.ancho, C.y, 2);
    DDA(C.x + C.ancho, C.y, C.x + C.ancho, C.y + C.altura, 2);
    DDA(C.x + C.ancho, C.y + C.altura, C.x, C.y + C.altura, 2);
    DDA(C.x, C.y + C.altura, C.x, C.y, 2);

    // Dibujar techo de la casa
    glColor3f(0.0, 1.0, 0.0);
    DDA(C.x, C.y + C.altura, C.x + C.ancho / 2, C.y + C.altura + C.
   altura / 2, 2);
    DDA(C.x + C.ancho / 2, C.y + C.altura + C.altura / 2, C.x + C.ancho,
    C.y + C.altura, 2);

    // Dibujar puerta
    glColor3f(0.5, 0.3, 0.0);
    int puertaAltura = C.altura / 2;
    int puertaAncho = C.ancho / 5;
    int puertaX = C.x + C.ancho / 2 - puertaAncho / 2;
    int puertaY = C.y;
    DDA(puertaX, puertaY, puertaX + puertaAncho, puertaY, 2);
    DDA(puertaX + puertaAncho, puertaY, puertaX + puertaAncho, puertaY +
    puertaAltura , 2);
    DDA(puertaX + puertaAncho, puertaY + puertaAltura, puertaX, puertaY
   + puertaAltura, 2);
    DDA(puertaX, puertaY + puertaAltura, puertaX, puertaY, 2);
}
```

2.3. Función crearCarro

Similar a crearCasa, esta función toma una estructura Carro y dibuja una representación gráfica del carro. Utiliza las funciones DDA para dibujar el cuerpo y la cabina del carro, y la función drawCircle para dibujar las ruedas.

2.4. Estructura y función para dibujar un carro

```
struct Carro {
    int x; // origen en x
    int y; // origen en y
    int altura;
    int ancho;
```

```
};

void crearCarro(Carro& C) {
    // Dibujar cuerpo del carro
    glColor3f(0.0, 0.0, 1.0);
    DDA(C.x, C.y, C.x + C.ancho, C.y, 2);
    DDA(C.x + C.ancho, C.y, C.x + C.ancho, C.y + C.altura, 2);
    DDA(C.x + C.ancho, C.y + C.altura, C.x, C.y + C.altura, 2);
    DDA(C.x, C.y + C.altura, C.x, C.y, 2);

    // Dibujar cabina del carro
    glColor3f(1.0, 1.0, 0.0);
    int cabinaY = C.y + C.altura;
    DDA(C.x + C.ancho / 4, cabinaY, C.x + 3 * C.ancho / 4, cabinaY, 2);
    DDA(C.x + 3 * C.ancho / 4, cabinaY, C.x + 3 * C.ancho / 4, cabinaY +
    C.altura / 2, 2);
    DDA(C.x + 3 * C.ancho / 4, cabinaY + C.altura / 2, C.x + C.ancho /
   4, cabinaY + C.altura / 2, 2);
    DDA(C.x + C.ancho / 4, cabinaY + C.altura / 2, C.x + C.ancho / 4,
   cabinaY, 2);

    // Dibujar ruedas del carro
    glColor3f(0.0, 1.0, 0.0);
    int radio = C.altura / 4;
    drawCircle(C.x + C.ancho / 4, C.y - radio, radio);
    drawCircle(C.x + 3 * C.ancho / 4, C.y - radio, radio);
}
```

3. Traslación de figuras

Para trasladar las figuras a otra posición, se han implementado funciones que modifican las coordenadas de origen de las estructuras.

Estas funciones modifican las coordenadas x e y de las estructuras Casa y Carro, respectivamente. Permiten mover las figuras a una nueva posición en la pantalla.

3.1. Traslación de la casa

```
void trasladarCasa(Casa& C, int deltaX, int deltaY) {
    C.x += deltaX;
    C.y += deltaY;
}
```

3.2. Traslación del carro

```
void trasladarCarro(Carro& C, int deltaX, int deltaY) {
    C.x += deltaX;
    C.y += deltaY;
```

```
}
```

4. Escalado de figuras

Para escalar las figuras, se han implementado funciones que modifican las dimensiones de las estructuras. Estas funciones modifican las dimensiones (altura y ancho) de las estructuras Casa y Carro, multiplicándolas por un factor de escala. Esto permite aumentar o disminuir el tamaño de las figuras.

4.1. Escalado de la casa

```
void escalarCasa(Casa& C, float factor) {
    C.altura *= factor;
    C.ancho *= factor;
}
```

4.2. Escalado del carro

```
void escalarCarro(Carro& C, float factor) {
    C.altura *= factor;
    C.ancho *= factor;
}
```

5. Ejemplo de uso

Se han creado funciones para demostrar la traslación y el escalado de las figuras.

```
void dibujarCasaTrasladadaYEscalada(void) {
    Casa C;
    C.x = 100;
    C.y = 100;
    C.altura = 100;
    C.ancho = 200;
    trasladarCasa(C, 10, 20);
    escalarCasa(C, 1.5);
    crearCasa(C);
}

void dibujarCarroTrasladadoYEscalado(void) {
    Carro C;
    C.x = 200;
    C.y = 100;
    C.altura = 50;
    C.ancho = 100;
    trasladarCarro(C, 10, 20);
```

```
    escalarCarro(C, 1.5);
    crearCarro(C);
}
```

Estas funciones crean una instancia de la figura, la trasladan, la escalan y luego la dibujan, demostrando así la funcionalidad de las operaciones de traslación y escalado y finalmente la función principal del programa. Inicializa GLUT, configura la ventana de visualización, establece el sistema de coordenadas y registra la función de callback display. Finalmente, inicia el bucle principal de eventos de GLUT.

6. Resultados

6.1. Código Final

```
// Inclusi n de bibliotecas necesarias
#include <GL/glut.h>  // Para funciones de OpenGL y GLUT
#include <cmath>      // Para funciones matem ticas
#include <algorithm>  // Para std::max
#include <cstdlib>    // Para funciones est ndar de C

using namespace std;

// Funci n DDA (Digital Differential Analyzer) para dibujar l neas
void DDA(int x1, int y1, int x2, int y2, int style) {
    // C lculo de diferencias y pasos
    float dx = x2 - x1;
    float dy = y2 - y1;
    float steps = std::max(abs(dx), abs(dy));

    // C lculo de incrementos
    float xIncrement = dx / steps;
    float yIncrement = dy / steps;

    // Inicializaci n de coordenadas
    float x = x1;
    float y = y1;

    glBegin(GL_POINTS);
    for (int i = 0; i <= steps; i++) {
        // Dibujo seg n el estilo especificado
        if (style == 0) {  // Estilo de puntos
            if (i % 2 == 0) {
                glVertex2i(round(x), round(y));
            }
        }
        else if (style == 1) {  // Estilo de guiones
            if (i % 10 < 5) {
                glVertex2i(round(x), round(y));
            }
        }
        else {  // L nea s lida
            glVertex2i(round(x), round(y));
```

```
        }
        // Actualizaci n de coordenadas
        x += xIncrement;
        y += yIncrement;
    }
    glEnd();
    glFlush();
}

// Funci n de Bresenham para dibujar l neas
void Bresenham(int x1, int y1, int x2, int y2, int style) {
    // C lculo de diferencias y direcciones
    int dx = std::abs(x2 - x1);
    int dy = std::abs(y2 - y1);
    int sx = x1 < x2 ? 1 : -1;
    int sy = y1 < y2 ? 1 : -1;

    // Inicializaci n de error
    int err = dx - dy;
    int e2;

    glBegin(GL_POINTS);
    while (true) {
        // Dibujo seg n el estilo especificado
        if (style == 0) {  // Estilo de puntos
            glVertex2i(x1, y1);
        }
        else if (style == 1) {  // Estilo de guiones
            if ((x1 + y1) % 10 < 5) {
                glVertex2i(x1, y1);
            }
        }
        else {  // L nea s lida
            glVertex2i(x1, y1);
        }

        // Condici n de salida
        if (x1 == x2 && y1 == y2) break;

        // Actualizaci n de error y coordenadas
        e2 = err * 2;
        if (e2 > -dy) {
            err -= dy;
            x1 += sx;
        }
        if (e2 < dx) {
            err += dx;
            y1 += sy;
        }
    }
    glEnd();
    glFlush();
}

// Definici n de la estructura Casa
struct Casa {
    int x;      // Coordenada x del origen
```

```
    int y;      // Coordenada y del origen
    int altura; // Altura de la casa
    int ancho;  // Ancho de la casa
};

// Funci n para crear y dibujar una casa
void crearCasa(Casa& C) {
    // Dibujo de la base de la casa
    glColor3f(1.0, 0.0, 0.0);  // Color rojo
    DDA(C.x, C.y, C.x + C.ancho, C.y, 2);
    DDA(C.x + C.ancho, C.y, C.x + C.ancho, C.y + C.altura, 2);
    DDA(C.x + C.ancho, C.y + C.altura, C.x, C.y + C.altura, 2);
    DDA(C.x, C.y + C.altura, C.x, C.y, 2);

    // Dibujo del techo
    glColor3f(0.0, 1.0, 0.0);  // Color verde
    DDA(C.x, C.y + C.altura, C.x + C.ancho / 2, C.y + C.altura + C.
   altura / 2, 2);
    DDA(C.x + C.ancho / 2, C.y + C.altura + C.altura / 2, C.x + C.ancho,
    C.y + C.altura, 2);

    // Dibujo de la puerta
    glColor3f(0.5, 0.3, 0.0);  // Color marr n
    int puertaAltura = C.altura / 2;
    int puertaAncho = C.ancho / 5;
    int puertaX = C.x + C.ancho / 2 - puertaAncho / 2;
    int puertaY = C.y;
    DDA(puertaX, puertaY, puertaX + puertaAncho, puertaY, 2);
    DDA(puertaX + puertaAncho, puertaY, puertaX + puertaAncho, puertaY +
    puertaAltura, 2);
    DDA(puertaX + puertaAncho, puertaY + puertaAltura, puertaX, puertaY
   + puertaAltura, 2);
    DDA(puertaX, puertaY + puertaAltura, puertaX, puertaY, 2);
}

// Funci n para dibujar una casa con dimensiones predefinidas
void dibujarCasa(void) {
    Casa C;
    C.x = 50;
    C.y = 50;
    C.altura = 100;
    C.ancho = 200;
    crearCasa(C);
}

// Funci n para dibujar un c rculo
void drawCircle(int xc, int yc, int radius) {
    glBegin(GL_POINTS);
    for (int angle = 0; angle <= 360; angle++) {
        float theta = angle * 3.14159 / 180;
        int x = xc + radius * cos(theta);
        int y = yc + radius * sin(theta);
        glVertex2i(x, y);
    }
    glEnd();
}
```

```
// Definici n de la estructura Carro
struct Carro {
    int x;      // Coordenada x del origen
    int y;      // Coordenada y del origen
    int altura; // Altura del carro
    int ancho;  // Ancho del carro
};

// Funci n para crear y dibujar un carro
void crearCarro(Carro& C) {
    // Dibujo del cuerpo del carro
    glColor3f(0.0, 0.0, 1.0);  // Color azul
    DDA(C.x, C.y, C.x + C.ancho, C.y, 2);
    DDA(C.x + C.ancho, C.y, C.x + C.ancho, C.y + C.altura, 2);
    DDA(C.x + C.ancho, C.y + C.altura, C.x, C.y + C.altura, 2);
    DDA(C.x, C.y + C.altura, C.x, C.y, 2);

    // Dibujo de la cabina del carro
    glColor3f(1.0, 1.0, 0.0);  // Color amarillo
    int cabinaY = C.y + C.altura;
    DDA(C.x + C.ancho / 4, cabinaY, C.x + 3 * C.ancho / 4, cabinaY, 2);
    DDA(C.x + 3 * C.ancho / 4, cabinaY, C.x + 3 * C.ancho / 4, cabinaY +
    C.altura / 2, 2);
    DDA(C.x + 3 * C.ancho / 4, cabinaY + C.altura / 2, C.x + C.ancho /
   4, cabinaY + C.altura / 2, 2);
    DDA(C.x + C.ancho / 4, cabinaY + C.altura / 2, C.x + C.ancho / 4,
   cabinaY, 2);

    // Dibujo de las ruedas del carro
    glColor3f(0.0, 1.0, 0.0);  // Color verde
    int radio = C.altura / 4;
    drawCircle(C.x + C.ancho / 4, C.y - radio, radio);
    drawCircle(C.x + 3 * C.ancho / 4, C.y - radio, radio);
}

// Funci n para dibujar un carro con dimensiones predefinidas
void dibujarCarro(void) {
    Carro C;
    C.x = 300;
    C.y = 50;
    C.altura = 50;
    C.ancho = 100;
    crearCarro(C);
}

// Funci n para trasladar una casa
void trasladarCasa(Casa& C, int deltaX, int deltaY) {
    C.x += deltaX;
    C.y += deltaY;

    // Limitar la posici n dentro de la ventana
    if (C.x < 0) C.x = 0;
    if (C.y < 0) C.y = 0;
    if (C.x + C.ancho > 500) C.x = 500 - C.ancho;
    if (C.y + C.altura > 500) C.y = 500 - C.altura;
}
```

```
// Funci n para escalar una casa
void escalarCasa(Casa& C, float factor) {
    int nuevaAltura = C.altura * factor;
    int nuevoAncho = C.ancho * factor;

    // Limitar el tama o dentro de la ventana
    if (C.x + nuevoAncho > 500) {
        nuevoAncho = 500 - C.x;
        C.altura = nuevoAncho / factor;
    }
    if (C.y + nuevaAltura > 500) {
        nuevaAltura = 500 - C.y;
        C.ancho = nuevaAltura / factor;
    }

    C.altura = nuevaAltura;
    C.ancho = nuevoAncho;
}

// Funci n para trasladar un carro
void trasladarCarro(Carro& C, int deltaX, int deltaY) {
    C.x += deltaX;
    C.y += deltaY;

    // Limitar la posici n dentro de la ventana
    if (C.x < 0) C.x = 0;
    if (C.y < 0) C.y = 0;
    if (C.x + C.ancho > 500) C.x = 500 - C.ancho;
    if (C.y + C.altura > 500) C.y = 500 - C.altura;
}

// Funci n para escalar un carro
void escalarCarro(Carro& C, float factor) {
    int nuevaAltura = C.altura * factor;
    int nuevoAncho = C.ancho * factor;

    // Limitar el tama o dentro de la ventana
    if (C.x + nuevoAncho > 500) {
        nuevoAncho = 500 - C.x;
        C.altura = nuevoAncho / factor;
    }
    if (C.y + nuevaAltura > 500) {
        nuevaAltura = 500 - C.y;
        C.ancho = nuevaAltura / factor;
    }

    C.altura = nuevaAltura;
    C.ancho = nuevoAncho;
}

// Funci n para dibujar una casa trasladada y escalada
void dibujarCasaTrasladadaYEscalada(void) {
    Casa C;
    C.x = 50;
    C.y = 300;
    C.altura = 150;
    C.ancho = 300;
```

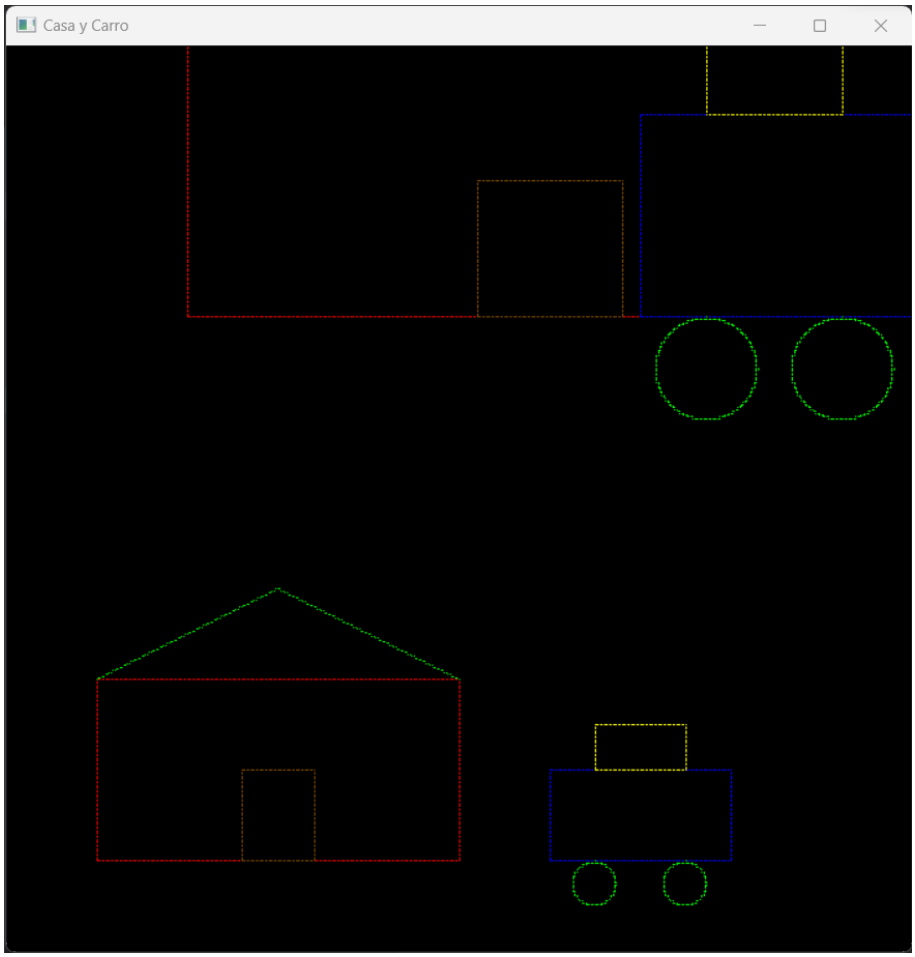
```
    trasladarCasa(C, 50, 50);
    escalarCasa(C, 1.5);
    crearCasa(C);
}

// Funci n para dibujar un carro trasladado y escalado
void dibujarCarroTrasladadoYEscalado(void) {
    Carro C;
    C.x = 300;
    C.y = 300;
    C.altura = 75;
    C.ancho = 150;
    trasladarCarro(C, 50, 50);
    escalarCarro(C, 1.5);
    crearCarro(C);
}

// Funci n principal de visualizaci n
void display() {
    glClear(GL_COLOR_BUFFER_BIT);  // Limpia el buffer de color
    dibujarCasa();
    dibujarCarro();
    dibujarCasaTrasladadaYEscalada();
    dibujarCarroTrasladadoYEscalado();
    glFlush();  // Fuerza el dibujado de todos los comandos OpenGL
   pendientes
}

// Funci n principal
int main(int argc, char** argv) {
    glutInit(&argc, argv);  // Inicializa GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);  // Configura el modo
   de visualizaci n
    glutInitWindowSize(700, 700);  // Establece el tama o de la ventana
    glutCreateWindow("Casa y Carro");  // Crea la ventana con un t tulo
    gluOrtho2D(0, 500, 0, 500);  // Establece el sistema de coordenadas
    glutDisplayFunc(display);  // Registra la funci n de visualizaci n
    glutMainLoop();  // Inicia el bucle principal de GLUT
    return 0;
}
```

6.2. Imagen


Casa y Carro