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: , где  — амплитуда прозрачности, — ее угловая частота, — 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.

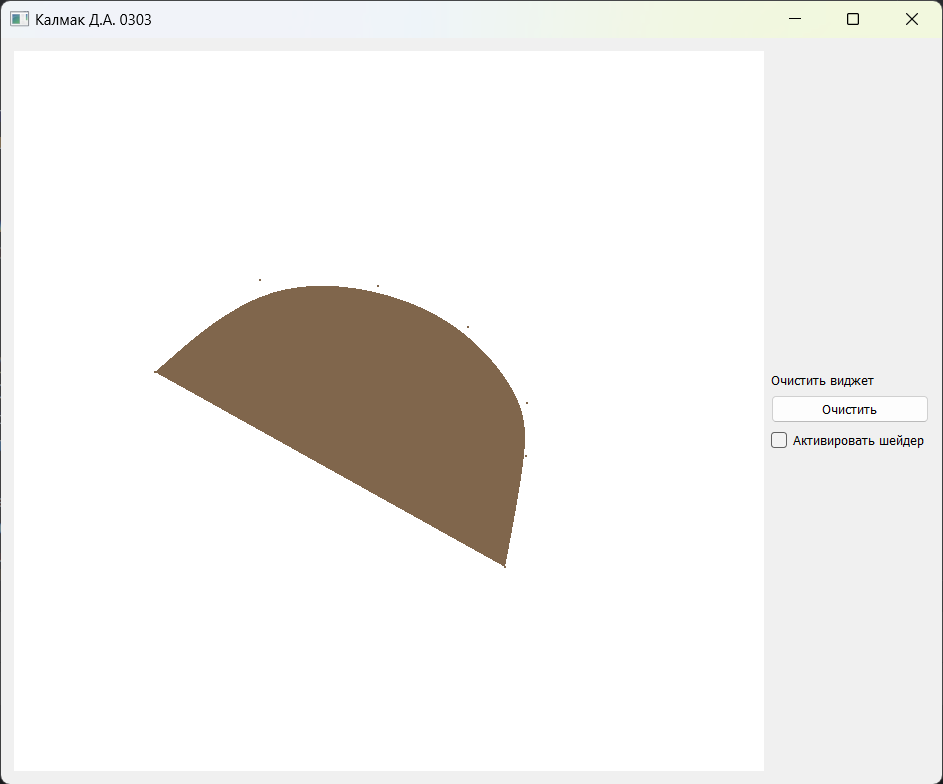


Рисунок 1 – Шейдер выключен

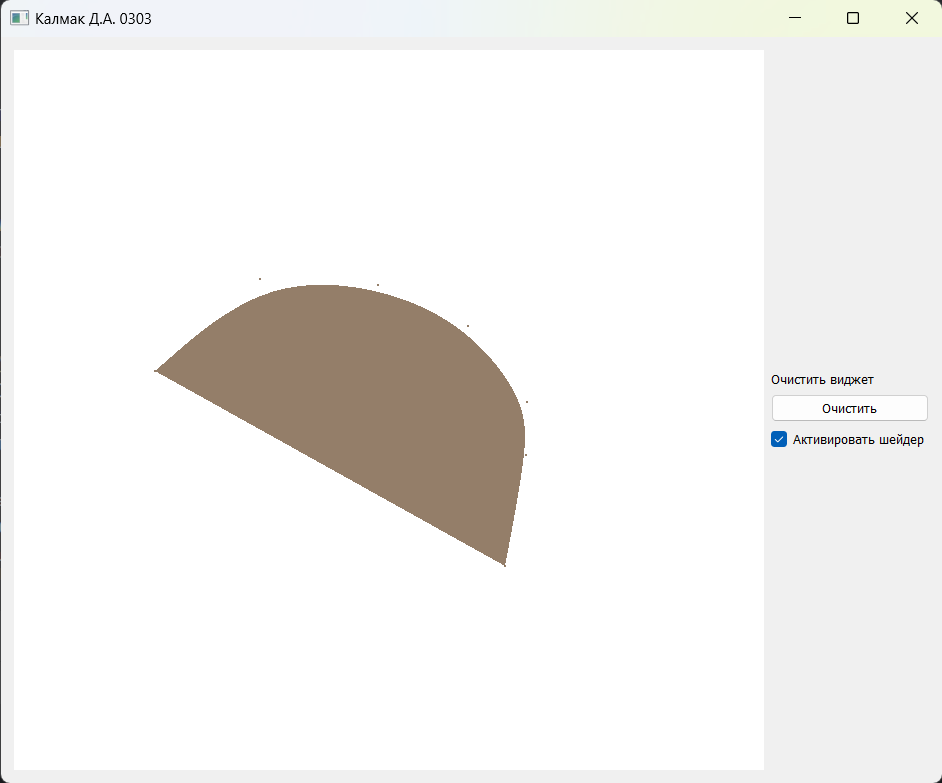


Рисунок 2 – Анимация с шейдером работают

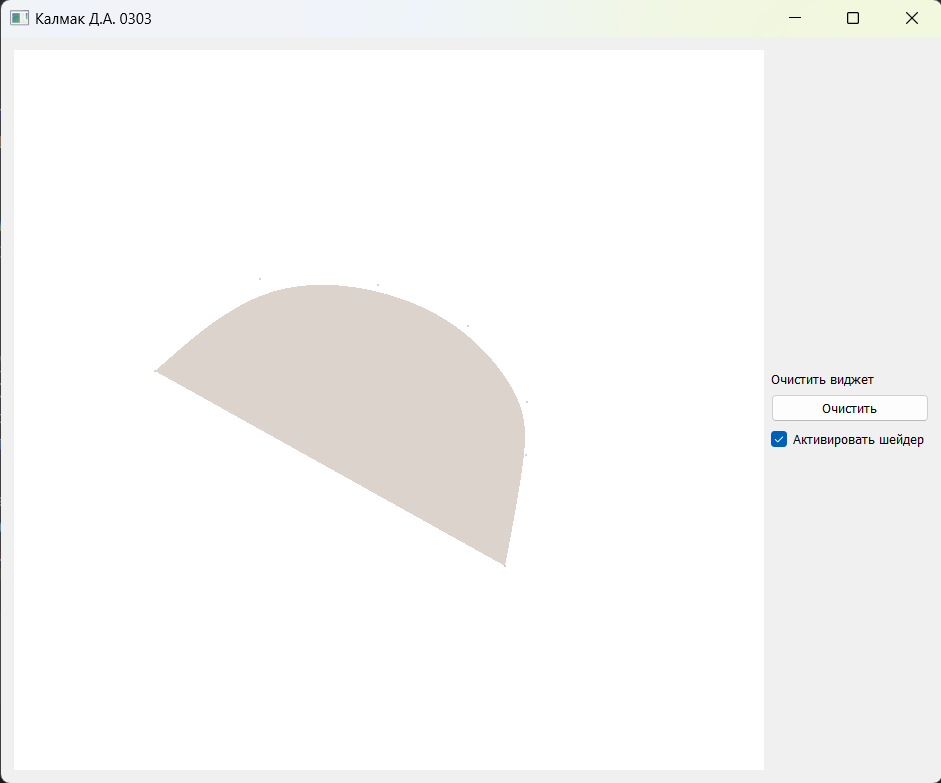


Рисунок 3 – Анимация с шейдером работают в другой момент времени

## 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));  
}