Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 4
"КУБИЧЕСКИЕ СПЛАЙНЫ"

Выполнил: Калмак Д.А.
Факультет: ФКТИ
Группа: 0303
Преподаватель: Герасимова Т.В.

Санкт-Петербург
2023 г.

Цель работы.

- ознакомление со сплайнами.

- проанализировать полученное задание, выделить информационные объекты и действия;

- разработать программу, реализующую B-сплайн.

Задание.

Разработать программу, реализующую B-сплайн со следующими параметрами: n = 6, k = 4. Узловой вектор равномерный.

Выполнение работы.

Уравнение B-сплайна представляет собой следующее выражение:

$Q(t) = \frac{\sum_{i=0}^{n} P_i N_{i,k}(t) w_i}{\sum_{i=0}^{n} P_i N_{i,k}(t)}$, где базовая функция $N_{i,k}$ определена рекурсивно формулами Кокса-де Бура:

$$N_{i,k}(t): N_{i,0}(t) = \begin{cases} 1, t_i \le t \le t_{i+1} \\ 0, \text{иначе} \end{cases}$$

$$\forall k > 0, N_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} N_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(t)$$

P_i – контрольные точки в количестве семи значений, а w_i – вес, ассоциированный с контрольной точкой P_i и определяющий ее «влияние» на построение сплайна. Узловой вектор равномерный, поэтому t = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]. Веса для контрольных точек взяты за единицу w = [1, 1, 1, 1, 1, 1, 1].

В классе mainWindow в self.stack (QStackedWidget) добавлен виджет класса glWidgetSpline, который наследуется от glWidget0. В классе mainWindow добавлен атрибут self.lblclear, который принадлежит классу QLabel. Он содержит текст над кнопкой очистки виджета. Кнопка очистки виджета self.btnclear принадлежит классу

QPushButton. Нажимая на кнопку, с помощью метода clicked у self.btnclear, который передает сигнал, что кнопка нажата, и метода connect, который связывает кнопку с методом update_clear, метод update_clear в классе mainWindow запускается от нажатия кнопки. В методе для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, обновляется значение атрибута self.clearstatus, который добавлен в класс glWidget0, от которого наследуются класс glWidgetSpline. Происходит обновление виджетов с помощью метода updateGL. В слой buttonsLayout добавлены виджеты self.lblclear, self.btnclear.

В виджете класса glWidgetFractal в методе paintGL осуществляется смещение координат с помощью функции glTranslatef(). Создан список t значений узлового вектора t = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]. Создан список w значений веса для контрольных точек w = [1, 1, 1, 1, 1, 1, 1]. Обработка контрольных точек происходит следующим образом: в класс glWidgetSpline добавлен метод mousePressEvent(self, event). Поскольку используются такие функции, как glTranslatef() и gluPerspective(), то необходим перерасчет граничных координат виджета для правильной отработки захвата места нажатия мыши. Для этого вычисляется aspect a, который равен отношению ширины к высоте виджета, значения для этого берутся из атрибутов класса, которые содержат в себе актуальные значения размера виджета. Вычисляется параметр t, как произведение двух и тангенса половины угла перспективы. Затем уже получаются коэффициенты для координат x и y произведением смещения по оси z на половину параметра t и для ширины еще на коэффициент a. С помощью метода pos() для event получаются значения текущих координат нажатия мыши, однако они переопределяются с учетом полученных коэффициентов и сдвига к центру. После того как координаты вычислены и если длина списка self.xy контрольных точек, который был добавлен как атрибут в класс glWidget0, меньше семи, то в список добавляется эта контрольная точка. Затем вызывается метод self.updateGL() для обновления виджета. В методе paintGL

отрисовываются сразу контрольные точки, которые пользователь вводит. Для наглядности их размер увеличен с помощью функции glPointSize(2.0). После отрисовки точек размер для отрисовки возвращен с помощью этой же функции в исходное значение. Если список контрольных точек заполняется семью значениями, то на виджет нажатия уже не повлияют, начинается обработка B-сплайна. Получаются координаты x, y, z контрольных точек. Создан список xlist, который будет содержать x координаты точек сплайна. Затем запускается цикл, в котором значение t будет перебираться от минимального значения до максимального в узловом векторе с шагом 0.01. Для каждого значения t запускается метод q класса glWidgetSpline, в него передаются степень, текущее значение t: ti, список t, список x координат контрольных точек, список w веса контрольных точек. Метод q – это запрограммированная функция Q(t). Для суммы числителя в методе используется переменная sum1, а для суммы знаменателя sum2. Метод возвращает их отношение. Внутри метода вызывается метод n класса glWidgetSpline, который принимает индекс i, степень, текущее значение t: ti, список узлового вектора t. Метод n – это запрограммированная функция $N_{i,k}(t)$. Сначала проводится проверка для k = 0 и если $\mathrm{t[i]} <= \mathrm{ti} < \mathrm{t[i+1]}$, то возвращается 1, иначе 0. Если $k > 0$, то метод возвращает значение выражения $\frac{(\mathrm{ti} - \mathrm{t[i]})}{(\mathrm{t[i+k]} - \mathrm{t[i]})} * \mathrm{self.n(i, k-1, ti, t)} + \frac{(\mathrm{t[i+k+1]} - \mathrm{ti})}{(\mathrm{t[i+k+1]} - \mathrm{t[i+1]})} * \mathrm{self.n(i+1, k-1, ti, t)}$ в соответствии с формулой. Аналогично координатам x для B-сплайна рассчитываются координаты y. Для этого создан список ylist. После вычисления всех точек B-сплайна с помощью функций glBegin(GL_LINE_STRIP), glVertex3f(), glEnd() отрисовывается B-сплайн. После этого блок для очистки виджета. Если self.clearstatus равен True, то с помощью функции glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) происходит очистка, также список self.xy очищается, а self.clearstatus переводится в False.

ТЕСТИРОВАНИЕ.

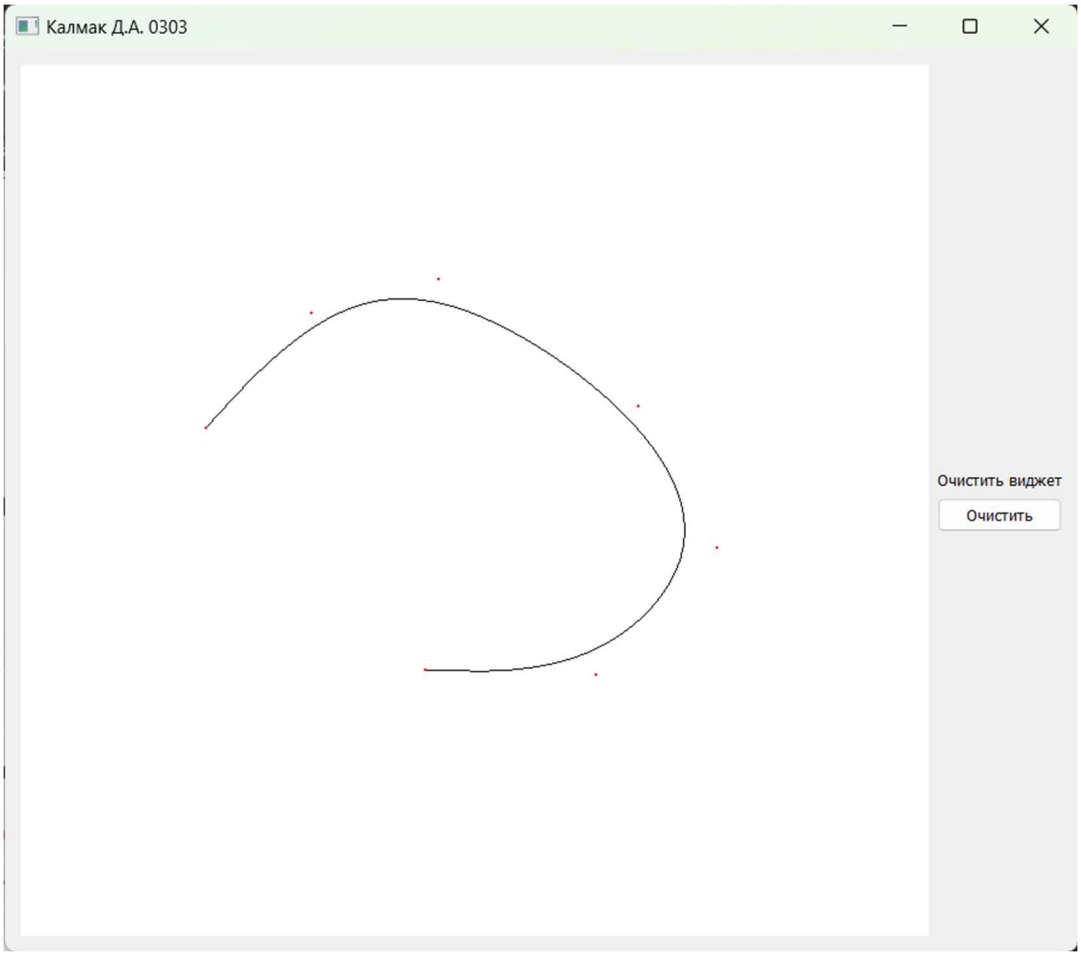

Рисунок 1 – B-сплайн

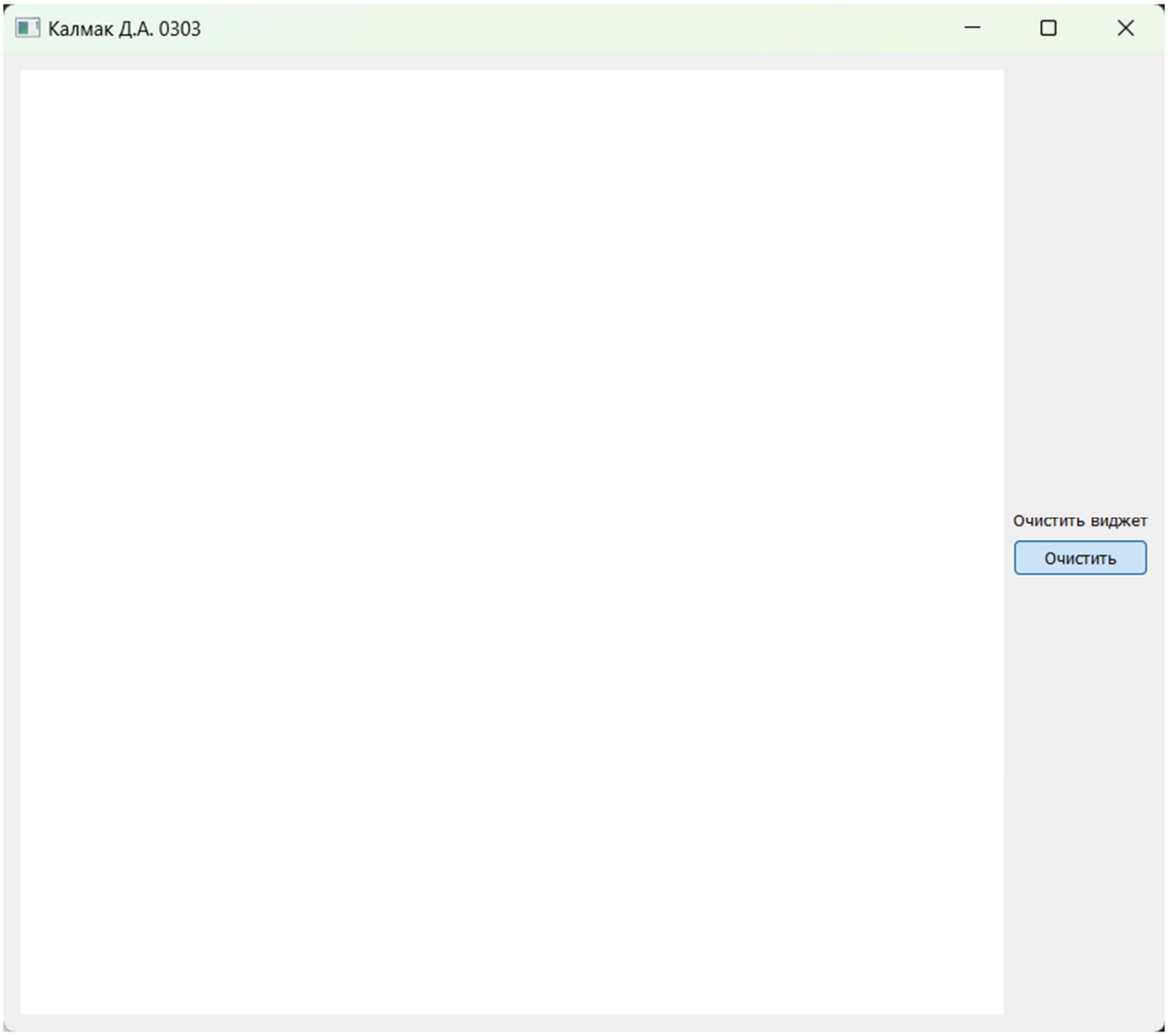

Рисунок 2 – Очистка виджета

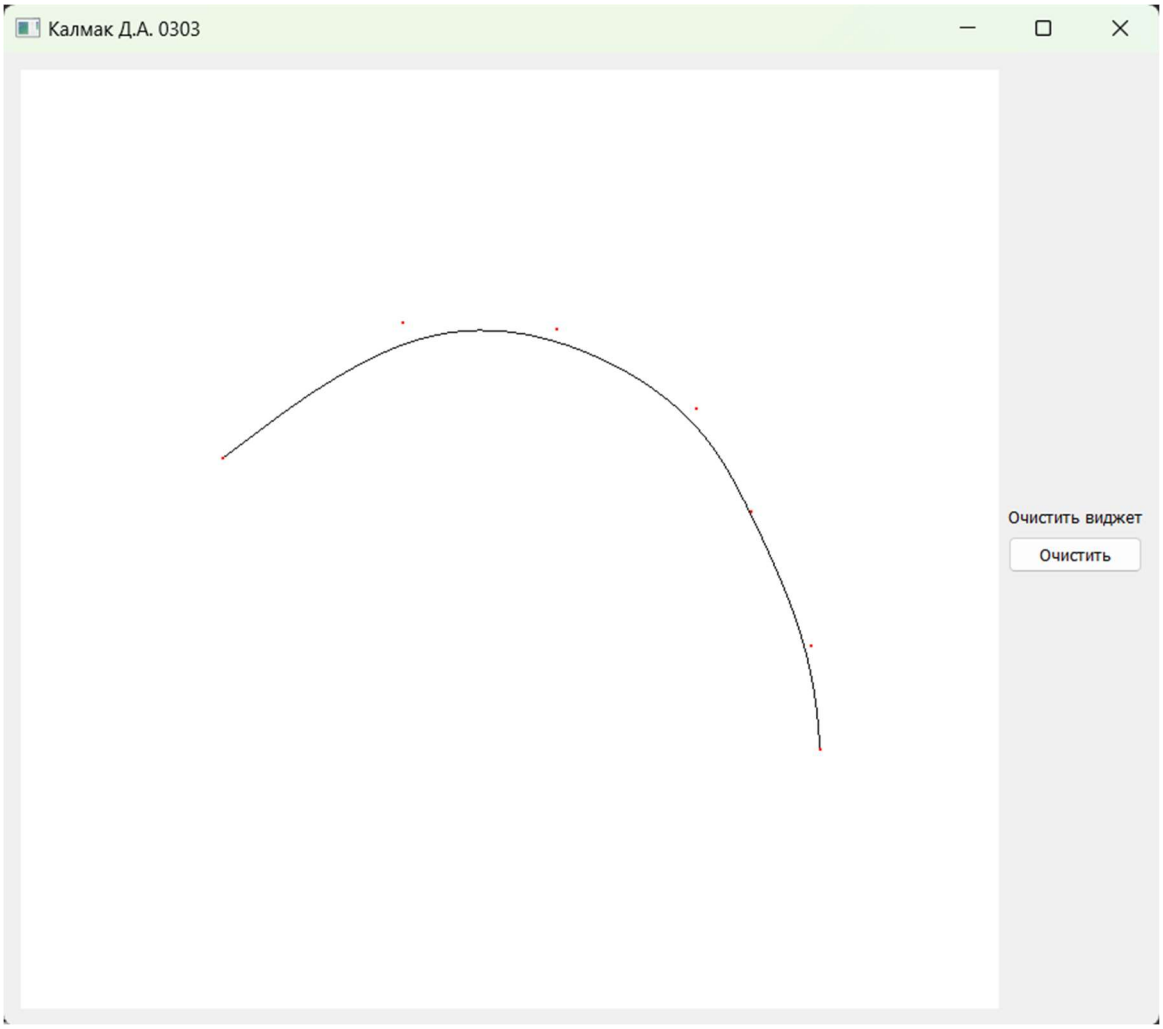

Рисунок 3 – B-сплайн после очистки виджета

Вывод.

В результате выполнения лабораторной работы была разработана программа, реализующая представление заданного B-сплайна с заданными параметрами.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import math
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from PyQt5.QtCore import Qt
from PyQt5.QtOpenGL import *
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import (QWidget, QLabel,
                             QComboBox, QStackedWidget, QSlider, QCheckBox,
QPushButton)


class mainWindow(QWidget):
    def __init__(self, parent=None):
        super(mainWindow, self).__init__()
        self.stack = QStackedWidget()
        self.stack.addWidget(glWidgetSpline())

        buttonsLayout = QtWidgets.QVBoxLayout()
        self.lblclear = QLabel("Очистить виджет", self)
        self.btnclear = QPushButton("Очистить", self)
        self.btnclear.clicked.connect(self.update_clear)
        buttonsLayout.addStretch()
        buttonsLayout.addWidget(self.lblclear)
        buttonsLayout.addWidget(self.btnclear)
        buttonsLayout.addStretch()

        mainLayout = QtWidgets.QHBoxLayout()
        widgetLayout = QtWidgets.QHBoxLayout()
        widgetLayout.addWidget(self.stack)
        mainLayout.addLayout(widgetLayout)
        mainLayout.addLayout(buttonsLayout)
        self.setLayout(mainLayout)
        self.setWindowTitle("Калмак Д.А. 0303")

    def update_clear(self):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).clearstatus = True
            self.stack.widget(i).updateGL()


class glWidget0(QGLWidget):
    def __init__(self, parent=None):
        QGLWidget.__init__(self, parent)
        self.setMinimumSize(750, 720)
        self.w = 480
        self.h = 480
        self.xy = []
        self.clearstatus = False

    def initializeGL(self):
```

```
        glClearColor(1.0, 1.0, 1.0, 0.1)
        glClearDepth(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)
        glShadeModel(GL_SMOOTH)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(45.0, 1, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)

    def paintGL(self):
        pass

    def resizeGL(self, w, h):
        self.w = w
        self.h = h
        glViewport(0, 0, w, h)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        aspect = w / h
        gluPerspective(45.0, aspect, 0.1, 100)
        glMatrixMode(GL_MODELVIEW)


class glWidgetSpline(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(0, 0, -4.0)
        t = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
        w = [1, 1, 1, 1, 1, 1, 1]
        glColor4f(1, 0, 0, 1)
        glPointSize(2.0)
        glBegin(GL_POINTS)
        for i in range(len(self.xy)):
            glVertex3f(self.xy[i][0], self.xy[i][1], self.xy[i][2])
        glEnd()
        glColor4f(0, 0, 0, 1)
        glPointSize(1.0)
        if len(self.xy) == 7:
            x = list(map(list, zip(*self.xy)))[0]
            y = list(map(list, zip(*self.xy)))[1]
            z = list(map(list, zip(*self.xy)))[2]
            xlist = []
            for i in range(len(t) - 1):
                ti = t[i]
                xlist.append(self.q(3, ti, t, x, w))
                while ti < t[i + 1]:
                    ti += 0.01
                    xlist.append(self.q(3, ti, t, x, w))
            xlist = xlist[1:len(xlist) - 1]
            ylist = []
            for i in range(len(t) - 1):
                ti = t[i]
                ylist.append(self.q(3, ti, t, y, w))
                while ti < t[i + 1]:
                    ti += 0.01
```

```
                    ylist.append(self.q(3, ti, t, y, w))
            ylist = ylist[1:len(ylist) - 1]
            glBegin(GL_LINE_STRIP)
            for i in range(len(xlist)):
                glVertex3f(xlist[i], ylist[i], 0)
            glEnd()

        if self.clearstatus:
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
            self.xy = []
            self.clearstatus = False

    def q(self, k, ti, t, c, w):
        sum = 0
        for i in range(len(c)):
            sum += c[i] * self.n(i, k, ti, t) * w[i]
        sum2 = 0
        for i in range(len(c)):
            sum2 += self.n(i, k, ti, t)
        if sum2 == 0:
            return 0
        return sum / sum2

    def n(self, i, k, ti, t):
        if k == 0:
            if t[i] <= ti < t[i+1]:
                return 1
            else:
                return 0
        else:
            return (ti - t[i]) / (t[i+k] - t[i]) * self.n(i, k-1, ti, t) + (t[i+k+1]
- ti) / (t[i+k+1] - t[i+1]) * self.n(i+1, k-1, ti, t)

    def mousePressEvent(self, event):
        a = self.w / self.h
        t = math.tan(45 / 2 * math.pi / 180) * 2
        xcoef = 4 * a * (t / 2)
        ycoef = 4 * (t / 2)
        xpos = (-(self.w / 2) + event.pos().x()) / self.w * 2 * xcoef
        ypos = -(-(self.h / 2) + event.pos().y()) / self.h * 2 * ycoef
        if len(self.xy) < 7:
            self.xy.append([xpos, ypos, 0])
            print(len(self.xy))
        self.updateGL()
        super().mousePressEvent(event)


if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    qWindow = QtWidgets.QMainWindow()
    window = mainWindow(qWindow)
    window.show()
    sys.exit(app.exec_())
```