МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний Авіаційний Університет

Факультет комп’ютерних наук та технологій

Лабораторна робота № 2

На тему: «Відсікання прямолінійних відрізків»

Дисципліна: «Обчислювальна геометрія та комп´ютерна графіка»

Студента групи 351:

О.Т.Архіпова

Керівник:

асистент кафедри

прикладної математики

А. В. Темніков

Оцінка:

Київ 2023

**Зміст**

**Вступ………………………………………………………………………………………………………………..3**

**Постановка задачі……………………………………………………………………………………………4**

**Розробка програмного забезпечення…………………………………………………………….5**

**Висновок………………………………………………………………………………………………………….8**

**Додаток……………………………………………………………………………………………………………9**

**Вступ**

Формування зображення та різноманітні дії з ним вимагають від користувача відомої математичної грамотності. Геометричні поняття, формули та факти, що відносяться до плоского та тривимірного випадків, грають у завданнях комп'ютерної графіки особливу роль. Принципи аналітичної геометрії у поєднанні з можливостями обчислювальної техніки, що постійно розширюються, є невичерпним джерелом істотних поступів на шляху розвитку комп'ютерної графіки.

На сьогодні комп’ютерна графіка – це наукова область, що має безліч застосувань. Вона широко застосовується в різних сферах діяльності людини: будівництві та архітектурі, промисловості, в комп’ютерних іграх та кіноіндустрії. Крім того графіка знайшла своє місце і в медицині, астрономії, картографії, фотограмметрії і в багатьох інших областях знання.

**Постановка задачі**

Актуальність: широко застосовується у моделюванні та комп’ютерній графіці.

Мета дослідження: потрібно реалізувати алгоритм відсікання підвищеної ефективності, розроблений на основі вивчених класичних алгоритмів.

Об’єкт дослідження: відрізки на графічній площині, область видимості.

Предмет дослідження: алгоритми відсікання прямолінійних відрізків.

Постановка задачі: потрібно реалізувати відсікання прямолінійних відрізків за допомогою відрізків на графічній площині та області видимості.

**Розробка програмного забезпечення**

У комп'ютерній графіці часто доводиться вирішувати завдання виділення деякої області зображуваної сцени, причому завдання це може вирішуватися як у застосуванні до плоскої області (якщо сцена вже спроектована на картинну площину), так і до тривимірної. Алгоритми відсікання застосовуються для видалення невидимих поверхонь і ліній, для побудови тіней, при формуванні текстур. Відсікається область може бути як правильної форми (прямокутник або паралелепіпед зі сторонами, паралельними осям координат або координатним площинам), так і неправильної (довільний багатокутник або багатогранник). Для того щоб ці алгоритми можна було використовувати в задачах зображення динамічних сцен, вони повинні бути ефективними у відношенні часу обчислень.

У своїй роботі я використав суміш декількох алгоритмів, тож я напишу тільки те, що я брав з я кого алгоритму та для яких відрізків.

1. Для вирішення проблем відрізків подібних, як на рис 1

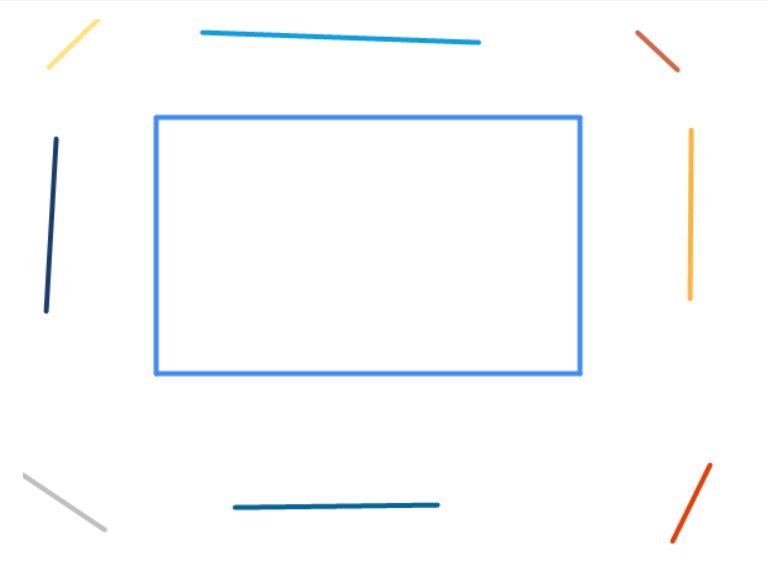


Рис 1

я обрав алгоритм Сазерленда-Коена.

Наш прямокутник заданий умовами 

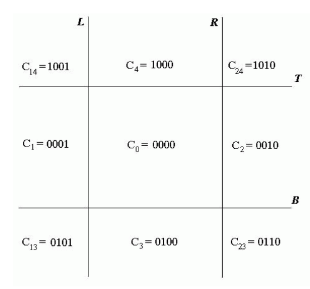
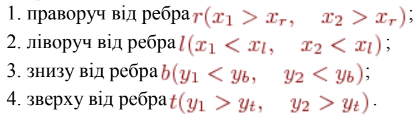


Рис 2

Нехай кінці відрізка задані точками (х1,у1) і (х2,у2). Перший крок алгоритму націлений на те, щоб виявити повністю видимі і повністю невидимі відрізки.

Відрізок повністю невидимий, якщо обидва його кінця лежать



За допомогою цього на рис 1 залишиться тільки прямокутник.

1. Для прямих подібних рис 3 я додав додаткову умову до попередніх.



Рис 3

Якщо x1>xl && x1 < xr && x2>xl && x2 < xr && y1>xb && y1 < xt && y2>xb && y2 < xt.

Використавши дану умову подібні відрізки залишаться.

1. Для вирішення проблем відрізків подібних, як на рис 4.

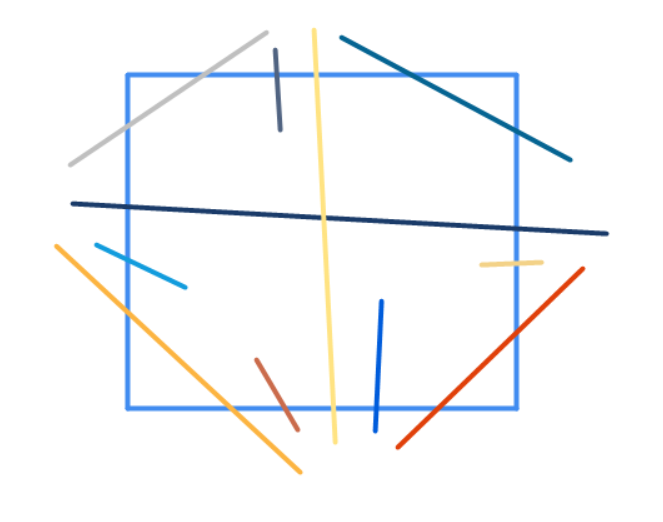


Рис 4

Пройшовши всі попередні перевірки, або відрізків не залишиться, або залишаться подібні відрізки.

Для вирішення подібних відрізків я за допомогою алгоритма Ляна-Барського знаходжу перетин між двома прямими і знаходжу точку перетину.

Я одну пряму кодую за допомогою перевірки перетину її зі стороною прямокутника, якщо перетин є то я повертаю 0, якщо його немає то повертаю. Поєднання двох алгоритмів: Ляна-Барського та Сазерленда-Коена дозволяє мені дізнатися які сторони прямокутника пряма перетинає, які ні.

Якщо вона перетинає дві сторони прямокутника, то ці точки просто з’єдную, якщо є тільки один претин то я з’єдную точку перетину з точкою, яка лежить у цьому прямокутнику.

За допомогою такого алгоритму криві на рис 4 перетворюються на прямі на рис 5.



Рис 5

**Висновок**

1. Для реалізації відсікання прямолінійних відрізків за допомогою відрізків на графічній площині та області видимості я використав алгоритми Ляна-Барського та Сазерленда-Коена.
2. Для реалізації алгоритмів Ляна-Барського та Сазерленда-Коена я використав точки прямих та вершини прямокутника.
3. Використав мову с# тому, що вона гарно підходить для візуалізації та зображень.

**Додаток**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

namespace lab2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

chart1.Legends.Clear();

chart1.ChartAreas[0].AxisX.Enabled = AxisEnabled.False;

chart1.ChartAreas[0].AxisY.Enabled = AxisEnabled.False;

chart1.Series[0].ToolTip = "X = #VALX, Y = #VALY";

chart1.ChartAreas[0].AxisX.Minimum = 0;

chart1.ChartAreas[0].AxisX.Maximum = 30;

chart1.ChartAreas[0].AxisY.Minimum = 0;

chart1.ChartAreas[0].AxisY.Maximum = 30;

radioButton1.Checked = true;

}

List<double> X = new List<double>();

List<double> Y = new List<double>();

List<double> X2 = new List<double>();

List<double> Y2 = new List<double>();

int c = 0;

private void chart1\_MouseClick(object sender, MouseEventArgs e)

{

if (radioButton1.Checked)

{

Series s;

X.Add(chart1.ChartAreas[0].AxisX.PixelPositionToValue(e.X));

Y.Add(chart1.ChartAreas[0].AxisY.PixelPositionToValue(e.Y));

if (c == 1)

{

s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(X[X.Count - 2], Y[Y.Count - 2]);

s.Points.AddXY(X[X.Count - 1], Y[Y.Count - 1]);

chart1.Series.Add(s);

c = -1;

}

c++;

}

else if (radioButton2.Checked)

{

X2.Add(chart1.ChartAreas[0].AxisX.PixelPositionToValue(e.X));

Y2.Add(chart1.ChartAreas[0].AxisY.PixelPositionToValue(e.Y));

if (c == 1)

{

chart1.Series[0].BorderWidth = 4;

chart1.Series[0].Points.AddXY(X2[X2.Count - 2], Y2[Y2.Count - 2]);

chart1.Series[0].Points.AddXY(X2[X2.Count - 2], Y2[Y2.Count - 1]);

chart1.Series[0].Points.AddXY(X2[X2.Count - 1], Y2[Y2.Count - 1]);

chart1.Series[0].Points.AddXY(X2[X2.Count - 1], Y2[Y2.Count - 2]);

chart1.Series[0].Points.AddXY(X2[X2.Count - 2], Y2[Y2.Count - 2]);

c = -1;

}

c++;

}

}

void R()

{

Random r = new Random();

int x = r.Next(0, 30);

int y = r.Next(0, 30);

int x2 = r.Next(0, 30);

int y2 = r.Next(0, 30);

X2.Add(x);

X2.Add(x2);

Y2.Add(y);

Y2.Add(y2);

Series s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(x, y);

s.Points.AddXY(x2, y);

s.Points.AddXY(x2, y2);

s.Points.AddXY(x, y2);

s.Points.AddXY(x, y);

chart1.Series.Add(s);

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

if (checkBox1.Checked && radioButton1.Checked)

{

Random r = new Random();

int N = Convert.ToInt32(tbN.Text);

for(int i = 0; i < N; i++)

{

int x = r.Next(0, 30);

int y = r.Next(0, 30);

int x2 = r.Next(0, 30);

int y2 = r.Next(0, 30);

X.Add(x);

X.Add(x2);

Y.Add(y);

Y.Add(y2);

Series s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(x, y);

s.Points.AddXY(x2, y2);

chart1.Series.Add(s);

checkBox1.Checked = false;

}

}

else if(checkBox1.Checked && radioButton2.Checked)

{

R();

checkBox1.Checked = false;

}

}

private void btnClear\_Click(object sender, EventArgs e)

{

for (int i = 1; i < chart1.Series.Count; i++)

{

chart1.Series.RemoveAt(i);

i--;

}

chart1.Series[0].Points.Clear();

X.Clear();

Y.Clear();

X2.Clear();

Y2.Clear();

}

int Perevirka(double x, double y, int c, double x2,double x3,double y2,double y3)

{

List<double> X0 = new List<double>();

List<double> Y0 = new List<double>();

X0.Add(x2);

X0.Add(x3);

X0.Sort();

Y0.Add(y2);

Y0.Add(y3);

Y0.Sort();

if (Math.Round(x) > Math.Round(X2.Max()) || Math.Round(x) < Math.Round(X2.Min()))

{

c = 1;

}

else if (Math.Round(y) > Math.Round(Y2.Max()) || Math.Round(y) < Math.Round(Y2.Min()))

{

c = 1;

}

else if(Math.Round(x) > Math.Round(X0[1]) || Math.Round(x) < Math.Round(X0[0]))

{

c = 1;

}

else if (Math.Round(y) > Math.Round(Y0[1]) || Math.Round(y) < Math.Round(Y0[0]))

{

c = 1;

}

else c = 0;

return c;

}

void Draw()

{

for (int i = 1; i < chart1.Series.Count; i++)

{

chart1.Series.RemoveAt(i);

i--;

}

for (int i = 0; i < X.Count; i+=2)

{

if (X[i] > X2.Max() && X[i+1] > X2.Max())

{

X.RemoveAt(i);

X.RemoveAt(i);

Y.RemoveAt(i);

Y.RemoveAt(i);

//chart1.Series.RemoveAt(i / 2 + 1);

i-=2;

}

else if(X[i] < X2.Min() && X[i + 1] < X2.Min())

{

X.RemoveAt(i);

X.RemoveAt(i);

Y.RemoveAt(i);

Y.RemoveAt(i);

//chart1.Series.RemoveAt(i / 2 + 1);

i -= 2;

}

else if (Y[i] < Y2.Min() && Y[i + 1] < Y2.Min())

{

X.RemoveAt(i);

X.RemoveAt(i);

Y.RemoveAt(i);

Y.RemoveAt(i);

//chart1.Series.RemoveAt(i / 2 + 1);

i -= 2;

}

else if (Y[i] > Y2.Max() && Y[i + 1] > Y2.Max())

{

X.RemoveAt(i);

X.RemoveAt(i);

Y.RemoveAt(i);

Y.RemoveAt(i);

//chart1.Series.RemoveAt(i / 2 + 1);

i -= 2;

}

else if(X[i] <= X2.Max() && X[i + 1] <= X2.Max() && X[i] >= X2.Min() && X[i + 1] >= X2.Min() && Y[i] <= Y2.Max() && Y[i + 1] <= Y2.Max() && Y[i] >= Y2.Min() && Y[i + 1] >= Y2.Min())

{

Series s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(X[i], Y[i]);

s.Points.AddXY(X[i+1], Y[i+1]);

chart1.Series.Add(s);

}

}

if (X.Count != 0)

{

c = 0;

List<double> A2 = new List<double>();

List<double> B2 = new List<double>();

List<double> C2 = new List<double>();

A2.Add(Y2[0] - Y2[1]);

A2.Add(Y2[1] - Y2[1]);

A2.Add(Y2[1] - Y2[0]);

A2.Add(Y2[0] - Y2[0]);

B2.Add(X2[0] - X2[0]);

B2.Add(X2[0] - X2[1]);

B2.Add(X2[1] - X2[1]);

B2.Add(X2[1] - X2[0]);

C2.Add(X2[0] \* Y2[1] - Y2[0] \* X2[0]);

C2.Add(-X2[0] \* Y2[1] + Y2[1] \* X2[1]);

C2.Add(X2[1] \* Y2[0] - Y2[1] \* X2[1]);

C2.Add(-X2[1] \* Y2[0] + Y2[0] \* X2[0]);

for (int i = 0; i < X.Count(); i += 2)

{

List<double> P1 = new List<double>();

List<double> P2 = new List<double>();

List<int> C = new List<int>();

double a1, b1, c1;

double x, y;

a1 = Y[i] - Y[i + 1];

b1 = X[i + 1] - X[i];

c1 = X[i] \* Y[i + 1] - Y[i] \* X[i + 1];

for (int j = 0; j < 4; j++)

{

x = -(c1 \* B2[j] - b1 \* C2[j]) / (a1 \* B2[j] - b1 \* A2[j]);

y = -(a1 \* C2[j] - c1 \* A2[j]) / (a1 \* B2[j] - b1 \* A2[j]);

c = Perevirka(x, y, c, X[i], X[i + 1], Y[i], Y[i + 1]);

if (c == 0)

{

P1.Add(x);

P2.Add(y);

}

C.Add(c);

}

if (P1.Count == 2)

{

Series s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(P1[0], P2[0]);

s.Points.AddXY(P1[1], P2[1]);

chart1.Series.Add(s);

}

else if (P1.Count == 1)

{

if (C[0] == 0)

{

if (X[i] > X[i + 1])

{

Series s1 = new Series();

s1.ChartType = SeriesChartType.Line;

s1.BorderWidth = 4;

s1.Points.AddXY(P1[0], P2[0]);

s1.Points.AddXY(X[i], Y[i]);

chart1.Series.Add(s1);

}

else

{

Series s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(P1[0], P2[0]);

s.Points.AddXY(X[i + 1], Y[i + 1]);

chart1.Series.Add(s);

}

}

else if (C[1] == 0)

{

if (Y[i] > Y[i + 1])

{

Series s1 = new Series();

s1.ChartType = SeriesChartType.Line;

s1.BorderWidth = 4;

s1.Points.AddXY(P1[0], P2[0]);

s1.Points.AddXY(X[i], Y[i]);

chart1.Series.Add(s1);

}

else

{

Series s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(P1[0], P2[0]);

s.Points.AddXY(X[i + 1], Y[i + 1]);

chart1.Series.Add(s);

}

}

else if (C[3] == 0)

{

if (Y[i] > Y[i + 1])

{

Series s1 = new Series();

s1.ChartType = SeriesChartType.Line;

s1.BorderWidth = 4;

s1.Points.AddXY(P1[0], P2[0]);

s1.Points.AddXY(X[i+1], Y[i+1]);

chart1.Series.Add(s1);

}

else

{

Series s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(P1[0], P2[0]);

s.Points.AddXY(X[i], Y[i]);

chart1.Series.Add(s);

}

}

else if (C[2] == 0)

{

if (X[i] > X[i + 1])

{

Series s1 = new Series();

s1.ChartType = SeriesChartType.Line;

s1.BorderWidth = 4;

s1.Points.AddXY(P1[0], P2[0]);

s1.Points.AddXY(X[i+1], Y[i+1]);

chart1.Series.Add(s1);

}

else

{

Series s = new Series();

s.ChartType = SeriesChartType.Line;

s.BorderWidth = 4;

s.Points.AddXY(P1[0], P2[0]);

s.Points.AddXY(X[i], Y[i]);

chart1.Series.Add(s);

}

}

}

}

}

c = 0;

}

private void btnDraw\_Click(object sender, EventArgs e)

{

Draw();

}

}

}