Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 5
"Расширения OpenGL, программируемый графический конвейер. Шейдеры"

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

Санкт-Петербург
2023 г.

Цель работы.

- ознакомление с шейдерами.

- проанализировать полученное задание, выделить информационные объекты и действия;

- разработать программу, реализующую шейдер.

Задание.

Разработать программу, реализующую шейдер с анимацией прозрачности, изменяющейся по синусоидальному закону.

Выполнение работы.

В классе mainWindow в self.stack (QStackedWidget) добавлен виджет класса glWidgetShader, который наследуется от glWidget0 и является классом glWidgetSpline, но дополненным шейдером. Атрибут self.boxshader пренадлежит классу QCheckBox. Он необходим для управления режимом шейдера: включен и выключен. Когда в нем меняется состояние, через метод stateChanged у self.boxshader передается с помощью метода connect состояние в метод self.update_shader. Если активно, то есть равно Qt.Checked, то для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, меняется значение атрибута shader_flag на True. Атрибут self.shader_flag был добавлен в класс glWidget0, от которого наследуются класс glWidgetShader. Иначе атрибуту присваивается значение False. В обоих случаях происходит обновление виджетов с помощью метода updateGL. В слой buttonsLayout добавлен виджет self.boxshader.

В класс glWidget0, от которого наследуется класс glWidgetShader, добавлены атрибуты self.time для синусоидальной зависимости прозрачности в шейдере, атрибут

self.shader_program принадлежит классу QOpenGLShaderProgram, который связывает и использует шейдеры. Атрибут self.timer принадлежит классу Qtimer. Он необходим, чтобы после истечения таймера виджет обновлялся. С помощью метода start(50) у self.timer задан таймер. С помощью сигнала timeout у self.timer и метода connect self.timer связан с методом self.updateGL и после истечения таймера каждый раз виджет обновляется. В методе initilizeGL добавлены вызовы методов addShaderFromSourceFile у self.shader_program для компиляции и загрузки шейдеров: вершинного QOpenGLShader.Vertex "v5.vert" и фрагментного QOpenGLShader.Fragment "f5.frag". Вершинный шейдер версии 430. В функции main gl_Position равно ftransform() и оставляет вершины без изменений. Фрагментный шейдер версии 430. Принимает параметр uniform float time для синусоидальной зависимости прозрачности. В функции main добавлены параметры float w – угловая частота, и float a – начальная фаза. Формула прозрачности, которая изменяется по синусоидальному закону: $Alpha = Alpha_m * \sin(wt + a)$, где $Alpha_m$ — амплитуда прозрачности, $w = \frac{2\pi}{T}$ — ее угловая частота, a — начальная фаза. gl_Fragcolor равен вектору vec4, содержащему цвет, а также прозрачность, равную sin(w * time + a). С помощью метода link у self.shader_program шейдеры связываются.

В классе glWidgetShader в методе paintGL отключается запись в буфер глубины для корректности полупрозрачности с помощью функции glDepthMask(GL_FALSE), включается режим смешения с помощью glEnable(GL_BLEND). Смешение задается функцией glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA), где sfactor равен GL_SRC_ALPHA, а dfactor GL_ONE_MINUS_SRC_ALPHA. После отрисовки запись в буфер глубины включается с помощью glDepthMask(GL_TRUE), режим смешения выключается с помощью glDisable(GL_BLEND). После этого блок для шейдеров. Если self.shader_flag равен True, то к атрибуту self.time прибавляется 0.1. С помощью метода bind у self.shader_program self.shader_program делается

текущей шейдерной программой. С помощью методов setUniformValue и uniformLocation у self.shader_program uniform переменная time получает значение self.time. Заливка поверхности включена с помощью функции glPolygonMode(GL_FRONT_AND_BACK, GL_FILL).

ТЕСТИРОВАНИЕ.

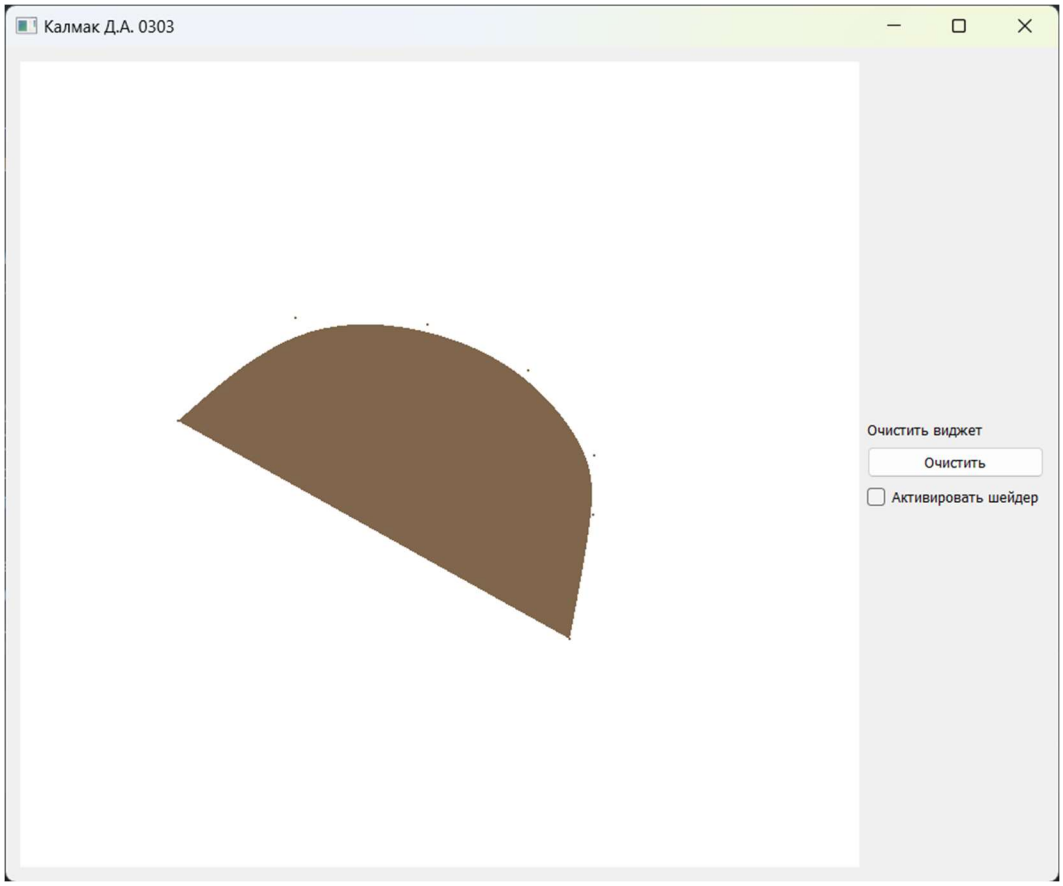

Рисунок 1 – Шейдер выключен

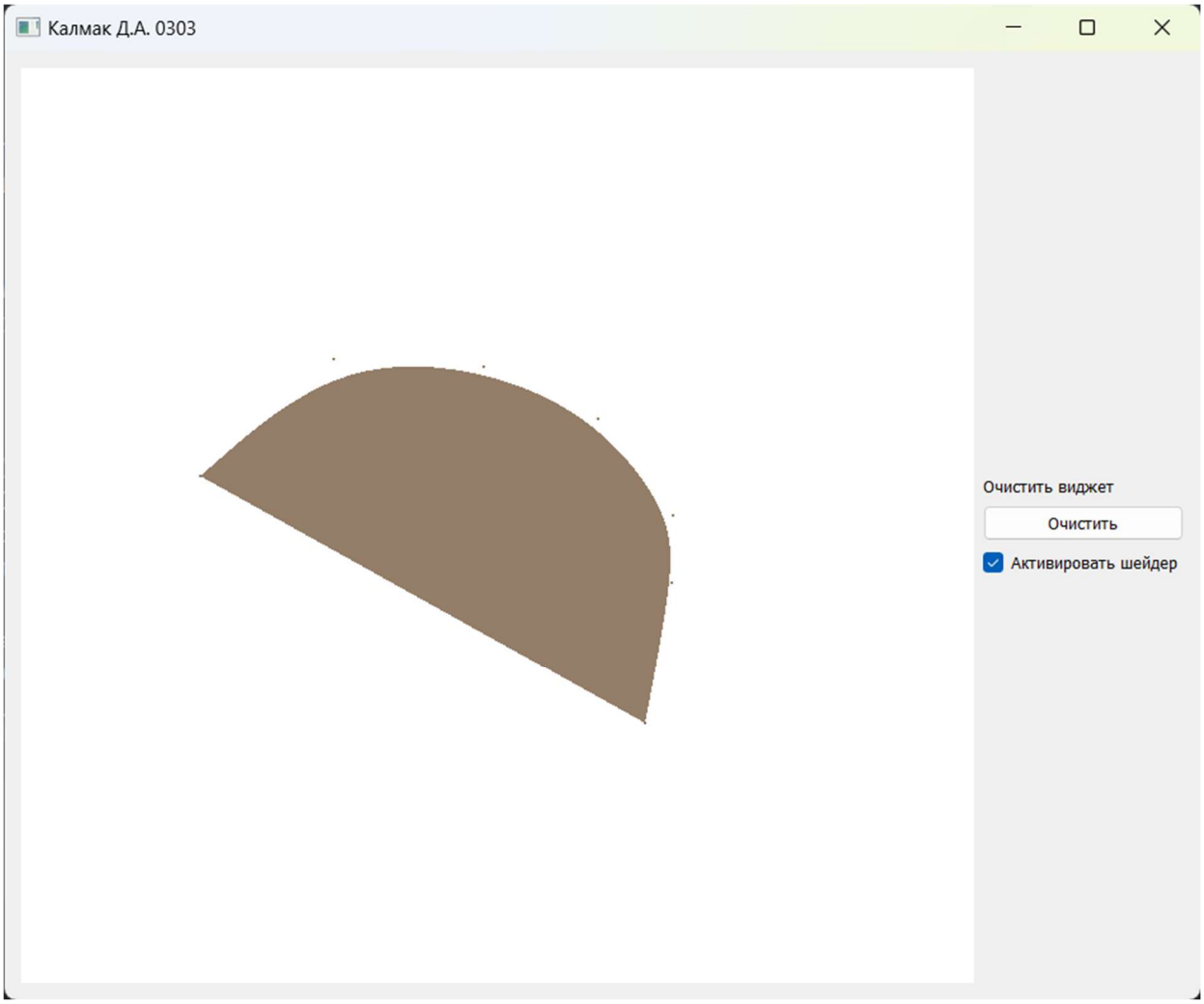

Рисунок 2 – Анимация с шейдером работают

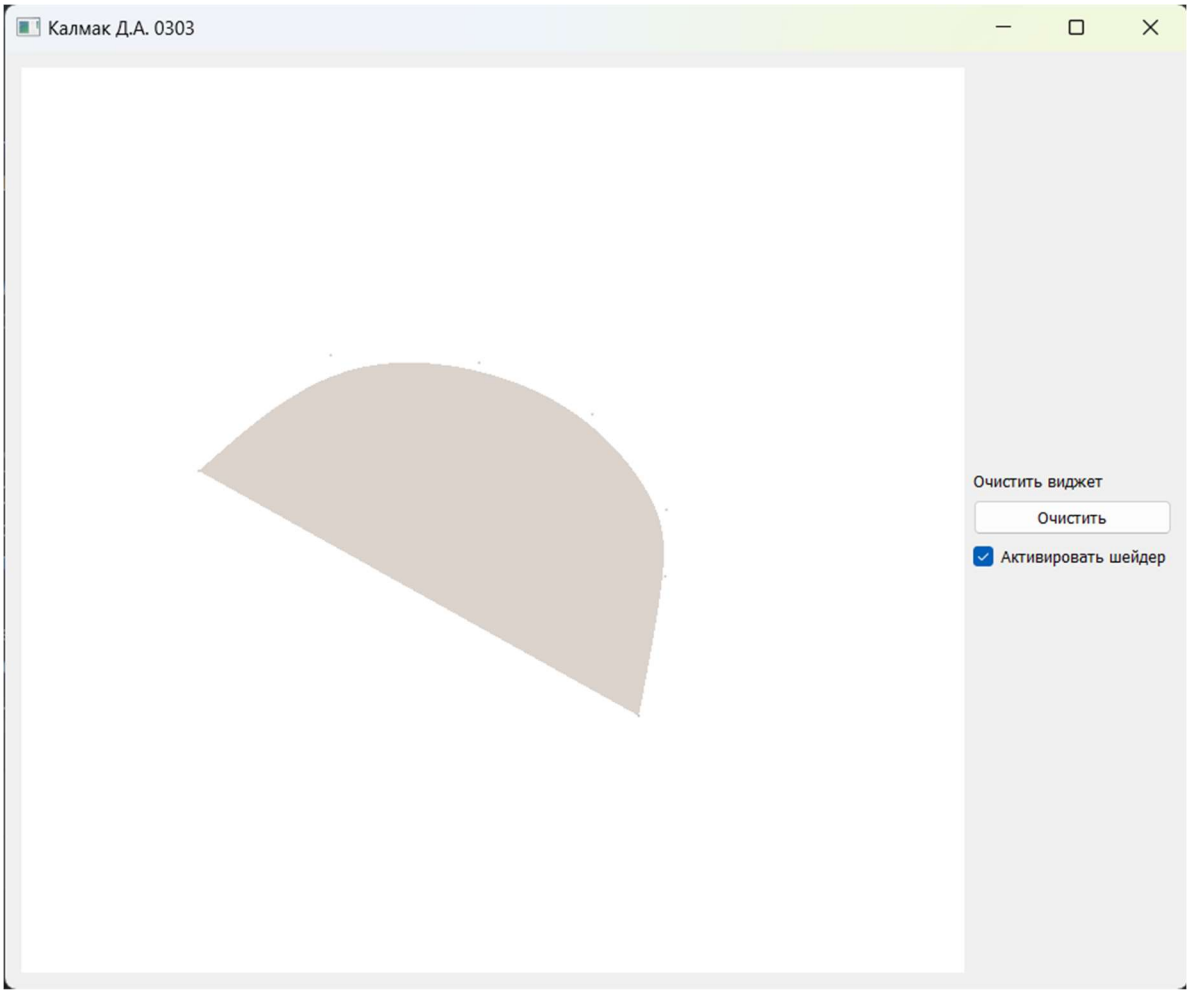

Рисунок 3 – Анимация с шейдером работают в другой момент времени

Вывод.

В результате выполнения лабораторной работы была разработана программа, реализующая шейдер с анимацией прозрачности, изменяющейся по синусоидальному закону.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main5.py

```
import math
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QOpenGLShaderProgram, QOpenGLShader
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(glWidgetShader())

        buttonsLayout = QtWidgets.QVBoxLayout()
        self.lblclear = QLabel("Очистить виджет", self)
        self.btnclear = QPushButton("Очистить", self)
        self.btnclear.clicked.connect(self.update_clear)
        self.boxshader = QCheckBox("Активировать шейдер", self)
        self.boxshader.stateChanged.connect(self.update_shader)
        buttonsLayout.addStretch()
        buttonsLayout.addWidget(self.lblclear)
        buttonsLayout.addWidget(self.btnclear)
        buttonsLayout.addWidget(self.boxshader)
        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()

    def update_shader(self, state):
        if state == Qt.Checked:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).shader_flag = True
                self.stack.widget(i).updateGL()
```

```
        else:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).shader_flag = False
                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
        self.time = 0
        self.shader_program = QOpenGLShaderProgram()
        self.shader_flag = False
        self.timer = QTimer()
        self.timer.start(50)
        self.timer.timeout.connect(self.updateGL)

    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, 750/720, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)
        self.shader_program.addShaderFromSourceFile(QOpenGLShader.Vertex, "v5.vert")
        self.shader_program.addShaderFromSourceFile(QOpenGLShader.Fragment,
"f5.frag")
        self.shader_program.link()

    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 glWidgetShader(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(0, 0, -4.0)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
```

```
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
        if self.shader_flag:
            self.time += 0.1
            self.shader_program.bind()

self.shader_program.setUniformValue(self.shader_program.uniformLocation("time"),
float(self.time))

        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(0.5, 0.4, 0.3, 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(0.5, 0.4, 0.3, 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]
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)
            glBegin(GL_POLYGON)
            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
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)

    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_())
```

Файл v5.vert

```
#version 430

void main(void)
{
    gl_Position = ftransform();
}
```

Файл f5.frag

```
#version 430

uniform float time;

void main()
{
    float w = 1;
    float a = 1.4;
```

```
    gl_FragColor = vec4(0.5, 0.4, 0.3, sin(w * time + a));
}
```