Rasterização de linhas

Matheus Fernandes de Sousa
Lucas Freitas de Barros
José Gabriel de Sousa Silva
Centro de Informática
Universidade Federal da Paraíba
João Pessoa, Brasil

I. Resumo

Neste trabalho, foram utilizados algoritmos de rasterização de pontos e linhas, bem como desenhados triângulos com o uso dos mesmos. Foram utilizados os cabeçalhos do OpenGL e GLUT, conforme visto em sala de aula.

II. Objetivos

A. Objetivos Gerais

Compreender o funcionamento do algoritmo de Bresenham, bem como desenhar formas geométricas básicas.

B. Objetivos Específicos

Implementar, utilizando o algoritmo de Bresenham, formas geométricas simples e entender o funciomento e configuração do OpenGL.

III. Introdução Teórica

A computação gráfica é uma área da computação que lida com a geração, manipulação e exibição de imagens e animações por meio de computadores. A computação gráfica tem muitas aplicações, desde a criação de gráficos para jogos e filmes até a simulação de sistemas físicos em ambientes virtuais.

OpenGL é uma API (Application Programming Interface) de gráficos 3D que permite que os desenvolvedores de software criem aplicativos gráficos avançados. Foi desenvolvido pela Silicon Graphics Inc. (SGI) em 1992 e é amplamente utilizado na indústria de jogos, animação, simulação, visualização científica, entre outras.

A rasterização de linhas é um processo utilizado em gráficos por computador para converter informações de geometria, como segmentos de linha, em pixels individuais que podem ser exibidos em uma tela ou monitor.

O processo de rasterização de linhas envolve o cálculo de pixels que correspondem à linha ou curva desejada com base em sua equação matemática. Em seguida, esses pixels são coloridos para criar a aparência final da linha ou curva.

A rasterização de linhas é um dos principais processos utilizados em gráficos por computador, e é uma parte fundamental da renderização de imagens em tempo real em jogos de computador, aplicativos de modelagem 3D, visualização científica e muitas outras áreas.

O algoritmo de Bresenham é um algoritmo de rasterização que desenha linhas em uma grade de pixels. A ideia por trás do algoritmo é traçar uma linha entre dois pontos, considerando que cada pixel tem uma coordenada inteira. Isso significa que o algoritmo precisa determinar quais pixels desenhar ao longo da linha para criar a aparência contínua da linha.

O algoritmo de Bresenham utiliza apenas operações de adição, subtração e comparação para determinar quais pixels desenhar ao longo da linha. Isso torna o algoritmo eficiente em termos de tempo de execução e uso de memória.

O algoritmo começa determinando em qual direção a linha está se movendo em relação ao eixo x e y. Com base nessa direção, o algoritmo escolhe incrementos de coordenadas para o próximo pixel a ser desenhado. O algoritmo então calcula um valor de erro para determinar se o próximo pixel deve estar na mesma linha ou na linha seguinte. O processo é repetido até que a linha esteja completamente desenhada.

O algoritmo de Bresenham também pode ser usado para desenhar círculos. Nesse caso, o algoritmo é utilizado para desenhar uma série de arcos, cada um dos quais representa um segmento do círculo. O resultado final é uma imagem de círculo suave e preciso.

Em resumo, o algoritmo de Bresenham é um algoritmo de rasterização eficiente e amplamente utilizado em sistemas de computação gráfica para desenhar linhas e círculos em telas de computador e impressoras.

Para exemplificar, podemos mostrar um código simples, capaz de desenhar um círculo:

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>

void display();

int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(400, 400);
  glutCreateWindow("Bresenham␣Circle␣Drawing␣
      Algorithm");
  glClearColor(1.0, 1.0, 1.0, 0);
  gluOrtho2D(-200, 200, -200, 200);
  glutDisplayFunc(display);
  glutMainLoop();
}
```

```
void display() {
  int r = 100, x = 0, y = r, d = 3 - 2 * r;
  glColor3f(0.0, 0.0, 0.0);
  glBegin(GL_POINTS);

  while (x <= y) {
    glVertex2f(x, y);
    glVertex2f(-x, y);
    glVertex2f(x, -y);
    glVertex2f(-x, -y);
    glVertex2f(y, x);
    glVertex2f(-y, x);
    glVertex2f(y, -x);
    glVertex2f(-y, -x);
    if (d < 0) {
      d = d + 4 * x + 6;
    } else {
      d = d + 4 * (x - y) + 10;
      y--;
    }
    x++;
  }
  glEnd();
  glFlush();
}
```

O programa começa inicializando o ambiente OpenGL, definindo o modo de exibição como GLUT_SINGLE—GLUT_RGB, o tamanho da janela e o título. A função de exibição display é definida e registada, juntamente com o loop principal do programa que espera por eventos.

A função display define as variáveis r, x, y e d, que são usadas para desenhar o círculo. A cor é definida como preta usando glColor3f. A função glBegin inicia a definição de um conjunto de pontos para o círculo e a função glEnd termina. Um laço while é utilizado para calcular os pontos necessários para desenhar o círculo, usando o algoritmo de Bresenham. Dentro do laço while, a função glVertex2f é usada para definir os pontos, que são calculados usando as coordenadas x e y. Depois que todos os pontos são calculados, a função glEnd encerra a definição do conjunto de pontos e a função glFlush atualiza a janela com a imagem desenhada.

IV. Metodologia

Para realizar a presente atividade foi necessário o cumprimento de uma sequência de etapas.

Para começar, o ambiente precisou ser configurado. O Sistema Operacional escolhido foi o GNU/Linux, utilizando como editor de texto o GNU nano. Para a instalação e configuração do OpenGL foram feitos os seguintes passos: A instalação do OpenGL Utility Toolkit (GLUT), do pacote ”Miscellaneous Mesa GL utilities”, das bibliotecas ”X11 miscellaneous utility library” e ”X11 Input extension library” e de um compilador C/C++, neste caso o GNU G++.

Como requisitado, a documentação é também um fator importante. Para tal o Overleaf foi utilizado para documentar as atividades desenvolvidas de maneira formal para análise do professor. Além disso, todo o código foi upado no GitHub, que pode ser acessado em https://github.com/jgss-gabriel-sousa/ICG-Atividade-Rasterizando-Linhas.

V. Discussões

Foram criadas três funções para a realização da atividade, sendo elas: PutPixel() que rasteriza um ponto na tela por meio das coordenadas (x,y), a função DrawLine() responsável por rasterizar uma linha na tela, onde recebe os paramêtros das coordenadas dos vértices inicial e final, bem como a função DrawTriangle(), que desenha as arestas de um triângulo na tela, onde recebe como parâmetros as posições dos três vértices (x0,y0), (x1,y1) e (x2,y2), recebendo também as cores de cada um dos vértices. Observação: o algoritmo implementado é o de Bresenham.

Para análise, o código pode ser expresso da seguinte forma:

```
#include "definitions.h"
#include <iostream>
#include <cmath>
#include <stdio.h>

using namespace std;


void MyGlDraw(void){
    DrawLine(Point(0,0), Point(255,255), RED);
    DrawTriangle(Point(350,0), Point(250,125), Point
        (350,100), GREEN);
}

void PutPixel(Point p, Color c) {
    int pixelIndex = (p.x + p.y * IMAGE_WIDTH) * 4;

    FBptr[pixelIndex] = c.red;
    FBptr[pixelIndex + 1] = c.green;
    FBptr[pixelIndex + 2] = c.blue;
    FBptr[pixelIndex + 3] = c.alpha;
}

void DrawLine(Point p0, Point p1, Color color)
{
    int x0 = p0.x;
    int y0 = p0.y;
    int x1 = p1.x;
    int y1 = p1.y;

    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int sx = x0 < x1 ? 1 : -1;
    int sy = y0 < y1 ? 1 : -1;
    int error = dx - dy;
    int x = x0;
    int y = y0;

    while (x != x1 || y != y1) {

        PutPixel(Point(x, y), color);

        int newError = 2 * error;
        if (newError > -dy) {
            error -= dy;
            x += sx;
        }
        if (newError < dx) {
            error += dx;
            y += sy;
        }
    }
}

void DrawTriangle(Point p0, Point p1, Point p2,
    Color color)
{
```

```
        DrawLine(p0, p1, color);
        DrawLine(p1, p2, color);
        DrawLine(p2, p0, color);
}
```

Esse código é uma implementação básica de uma biblioteca de desenho 2D, que utiliza a técnica de rasterização para desenhar primitivas gráficas, como linhas e triângulos, em uma matriz de pixels que representa uma imagem. O programa principal chama duas funções da biblioteca, ”DrawLine” e ”DrawTriangle”, que recebem os pontos e a cor de cada primitiva a ser desenhada. A função ”PutPixel” é responsável por escrever a cor do pixel correto na matriz de pixels. A função ”DrawLine” utiliza o algoritmo de Bresenham para desenhar uma linha, dado os pontos iniciais e finais, enquanto ”DrawTriangle” chama a função ”DrawLine” três vezes para desenhar as três arestas do triângulo. A biblioteca foi construída usando a linguagem C++.

O código, entretanto, utiliza as classes Point, Color e Pixel, que foram também criadas com o intuito de definir as características de cada ponto desenhado. Elas são definidas a partir do seguinte arquivo de cabeçalho:

```
#ifndef _MYGL_H_
#define _MYGL_H_
#endif

#include "definitions.h"


void MyGlDraw(void);

class Point{
public:
        int x;
        int y;

        Point(){}

        Point(int x, int y){
                this->x = x;
                this->y = y;
        }
};

class Color{
public:
        int red;
        int green;
        int blue;
        int alpha;

        Color(){}

        Color(int r, int g, int b, int a){
                red = r;
                green = g;
                blue = b;
                alpha = a;
        }
};

class Pixel{
public:
        Point p;
        Color c;

        Pixel(Point p, Color c){
                this->p = p;
                this->c = c;
        }
};

Color RED = Color(255,0,0,255);
Color GREEN = Color(0,255,0,255);
Color BLUE = Color(0,0,255,255);

void PutPixel(Point, Color);
void DrawLine(Point, Point, Color);
void DrawTriangle(Point, Point, Point, Color);
```

A classe Point define um ponto com coordenadas x e y. A classe Color define uma cor com os valores de canal de cor vermelho, verde, azul e alfa (transparência). A classe Pixel é uma classe composta que possui um ponto e uma cor.

As constantes RED, GREEN e BLUE são instâncias de Color que representam as cores vermelha, verde e azul, respectivamente.

As funções PutPixel, DrawLine e DrawTriangle são responsáveis por desenhar pontos, linhas e triângulos, respectivamente. A função MyGlDraw é uma função vazia que deve ser implementada pelo usuário para desenhar o que desejar.

Por fim, as definições estão protegidas por uma diretiva #ifndef que garante que o arquivo de cabeçalho será incluído apenas uma vez em cada arquivo fonte que o inclui, evitando erros de duplicação.

VI. Conclusão

A partir do presente trabalho foi possível observar de forma prática o funcionamento do OpenGL, bem como entender o desenvolvimento do algoritmo de Bresenham.

Através dele foi possível desenhar formas geométricas simples, traçar linhas, ligar pontos. Pode-se dizer, portanto, que os objetivos foram atingidos.

Abaixo pode-se observar o resultado da rasterização.

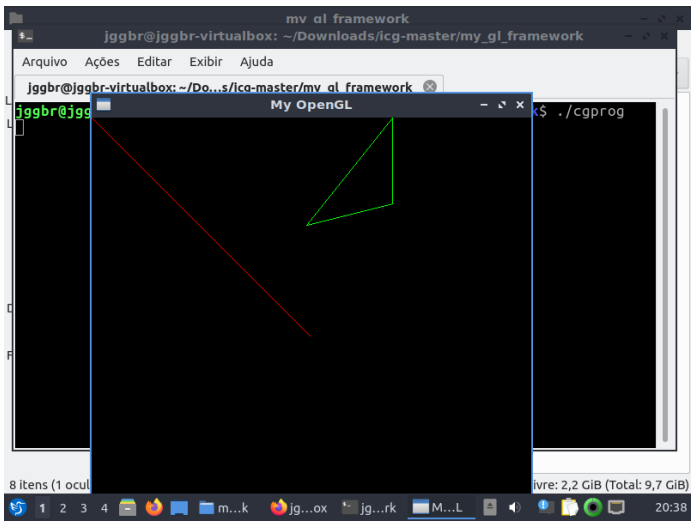

Fig. 1. Resultado da rasterização

VII. Referências

FOLEY, James D; DAM, Andries Van; FEINER, Steven K. Computer graphics:principles and practice. 2.ed.. Boston: Addison-Wesley, 1996. 1175 p. ISBN: 0201848406.

CONCI, Aura; AZEVEDO, Eduardo; LETA, Fabiana R. Computação gráfica:teoria e prática. Rio de Janeiro: Elsevier, 2008. v. ISBN: 8535223293, 9788535223293.

WOO, Mason. OpenGL:programming guide. 3.ed. Boston: Addison-Wesley, c1999. 730 p. ISBN: 0201604582.

BUSS, Samuel R. 3-D Computer Graphics:A Mathematical Introduction with OpenGL. Cambridge: Cambridge University, 2003, 2005. 371 p. ISBN: 0521821037.