**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**По лабораторной работе № 4**

**по дисциплине «Компьютерная графика»**

Тема: **Кубические сплайны**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0303 |  | Болкунов В.О. |
| Преподаватель |  | Герасимова Т.В. |

Санкт-Петербург

2023

**Задание**

Реализовать интерактивное приложение, отображающее заданные полиномиальные кривые: NURBS кривые, построение окружности

**Основные теоретические положения**

**Сплайны** – это гладкие (имеющие несколько непрерывных производных) кусочно-полиномиальные функции, которые могут быть использованы для представления функций, заданных большим количеством значений и для которых неприменима аппроксимация одним полиномом. Так как сплайны гладки, экономичны и легки в работе, они используются при построении произвольных функций для:

* моделирования кривых;
* аппроксимации данных с помощью кривых;
* выполнения функциональных аппроксимаций;
* решения функциональных уравнений.

**Неоднородный рациональный B-сплайн, NURBS** (Non-uniform rational B-spline) — математическая форма, применяемая в компьютерной графике для генерации и представления кривых и поверхностей. Как следует из названия, является частным случаем B-сплайнов, причём широко распространённым из-за своей стандартизированности и относительной простоты.

NURBS сплайны разрабатывались в инженерных целях для построения различных объектов и поверхностей произвольной формы (корпуса автомобилей, кораблей, самолётов и других аппаратов). Они широко применяются в системах автоматического проектирования и в 3D моделировании. NURBS кривые можно применять в компьютерной графике, так как они не дают петель, имеют несколько непрерывных производных (их количество зависит от количества узлов) и ограниченны контрольными точками.

Для построения NURBS кривой необходимо задать следующие параметры:

* массив узловых точек – данные узлы определяют, где и как контрольные точки будут влиять на кривую
* массив контрольных точек – контрольные точки задают форму кривой, ограничивая её
* массив весов контрольных точек – веса позволяют контролировать вклад каждой контрольной точки в итоговую кривую.

Степень итогового сплайна можно рассчитать как

Следующий шаг в построении NURBS-кривой – определение базисных функций , где *j* – степень базисной функции а *i* – её номер,

Базисные функции 0 степени являются константными кусочно-заданными функциями следующего вида:

Базисные функции высших порядков можно построить рекурсивно:

Где , а

После построения базисных функций степени *p* (степень итоговой кривой), по следующей формуле можно получить функцию полученной кривой:

И перебирая параметр можно получить все точки сплайна.

**Выполнение работы.**

Для выполнения работы были использованы язык Python 3.10 и библиотеки PyQt6 и PyOpenGL.

Подключение графической библиотеки осуществляется с помощью виджета QOpenGLWidget

В нашем случае был создан виджет GLWidget наследуемый от данного класса, в котором с помощью переопределённых методов **initializeGL и paintGL** осуществляется соответственно подготовка кадра и отрисовка изображения. Сама отрисовка для гибкости использования осуществляется задаваемой функцией в поле **function**. В переопределённом методе resizeGL отслеживается изменение размера окна, устанавливается Viewport и посылается сигнал об изменении размера области отрисовки.

Задавав функцию рисования и вызвав метод **update** у GLWidget можно добиться рисования любых объектов в соответствии с заданной функцией.

Функция построения сплайна определена в модуле **drawing**. В ней происходит построение базисных функций и итоговой функции кривой в соответствии с формулами в предыдущем разделе, базисные функции определены с помощью декоратора *@cache* библиотеки functools, это позволяет кэшировать результат выполнения базисных функций и значительно ускорить построение кривой.

Управление приложением осуществляется с помощью виджета **ControlPanel**, который содержит в себе ползунок для настройки количества узловых точек (генерируются равноотстоящие точки от 0 до 1) и ползунки для регулирования весов контрольных точек.

Элементы управления и графический виджет объединены компонентом **MainWindow** (наследуемом от QMainWindow), в нём происходит связывание событий интерфейса управления с обновлениями изображения.

**Тестирование**

Возможные изображения сплайна и интерфейса программы представлены на рисунках 1 и 2.

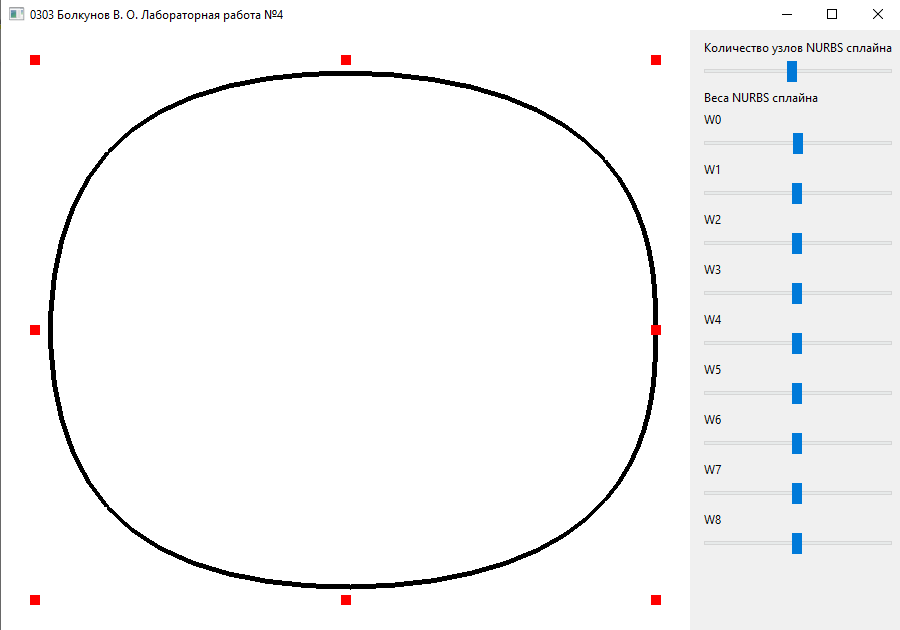


Рисунок 1: Сплайн с равными весами контрольных точек

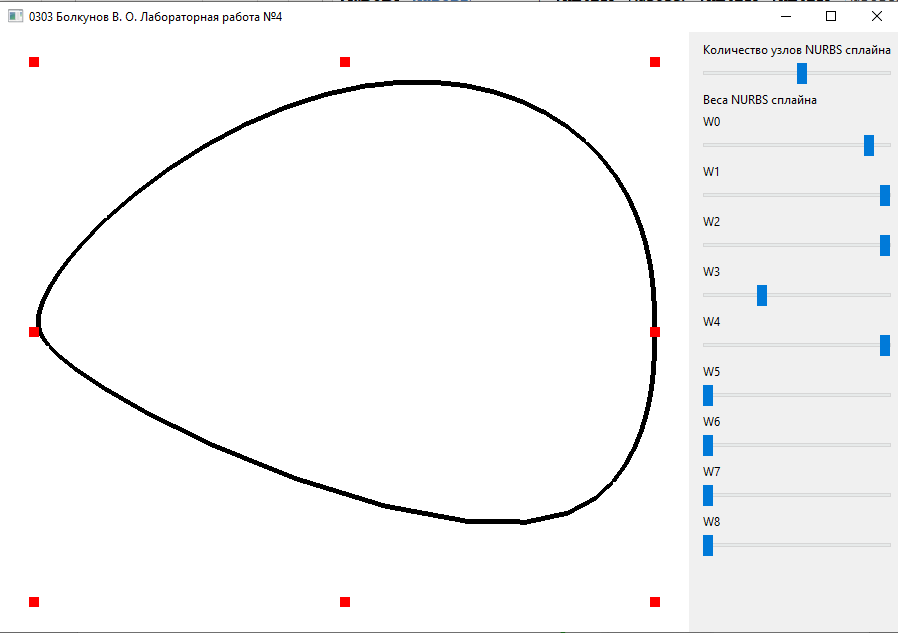


Рисунок 2: Сплайн со случайными весами

**Выводы:**

В ходе лабораторной работы были изучены принципы построения NURBS сплайнов. Была реализована программа позволяющая настраивать параметры сплайна для рисования кривой, аппроксимирующей окружность.

**ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД**

***Файл main.py***

import sys  
  
from PyQt6.QtWidgets import QApplication  
  
from MainWindow import MainWindow  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 w = MainWindow()  
 w.show()  
 sys.exit(app.exec())

***Файл GLWidget.py***

from OpenGL import GL as gl  
from PyQt6 import QtCore  
from PyQt6.QtOpenGLWidgets import QOpenGLWidget  
  
  
*# Виджет OpenGL*class GLWidget(QOpenGLWidget):  
 viewPortResized = QtCore.pyqtSignal((int, int))  
  
 def \_\_init\_\_(self, parent=None):  
 super().\_\_init\_\_(parent)  
 *# Функция вызываемая в цикле отрисовки (при обновлениях)* self.function = None  
  
 def resizeGL(self, w: int, h: int) -> None:  
 gl.glViewport(0, 0, w, h)  
 self.viewPortResized.emit(w, h)  
  
 *# Функция вызываемая перед любым обновлением* def initializeGL(self):  
 *# Заливка кадра* gl.glClearColor(1, 1, 1, 1)  
 *# Очистка буферов (цвета и глубины)* gl.glClear(gl.GL\_COLOR\_BUFFER\_BIT | gl.GL\_DEPTH\_BUFFER\_BIT)  
  
 *# Функция вызываемая при обновлении (посредством update или при изменении размеров)* def paintGL(self):  
 *# Вызов рендер-функции* if self.function is not None:  
 self.function()

***Файл drawing.py***

from functools import cache  
import numpy as np  
  
  
def buildNurbs(T: list[float], P: list[np.ndarray], W: list[float]):  
 if len(W) != len(P):  
 raise ValueError()  
  
 *# Узлы* m = len(T) - 1  
 *# Управляющие точки* n = len(P) - 1  
 *# Степень сплайна* p = m - n - 1  
  
 *# Базисные функции N\_(k, m)* N = dict()  
  
 *# Генерация базисных функций порядка 0* def buildNk1(ti, ti1):  
 return lambda t: 1 if (ti <= t <= ti1) else 0  
  
 for i in range(m):  
 N[(i, 0)] = buildNk1(T[i], T[i + 1])  
  
 *# Генерация базисных функций k-го порядка* def buildNkm(i, j):  
 nonlocal N, T  
  
 @cache  
 def Nin(t):  
 f = (t - T[i]) / (T[i + j] - T[i]) *#if (T[i + n] - T[i]) != 0 else 0* g = (T[i + 1 + j] - t) / (T[i + 1 + j] - T[i + 1]) *#if (T[i + 1 + n] - T[i + 1]) != 0 else 0* return f \* N[(i, j - 1)](t) + (g) \* N[(i + 1, j - 1)](t)  
  
 return Nin  
  
 for j in range(1, p + 1):  
 for i in range(n - j + p + 1):  
 N[(i, j)] = buildNkm(i, j)  
  
 def F(t):  
 f = [N[(i, p)](t) \* W[i] for i in range(n + 1)]  
 c1 = sum([f[i] \* P[i] for i in range(n + 1)])  
 c2 = sum([f[i] for i in range(n + 1)])  
 return c1 / c2 if c2 != 0 else c1  
  
 return F, N

***Файл ControlPanel.py***

from PyQt6.QtCore import Qt  
from PyQt6.QtWidgets import QWidget, QVBoxLayout, QSlider, QLabel  
from PyQt6 import QtCore  
  
  
*# Виджет панели управления*class ControlPanel(QWidget):  
 weightsChanged = QtCore.pyqtSignal()  
  
 def \_\_init\_\_(self, weights, parent=None):  
 super().\_\_init\_\_(parent)  
 lt = QVBoxLayout(self)  
 self.setLayout(lt)  
  
 knotsLabel = QLabel('Количество узлов NURBS сплайна', self)  
 self.knots = QSlider(Qt.Orientation.Horizontal)  
  
 weightsLabel = QLabel('Веса NURBS сплайна', self)  
 wLabels = [QLabel(f'W{i}') for i in range(weights)]  
 self.wSliders = [QSlider(Qt.Orientation.Horizontal) for i in range(weights)]  
  
 for w in [knotsLabel, self.knots, weightsLabel]:  
 lt.addWidget(w)  
  
 for i in range(weights):  
 lt.addWidget(wLabels[i])  
 lt.addWidget(self.wSliders[i])  
 self.wSliders[i].valueChanged.connect(lambda: self.weightsChanged.emit())  
 self.wSliders[i].setMaximum(100)  
 self.wSliders[i].setMinimum(1)  
  
 lt.addStretch()

***Файл MainWindow.py***

from PyQt6 import QtCore  
from PyQt6.QtWidgets import QMainWindow, QSplitter  
from ControlPanel import ControlPanel  
from GLWidget import GLWidget  
from drawing import \*  
from OpenGL import GL as gl  
  
  
*# Главное окно*class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 *# Создаём виджет OpenGL* self.glwidget = GLWidget(self)  
  
 *# Задаём рендер-функцию* self.glwidget.function = self.renderFunction  
  
 *# Точки окружности* self.P = P = [  
 np.array([0.9, 0]),  
 np.array([0.9, 0.9]),  
 np.array([0, 0.9]),  
 np.array([-0.9, 0.9]),  
 np.array([-0.9, 0]),  
 np.array([-0.9, -0.9]),  
 np.array([0, -0.9]),  
 np.array([0.9, -0.9]),  
 np.array([0.9, 0]),  
 ]  
 *# Узловой вектор* self.T = None  
 *# Веса контрольных точек* self.W = None  
  
 *# Виджет управления* self.control = ControlPanel(len(self.P), self)  
  
 *# Настраиваем ползунок количества узлов* self.control.knots.setMinimum(10)  
 self.control.knots.setMaximum(25)  
 self.control.knots.setValue(15)  
  
 *# Задаём начальные веса* for i in range(9):  
 self.control.wSliders[i].setValue(50)  
  
 self.onKnotsChanged()  
 self.onWeightsChanged()  
  
 self.control.weightsChanged.connect(self.onWeightsChanged)  
 self.control.knots.valueChanged.connect(self.onKnotsChanged)  
  
 sp = QSplitter(self)  
 sp.addWidget(self.glwidget)  
 sp.addWidget(self.control)  
 sp.setStretchFactor(0, 1)  
 self.setCentralWidget(sp)  
 self.resize(900, 600)  
 self.setWindowTitle("0303 Болкунов В. О. Лабораторная работа №4")  
  
 def onKnotsChanged(self):  
 self.T = np.linspace(0, 1, self.control.knots.value())  
 self.rebuildSpline()  
  
 def onWeightsChanged(self):  
 self.W = [s.value() for s in self.control.wSliders]  
 self.rebuildSpline()  
  
 *# Сборка сплайна* def rebuildSpline(self):  
 if self.W is None:  
 self.onWeightsChanged()  
 if self.T is None:  
 self.onKnotsChanged()  
  
 self.F, N = buildNurbs(self.T, self.P, self.W)  
 X = np.linspace(0, 1, 100)[1:-1]  
 self.Points = [self.F(x) for x in X]  
 self.redraw()  
  
 def renderFunction(self):  
 gl.glPointSize(10)  
 gl.glLineWidth(5)  
 gl.glBegin(gl.GL\_LINE\_STRIP)  
 gl.glColor3dv((0, 0, 0))  
 for p in self.Points:  
 gl.glVertex2dv(p)  
 gl.glEnd()  
 gl.glBegin(gl.GL\_POINTS)  
 gl.glColor3dv((1, 0, 0))  
 for p in self.P:  
 gl.glVertex2dv(p)  
 gl.glEnd()  
  
 *# Вызов обновления изображения* @QtCore.pyqtSlot()  
 def redraw(self):  
 self.glwidget.update()