МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Компьютерная графика»

Тема: Расширения OpenGL, программируемый графический конвейер. Шейдеры

Студент гр. 0382 Корсунов А.А.

Преподаватель Герасимова Т.В.

Санкт-Петербург

2023

Цель работы.

Реализовать приложение, демонстрирующее эффект растворения с использованием шейдеров.

Теоретические положения.

Язык программирования высокоуровневых расширений называются языком затенения OpenGL (OpenGL Shading Language –GLSL), иногда именуемым языком шедеров OpenGL (OpenGL Shader Language). Этот язык очень похож на язык С но имеет встроенные типы данных и функции полезные в шедерах вершин и фрагментов.

Шейдер является фрагментом шейдерной программы, которая заменяет собой часть графического конвейера видеокарты. Тип шейдера зависит от того, какая часть конвейера будет заменена. Каждый шейдер должен выполнить свою обязательную работу, т. е. записать какие-то данные и передать их дальше по графическому конвейеру.

Шейдерная программа – это небольшая программа, состоящая из шейдеров (вершинного и фрагментного, возможны и др.) и выполняющаяся на GPU (Graphics Processing Unit), т. е. на графическом процессоре видео-карты.

- Существует пять мест в графическом конвейере, куда могут быть встроены шейдеры. Соответственно шейдеры делятся на типы:

  – вершинный шейдер (vertex shader);

  – геометрический шейдер (geometric shader);

  – фрагментный шейдер (fragment shader);

  – два тесселяционных шейдера (tesselation), отвечающие за два разных этапа тесселяции (они доступны в OpenGL 4.0 и выше).

Дополнительно существуют вычислительные (compute) шейдеры, которые выполняются независимо от графического конвейера.

Вершинные шейдеры – это программы, которые производят математические операции с вершинами, иначе говоря, они предоставляют возможность выполнять программируемые алгоритмы по изменению параметров вершин. Каждая вершина определяется несколькими переменными, например, положение вершины в 3D-пространстве определяется координатами: x, y и z.

- Вершины также могут быть описаны характеристиками цвета, текстурными координатами и т. п. Вершинные шейдеры, в зависимости от алгоритмов, изменяют эти данные в процессе своей работы, например, вычисляя и записывая новые координаты и/или цвет. Входные данные вершинного шейдера – данные об одной вершине геометрической модели, которая в данный момент обрабатывается. Обычно это координаты в пространстве, нормаль, компоненты цвета и текстурные координаты. Результирующие данные выполняемой программы служат входными для дальнейшей части конвейера, растеризатор делает линейную интерполяцию входных данных для поверхности треугольника и для каждого пикселя исполняет соответствующий пиксельный шейдер.

- Для управления входными и выходными данными вершинного шейдера используются квалификаторы типов, определенные как часть языка шейдеров OpenGL:

– переменные-атрибуты (attribute) – передаются вершинному шейдеру от приложения для описания свойств каждой вершины;

– однообразные переменные (uniform) – используются для передачи данных как вершинному, так и фрагментному процессору. Не могут меняться чаще, чем один раз за полигон – относительно постоянные значения;

– разнообразные переменные (varying) – служат для передачи данных от вершинного к фрагментному процессору. Данные переменные могут быть различными для разных вершин, и для каждого фрагмента будет выполняться интерполяция.

- Фрагментный шейдер не может выполнять операции, требующие знаний о нескольких фрагментах, изменить координаты (пара x и y) фрагмента.

- Фрагментный шейдер не заменяет стандартные операции, выполняемые в конце обработки пикселей, но заменяет часть графического конвейера (ГК), обрабатывающего каждый полученный на предыдущих стадиях ГК фрагмент (не пиксель). Обработка может включать такие стадии, как получение данных из текстуры, просчет освещения, просчет смешивания.

- Обязательной работой для фрагментного шейдера является запись цвета фрагмента во встроенную переменную gl_FragColor, или его отбрасывание специальной командой discard. В случае отбрасывания фрагмента, никакие расчеты дальше с ним производиться не будут, и фрагмент уже не попадет в буфер кадра.

- Если задачей вершинного шейдера являлось вычисление позиции вершины, а также других выходных параметров вершины на основе uniform- и attribute-переменных, то в задачи фрагментного шейдера будет входить вычисление цвета фрагмента и его глубины на основе встроенных и определяемых пользователем varying- и uniform-переменных.

- Фрагментный шейдер обрабатывает входной поток данных и производит выходной поток данных – пикселей изображения.

- Фрагментный шейдер получает следующие данные:

– разнообразные переменные (varying) от вершинного шейдера – как встроенные, так и определенные разработчиком;

– однообразные переменные (uniform) – для передачи произвольных относительно редко меняющихся параметров.

Задание.

Разработать программу, реализующую эффект растворения (тает облако) с помощью шейдеров

Ход работы.

Лабораторная работа выполнялась на языке программирования «Python» с использованием модулей «tkinter» и «OpenGL».

1. Определение шейдеров

а) Вершинный шейдер

код шейдера:

```
vertex_shader = """
in vec3 position;
varying vec3 vertex_color;
void main()
{
  gl_Position = vec4(position, 1.0);
  vertex_color = vec4(0.0f, 1.0f, 1.0f, 1.0f);
}
```

```
"""
```

Очевидно, что нет необходимости как-то изменять координаты вершин будущего облака, отчего весь вершинный шейдер сводится к переписыванию уже заданных координат и объявлению разнообразной переменной vertex_color, в которую будет записан голубой цвет, эта переменная будет передана в фрагментный шейдер.

б) Фрагментный шейдер

код шейдера:

```
fragment_shader = """
varying vec3 vertex_color;
uniform float alpha;
void main()
{
  gl_FragColor = vec4(vertex_color, alpha);
}
"""
```

Т.к. стоит задача растворения изображенного объекта, то нужно будет изменять альфа-канал так, пока изображение не станет прозрачным (т. е. до 0). Для этого была объявлена однообразная переменная alpha, которая будет изменяться вне шейдера, а в шейдере будет устанавливаться в каждую итерацию новый альфа-канал.

2. Сборка шейдеров

а) Компиляция шейдеров

Для компиляции шейдеров была написана соответствующая функция, которая принимает код шейдера (коды были рассмотрены в пункте 1) и его тип (вершинный или фрагментный), описание работы функции приведены в комментариях кода (все комментарии начинаются с символа “#”)

```
def compileShader(source, shaderType):  # принимает (код шейдера, тип шейдера), возвращает скомпилированный шейдер
    shader = GL.glCreateShader(shaderType)  # создание объекта шейдера (принимает тип создаваемого шейдера)
    GL.glShaderSource(shader, source)  # загрузка кода в созданный объект шейдера
    GL.glCompileShader(shader)  # компиляция кода, загруженного ранее в объект шейдера
    return shader
```

б) VAO

Для создания объекта массива вершин (Vertex Array Object – VAO) также была написана функция, которая принимает шейдерную программу (будет описана в следующем пункте) и возвращает VAO. В этой функции помимо создания VAO происходит и его привязка к контексту OpenGL. Более подробное объяснение приведено в комментариях к коду.

```
def create_object(shader):  # принимает программу шейдеров, возвращает вершинны массив
    vertex_array_object = GL.glGenVertexArrays(1)  # создание вершинного массива (количество объектов - 1)
    GL.glBindVertexArray(vertex_array_object)  # связывание вершинного массива с текущим контекстом (при связывании
    # все последующие операции с вершинными массивами будут использовать вершинные данные, указанные в этом массиве

    vertex_buffer = GL.glGenBuffers(1)  # генерация идентификаторов буферов для хранения данных в памяти видеокарты
    GL.glBindBuffer(GL.GL_ARRAY_BUFFER, vertex_buffer)  # связывание буфера с текущим контекстом

    position = GL.glGetAttribLocation(shader, 'position')  # получаем индекс атрибута вершины в шейдере по имени
    # (расположение вершин)
    GL.glEnableVertexAttribArray(position)  # принимает индекс атрибута вершины и включает использование массива
    # атрибутов вершин, связанных с этим индексом, в процессе рисования графических примитивов.
    GL.glVertexAttribPointer(position, 3, GL.GL_FLOAT, False, 0, ctypes.c_void_p(0))  # описывает формат массива
    # атрибутов вершин и указывает OpenGL, как использовать эти атрибуты при рисовании графических примитивов.
    # position - индекс атрибута вершины, для которого мы хотим описать формат массива
    # 3 - количество компонентов в атрибуте вершины (для x,y,z - 3)
    # GL.GL_FLOAT - тип данных компонентов в атрибуте
```

```
    # False - указывает, должны ли значения атрибутов быть нормализованы между 0 и 1
    # 0 - количество байт между двумя соседними атрибутами вершин
    # ctypes.c_void_p(0) - смещение от начала вершины до начала первого компонента атрибута вершины

    vs = vertices.tobytes()
    GL.glBufferData(GL.GL_ARRAY_BUFFER, len(vs), vs, GL.GL_STATIC_DRAW)  # для передачи вершинных данных в буфер объекта
    #отвязываем от контекста
    GL.glBindVertexArray(0)
    GL.glDisableVertexAttribArray(position)
    GL.glBindBuffer(GL.GL_ARRAY_BUFFER, 0)
    return vertex_array_object
```

в) Создание шейдерной программы и VAO

Используя две вышеописанные функции создаются шейдерная программа и VAO

```
self.shader = GL.shaders.compileProgram(  # создание шейдерной программы
    #  Функция shaders.compileProgram() компилирует заданные шейдеры и связывает их вместе для создания
    #  программы шейдеров. Она принимает два аргумента - объекты вершинного и фрагментного шейдеров
    compileShader(vertex_shader, GL.GL_VERTEX_SHADER),  # компилируем вершинный шейдер
    compileShader(fragment_shader, GL.GL_FRAGMENT_SHADER)  # компилируем фрагментный шейдер
)

self.vertex_array_object = create_object(self.shader)
```

г) Создания эффекта растворения

```
vertexColorLocation = GL.glGetUniformLocation(self.shader, "alpha")
if self.alpha != 0:
            self.alpha -= 0.001
GL.glUseProgram(self.shader)  # Вызов функции glUseProgram устанавливает программу шейдеров, которая
# будет использоваться для отрисовки графики на экране.
GL.glUniform1f(vertexColorLocation, self.alpha)
GL.glBindVertexArray(self.vertex_array_object)
GL.glDrawArrays(GL.GL_POLYGON, 0, len(vertices))  # отрисовка примитивов
GL.glBindVertexArray(0)
GL.glUseProgram(0)
```

Как уже было упомянуто ранее, для создания эффекта растворения необходимо изменять альфа канал изображения. Т.к. в фрагментном шейдере была указана переменная типа uniform (с именем) alpha, то здесь можно к ней обратиться и записать в нее новое значение (альфа канал изначально равен 1, каждую итерацию он уменьшается на 0.001, пока не станет равным 0 — т. е. пока изображение не станет прозрачным). Для этого с помощью библиотечной функции *glGetUniformLocation* извлекается однообразная переменная по имени, после чего с помощью библиотечной функции *glUniform1f* в нее записывает измененное значение альфа канала.

Здесь же происходит установка программы шейдеров, привязка VAO и отрисовка облака.

3. Прорисовка облака и демонстрация его «таяния»

Благодаря предыдущим пунктам, остается только создать массив с координатами вершин. Для этого была написана функция отрисовки круга:

```
def createCircle(shift_x, shift_y, R):
    global vertices
    steps = 2000
    angle = math.pi * 2 / steps

    for i in range(steps):
        newX = R * math.sin(angle * i) + shift_x
        newY = -R * math.cos(angle * i) + shift_y
        vertices = numpy.append(vertices, [newX, newY, 0]).astype(numpy.float32)
```

функция принимает смещения относительно начала координат и радиус круга, после чего полученные координаты добавляются в массив для отрисовки.

С помощью этой функции (т. е. с помощью кругов) было нарисовано облако (рис.1)

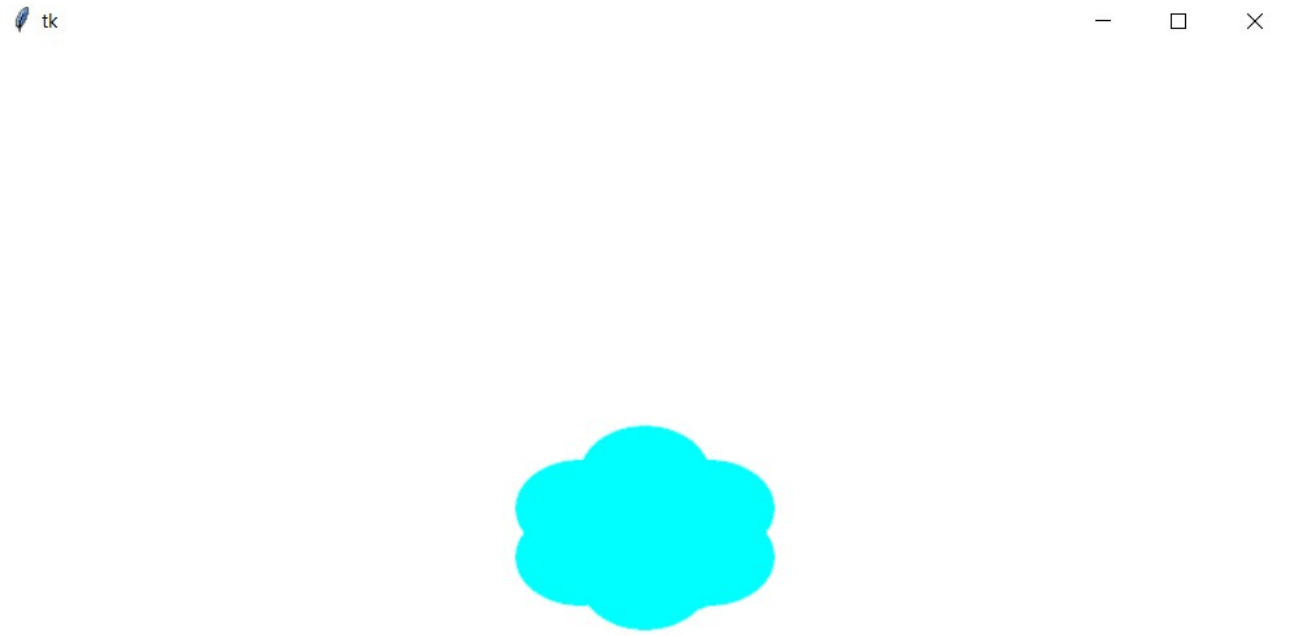

Рисунок 1 — Изначальное изображение облака

Как только пользователь запустит программу, облако сразу начнет «таять» (т. е. альфа канал будет уменьшаться) пока оно полностью не станет прозрачным

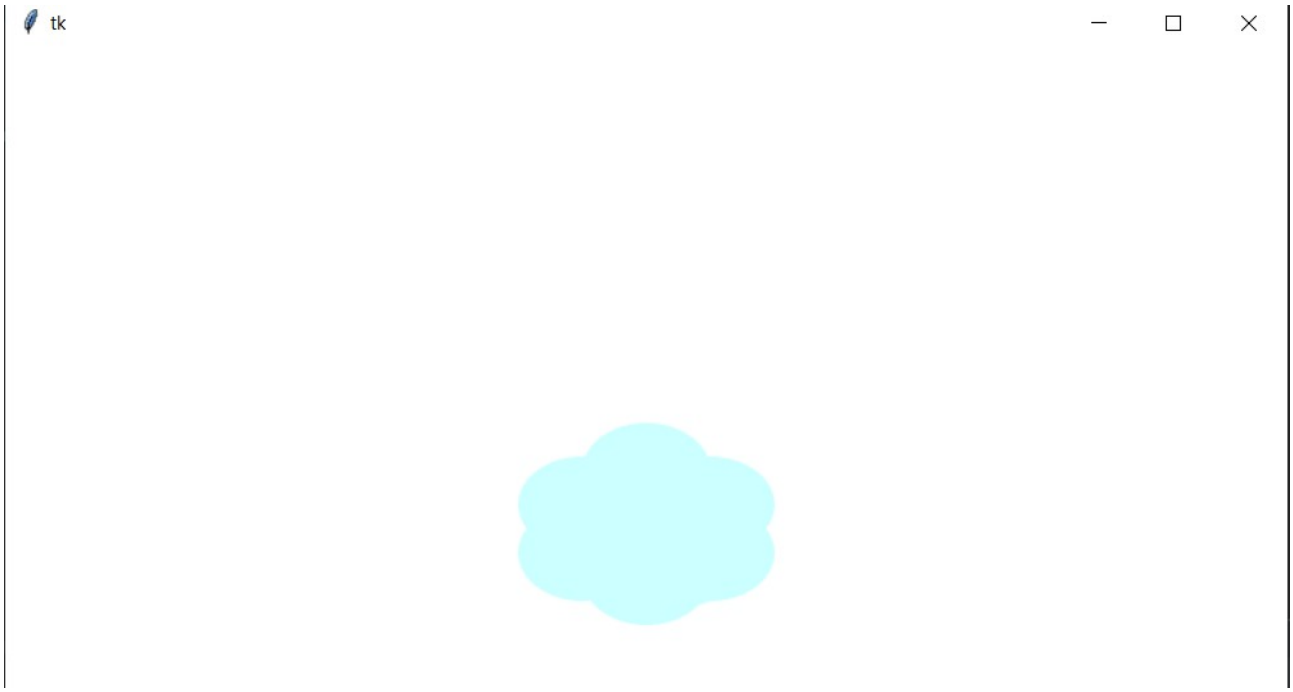

Рисунок 2 — Демонстрация эффекта растворения облака

Рисунок 3 — Почти полностью «растворенное» облако

Вывод.

Была разработана программа, реализующая эффект растворения с использованием шейдеров.

Приложение А. Исходный код.

Файл cloud.py

```
import math
from OpenGL import GL
import OpenGL.GL.shaders
import ctypes
import numpy
import pyopengltk
import tkinter as tk


def compileShader(source, shaderType):  # принимает (код шейдера, тип шейдера), возвращает скомпилированный шейдер
    shader = GL.glCreateShader(shaderType)  # создание объекта шейдера (принимает тип создаваемого шейдера)
    GL.glShaderSource(shader, source)  # загрузка кода в созданный объект шейдера
    GL.glCompileShader(shader)  # компиляция кода, загруженного ранее в объект шейдера
    return shader


vertex_shader = """
in vec3 position;
varying vec3 vertex_color;
void main()
{
  gl_Position = vec4(position, 1.0);
  vertex_color = vec4(0.0f, 1.0f, 1.0f, 1.0f);
}
"""

fragment_shader = """
varying vec3 vertex_color;
uniform float alpha;
void main()
{
  gl_FragColor = vec4(vertex_color, alpha);
}
"""


def create_object(shader):  # принимает программу шейдеров, возвращает вершинны массив
    vertex_array_object = GL.glGenVertexArrays(1)  # создание вершинного массива (количество объектов - 1)
    GL.glBindVertexArray(vertex_array_object)  # связывание вершинного массива с текущим контекстом (при связывании
    # все последующие операции с вершинными массивами будут использовать вершинные
```

```
данные, указанные в этом массиве

    vertex_buffer = GL.glGenBuffers(1)  # генерация идентификаторов буферов для хранения данных в памяти видеокарты
    GL.glBindBuffer(GL.GL_ARRAY_BUFFER, vertex_buffer)  # связывание буфера с текущим контекстом

    position = GL.glGetAttribLocation(shader, 'position')  # получаем индекс атрибута вершины в шейдере по имени
    # (расположение вершин)
    GL.glEnableVertexAttribArray(position)  # принимает индекс атрибута вершины и включает использование массива
    # атрибутов вершин, связанных с этим индексом, в процессе рисования графических примитивов.
    GL.glVertexAttribPointer(position, 3, GL.GL_FLOAT, False, 0, ctypes.c_void_p(0))  # описывает формат массива
    # атрибутов вершин и указывает OpenGL, как использовать эти атрибуты при рисовании графических примитивов.
    # position - индекс атрибута вершины, для которого мы хотим описать формат массива
    # 3 - количество компонентов в атрибуте вершины (для x,y,z - 3)
    # GL.GL_FLOAT - тип данных компонентов в атрибуте
    # False - указывает, должны ли значения атрибутов быть нормализованы между 0 и 1
    # 0 - количество байт между двумя соседними атрибутами вершин
    # ctypes.c_void_p(0) - смещение от начала вершины до начала первого компонента атрибута вершины

    vs = vertices.tobytes()
    GL.glBufferData(GL.GL_ARRAY_BUFFER, len(vs), vs, GL.GL_STATIC_DRAW)  # для передачи вершинных данных в буфер объекта
    #отвязываем от контекста
    GL.glBindVertexArray(0)
    GL.glDisableVertexAttribArray(position)
    GL.glBindBuffer(GL.GL_ARRAY_BUFFER, 0)
    return vertex_array_object


class ShaderFrame(pyopengltk.OpenGLFrame):

    def initgl(self):
        GL.glClearColor(1.0, 1.0, 1.0, 1.0)
        GL.glEnable(GL.GL_DEPTH_TEST)  # буфер глубины (z-координата) (для 3D)
        GL.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA)
        GL.glEnable(GL.GL_BLEND)

        self.shader = GL.shaders.compileProgram(  # создание шейдерной программы
            #  Функция shaders.compileProgram() компилирует заданные шейдеры и связывает их вместе для создания
            #  программы шейдеров. Она принимает два аргумента - объекты вершинного и
```

```
фрагментного шейдеров
        compileShader(vertex_shader, GL.GL_VERTEX_SHADER),  # компилируем вершинный
шейдер
        compileShader(fragment_shader, GL.GL_FRAGMENT_SHADER)  # компилируем
фрагментный шейдер
    )

    self.vertex_array_object = create_object(self.shader)
    self.alpha = 1

  def redraw(self):
    GL.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT)
    vertexColorLocation = GL.glGetUniformLocation(self.shader, "alpha")
    if self.alpha != 0:
      self.alpha -= 0.001

    GL.glUseProgram(self.shader)  # Вызов функции glUseProgram устанавливает программу
шейдеров, которая
    # будет использоваться для отрисовки графики на экране.
    GL.glUniform1f(vertexColorLocation, self.alpha)
    GL.glBindVertexArray(self.vertex_array_object)
    GL.glDrawArrays(GL.GL_POLYGON, 0, len(vertices))  # отрисовка примитивов
    GL.glBindVertexArray(0)
    GL.glUseProgram(0)


def createCircle(shift_x, shift_y, R):
  global vertices
  steps = 2000
  angle = math.pi * 2 / steps

  for i in range(steps):
    newX = R * math.sin(angle * i) + shift_x
    newY = -R * math.cos(angle * i) + shift_y
    vertices = numpy.append(vertices, [newX, newY, 0]).astype(numpy.float32)


root = tk.Tk()

vertices = numpy.array([[0, 0, 0.0]]).astype(numpy.float32)

createCircle(0, 0, 0.1)
createCircle(0, 0.12, 0.1)
createCircle(0.1, 0.05, 0.1)
createCircle(0.1, -0.05, 0.1)
createCircle(0, -0.1, 0.1)
createCircle(-0.1, 0.05, 0.1)
createCircle(-0.1, -0.05, 0.1)
```

```
app = ShaderFrame(root, width=800, height=600)
app.pack(fill=tk.BOTH, expand=tk.YES)
app.after(100, app.printContext)
app.animate = 1000 // 60
app.animate = 1
app.mainloop()
```