**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 7**

Тема: Построение плоских полиномиальных кривых

Студент: Шиляева Н. С.

Группа: 80-304

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Написать программу, строящую полиномиальную кривую по заданным точкам. Обеспечить возможность изменения позиции точек и, при необходимости, значений касательных векторов и натяжения.

1. Решение задачи

Язык программирования: C#.

Выбранный язык удобен для решения задачи, так как он обладает удобными библиотеками для построения фигур и вывода их на экран.

OpenGL (Open Graphics Library) - спецификация, определяющая платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трехмерную компьютерную графику.

Слово сплайн (английское слово "spline") означает гибкую линейку, используемую для проведения гладких кривых через заданные точки на плоскости. Форма этого универсального лекала на каждом отрезке описывается кубической параболой. Сплайны широко используются в инженерных приложениях, в частности, в компьютерной графике. Итак, на каждом *i*–м отрезке [*xi*–1*, xi*]*, i=*1, 2,…, *N,* решение будем искать в виде полинома третьей степени: *Si*(*x*)*=ai+bi*(*x–xi*)*+ci*(*x*–*xi*)2/2*+di*(*x–xi*)3/6

Неизвестные коэффициенты *ai, bi, ci, di, i=*1, 2,..., *N,* находим из:

• условий интерполяции: *Si*(*xi*)*=fi, i=*1, 2,..., *N*; *S*1(*x*0)*=f*0*,*

• непрерывности функции *Si*(*xi–*1*)=Si–*1(*xi*–1)*, i=*2, 3,..., *N,*

• непрерывности первой и второй производной:

*S /i*(*xi–*1)=*S /i–*1(*xi*–1)*, S //i*(*xi*–1)*=S //i*–1(*x i*–1)*, i=*2, 3,..., *N*.

Учитывая, что , для определения 4*N* неизвестных получаем систему 4*N*–2 уравнений:

*ai=fi, i=*1, 2,..., *N,*

*bi hi – cihi*2/2 *+ di hi*3/6*=fi – fi*–1*, i=*1, 2,..., *N,*

*bi – bi–1 = ci hi – di hi*2/2*, i=*2, 3,..., *N,*

*di hi = ci – ci–*1 *, i=*2, 3,..., *N.*

где *hi=xi – xi–*1*.* Недостающие два уравнения выводятся из дополнительных условий: *S //*(*a*)*=S //*(*b*)*=0.* Можно показать, что при этом . Из системы можно исключить неизвестные *bi , di ,* получив систему *N+*1 линейных уравнений (СЛАУ) для определения коэффициентов *ci*:

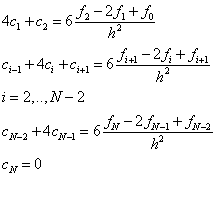
*c*0 *=*0*, cN =*0, *hici*–1*+*2(*hi+hi*+1)*ci+h i*+1*ci*+1*=*6

* , i=*1, 2,…, *N*–1*. (1)*

После этого вычисляются коэффициенты *bi, di:*

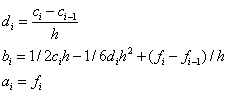
**, *i=*1, 2,..., *N.* (2)

В случае постоянной сетки *hi=h* эта система уравнений упрощается.



Данная СЛАУ имеет трехдиагональную матрицу и решается методом прогонки.

Коэффициенты определяются из формул:



Для вычисления значения *S*(*x*) в произвольной точке отрезка *z*∈[*a, b*] необходимо решить систему уравнений на коэффициенты *ci, i=*1,2,…, *N*–1*,* затем найти все коэффициенты *bi, di.* Далее, необходимо определить, на какой интервал [*xi*0*, xi*0–1] попадает эта точка, и, зная номер *i0,* вычислить значение сплайна и его производных в точке *z*

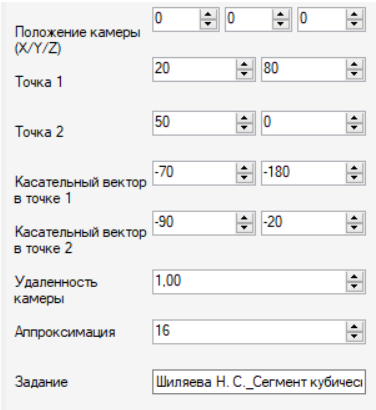
*S*(*z*)*=ai*0 *+bi*0(*z–xi*0)*+ci*0(*z–xi*0)2/2*+di*0(*z–x i*0)3/6

*S /*(*z*)*=bi*0*+ci*0(*z–xi*0)*+di*0(*z–x i*0)2/2*, S //*(*z*)*=ci*0*+di*0(*z–x i*0)*.*

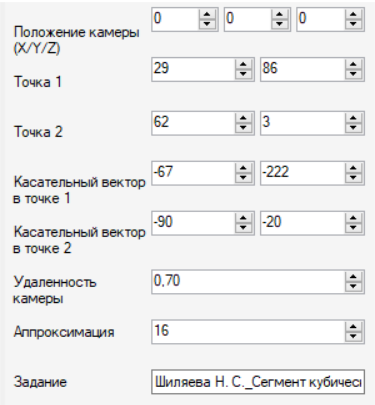
1. Набор тестов

Задаем все необходимые данные сегмента кубического сплайна, указывая координаты точек и касательных, вводим необходимое (удобное) для просмотра значение удаленности камеры и аппроксимации:

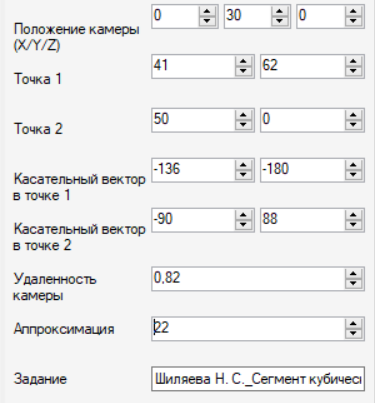
Тест 1:



Тест 2:



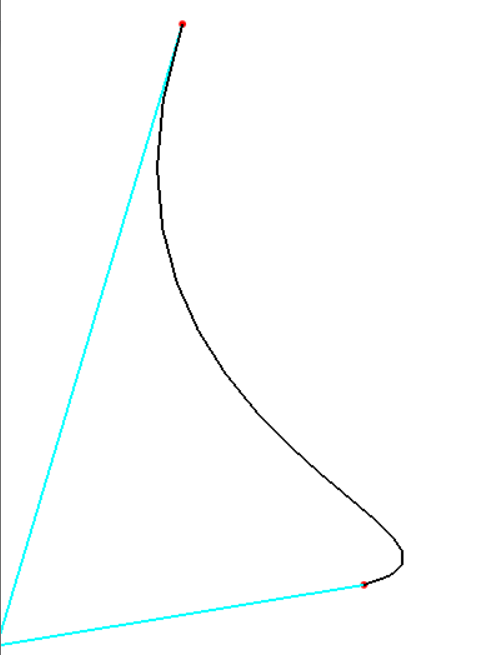
Тест 3:



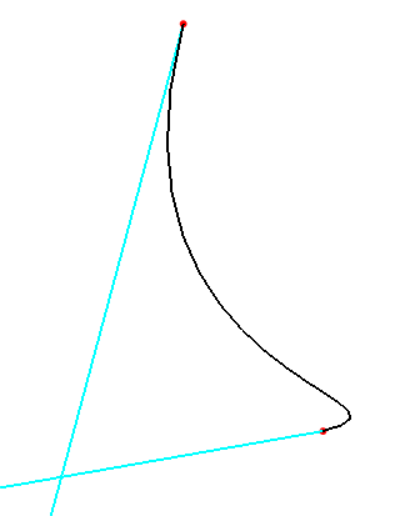
1. Результаты выполнения тестов

Наглядно видим выполнение программы по тестам, описанным ранее.

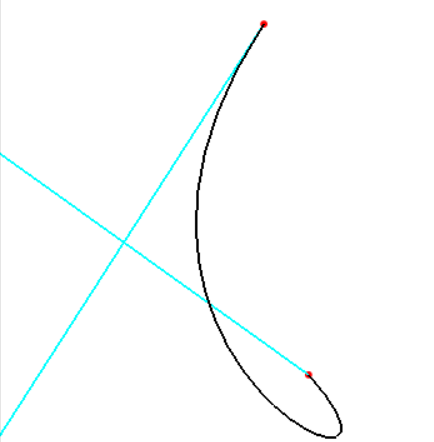
Тест 1:



Тест 2:



Тест 3:



1. Листинг программы

*Файл Program.cs*

using System;

using SharpGL;

using CGLabPlatform;

// Создание и работа с классом приложения аналогична предыдущим примерам, только в

// в данном случае наследуемся от шаблона OGLApplicationTemplate<T>, в силу чего

// для вывода графики будет использоваться элемент управления OGLDevice работающий

// через OpenGL (далее OGL). Код OGLDevice размещается в Controls\OGLDevice.cs

public abstract class CGLabDemoOGL : OGLApplicationTemplate<CGLabDemoOGL>

{

[STAThread] static void Main() { RunApplication(); }

public double angleX = 0;

public double angleY = 0;

public double angleZ = 0;

public double k = 0;

public double shiftX = 0;

public double shiftY = 0;

public double shiftZ = 0;

public double angle = 0;

public double toRad = Math.PI / 180;

public double [] xy(double t)

{

double [] res = new double[2];

double resx, resy;

resx = P1.X + P\_1.X \* t + (3 \* (P2.X - P1.X) - 2 \* P\_1.X - P\_2.X) \* t \* t;

resy = P1.Y + P\_1.Y \* t + (3 \* (P2.Y - P1.Y) - 2 \* P\_1.Y - P\_2.Y) \* t \* t;

res[0] = resx;

res[1] = resy;

return res;

}

#region Свойства

[DisplayNumericProperty(new[] { 0d, 0d, 0d }, 1, 0, "Положение камеры (X/Y/Z)")]

public virtual DVector3 cameraAngle

{

get { return Get<DVector3>(); }

set { if (Set(value)) UpdateModelViewMatrix(); }

}

[DisplayNumericProperty(new double[] { 20, 80 }, 1, "Точка 1")]

public virtual DVector2 P1 { set; get; }

[DisplayNumericProperty(new double[] { 50, 0 }, 1, "Точка 2")]

public virtual DVector2 P2 { set; get; }

[DisplayNumericProperty(new double[] { -70, -180 }, 1, "Касательный вектор в точке 1")]

public virtual DVector2 P\_1 { set; get; }

[DisplayNumericProperty(new double[] { -90, -20 }, 1, "Касательный вектор в точке 2")]

public virtual DVector2 P\_2 { set; get; }

[DisplayNumericProperty(1.0d, 0.01, 2, "Удаленность камеры", 0.01)]

public virtual double cameraDistance { set; get; }

[DisplayNumericProperty(16, 1, "Аппроксимация", 2)]

public virtual double apr { set; get; }

[DisplayTextBoxProperty("Сегмент кубического сплайна по конечным точкам и касательным", "Задание")]

public virtual string s { set; get; }

#endregion

// Само создание объекта типа OpenGL осуществляется при создании устройства вывода (класс OGLDevice)

// и доступ к нему можно получить при помощи свойства gl данного объекта (RenderDevice) или объекта

// типа OGLDeviceUpdateArgs передаваемого в качестве параметра методу OnDeviceUpdate(). Данный метод,

// как и сама работа с устройством OpenGL реализуются в параллельном потоке. Обращение к устройству

// OpenGL из другого потока не допускается (создание многопоточного рендера возможно, но это достаточно

// специфическая архитектура, например рендинг частей экрана в текустуры а потом их объединение).

// Для большинства функций библиотеки OpenGL при отладке DEBUG конфигурации осуществляется проверка

// ошибок выполнения и их вывод в окно вывода Microsoft Visual Studio. Поэтому при отладке и написании

// кода связанного с OpenGL необходимо также контролировать ошибки библиотеки OpenGL в окне вывода.

protected override void OnMainWindowLoad(object sender, EventArgs args)

{

base.VSPanelWidth = 300;

base.ValueStorage.RightColWidth = 60;

base.RenderDevice.VSync = 1;

#region Обработчики событий мыши и клавиатуры -------------------------------------------------------

RenderDevice.MouseWheel += (s, e) => cameraDistance += e.Delta / 1000.0;

RenderDevice.MouseMoveWithRightBtnDown += (s, e) =>

{

shiftX += e.MovDeltaX \* 0.2;

shiftY -= e.MovDeltaY \* 0.2;

};

RenderDevice.MouseMoveWithLeftBtnDown += (s, e) =>

{

angleX += e.MovDeltaX;

angleY += e.MovDeltaY;

angle += e.MovDelta;

};

#endregion

// Как было отмечено выше вся работа связанная с OGL должна выполнятся в одном потоке.

// осуществляется в отдельном потоке, а OnMainWindowLoad()является событием возбуждаемым потоком

// пользовательского интерфейса (UI). Поэтой причине весь код ниже добавляется в диспетчер устройства

// вывода (метод AddScheduleTask() объекта RenderDevice) и выполняется ассинхронно в контексте потока

// OGL. Сам диспетчер является очередью типа FIFO (First In First Out - т.е. задания обрабатываются

// строго в порядке их поступления) и гарантирует, что все задания добавленные в OnMainWindowLoad()

// будут выполнены до первого вызова метода OnDeviceUpdate() (aka OnPaint)

#region Инициализация OGL и параметров рендера -----------------------------------------------------

RenderDevice.AddScheduleTask((gl, s) =>

{

gl.Disable(OpenGL.GL\_DEPTH\_TEST);

gl.ClearColor(0, 0, 0, 0);

});

#endregion

#region Инициализация буфера вершин -----------------------------------------------------------------

RenderDevice.AddScheduleTask((gl, s) =>

{

// TODO:

}, this);

#endregion

#region Уничтожение буфера вершин по завершению работы OGL ------------------------------------------

RenderDevice.Closed += (s, e) => // Событие выполняется в контексте потока OGL при завершении работы

{

var gl = e.gl;

// TODO:

};

#endregion

#region Обновление матрицы проекции при изменении размеров окна и запуске приложения ----------------

RenderDevice.Resized += (s, e) =>

{

var gl = e.gl;

// TODO:

};

#endregion

}

private void UpdateModelViewMatrix()//метод вызывается при изменении свойств cameraAngle и cameraDistance

{

#region Обновление объектно-видовой матрицы ---------------------------------------------------------

RenderDevice.AddScheduleTask((gl, s) =>

{

// TODO:

});

#endregion

}

protected unsafe override void OnDeviceUpdate(object s, OGLDeviceUpdateArgs e)

{

double t = 0;

double dt = 1 / apr;

var gl = e.gl;

gl.ClearColor(1, 1, 1, 1);

// Очищаем буфер экрана и буфер глубины (иначе рисоваться все будет поверх старого)

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT | OpenGL.GL\_STENCIL\_BUFFER\_BIT);

gl.MatrixMode(OpenGL.GL\_PROJECTION);

gl.LoadIdentity();

gl.Ortho(-10, 90, -10, 90, -1, 1);

gl.Scale(cameraDistance, cameraDistance, cameraDistance);

gl.Translate(shiftX, shiftY, 0);

gl.Enable(OpenGL.GL\_POINT\_SMOOTH);

//точки

gl.PointSize(6);

gl.Color(1f, 0f, 0f);

gl.Begin(OpenGL.GL\_POINTS);

gl.Vertex(P1.X, P1.Y);

gl.Vertex(P2.X, P2.Y);

gl.End();

//касательные

gl.Color(0, 1f, 1f);

gl.LineWidth(2);

DVector2 n1 = (P1 + P\_1).Normalized();

gl.Begin(OpenGL.GL\_LINES);

gl.Vertex(P1.X, P1.Y);

gl.Vertex(P\_1.X, P\_1.Y);

gl.End();

gl.Color(0, 1f, 1f);

gl.LineWidth(2);

DVector2 n2 = (P2 + P\_2).Normalized();

gl.Begin(OpenGL.GL\_LINES);

gl.Vertex(P2.X, P2.Y);

gl.Vertex(P\_2.X, P\_2.Y);

gl.End();

//кривая

gl.Color(0f, 0f, 0f);

gl.LineWidth(2);

gl.Begin(OpenGL.GL\_LINES);

while (t < 1)

{

double[] p1 = xy(t);

double[] p2 = xy(t + dt);

if (t + dt > 1)

{

gl.Vertex(p1);

gl.Vertex(P2.X, P2.Y);

break;

}

else

{

gl.Vertex(p1);

gl.Vertex(p2);

t += dt;

}

}

gl.End();

gl.Flush();

}

}

1. Вывод

В ходе данной лабораторной работы получены основные навыки по построению полиномиальных кривых.

Список литературы

1. Шилдт, Герберт. Ш57 С# 4.0: полное руководство. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 1056 с.: ил. — Парал. тит. англ. (дата обращения: 30.11.2020).
2. Прайс. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов, 3-е издание. -М.: Питер, 2018 - 640 с. (дата обращения: 01.12.2020).