МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный электротехнический университет « ЛЭТИ» им. В. И.Ульянова(Ленина)

Задание для лабораторной работы № 5

" Расширения OpenGL, программируемый графический конвейер. Шейдеры.”

Студенты гр. 1384 _____ Усачева Д.В.
Пчелинцева К.Р.

Руководитель _____ Герасимова Т.В.

Санкт-Петербург

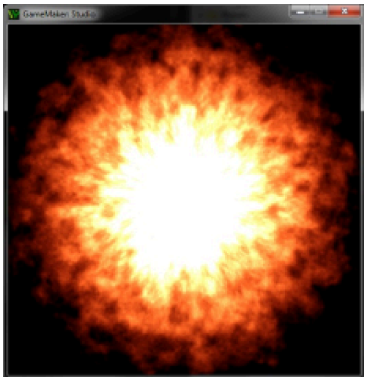
2024 г.

Задание

Разработать визуальный эффект по заданию, реализованный средствами языка шейдеров GLSL.

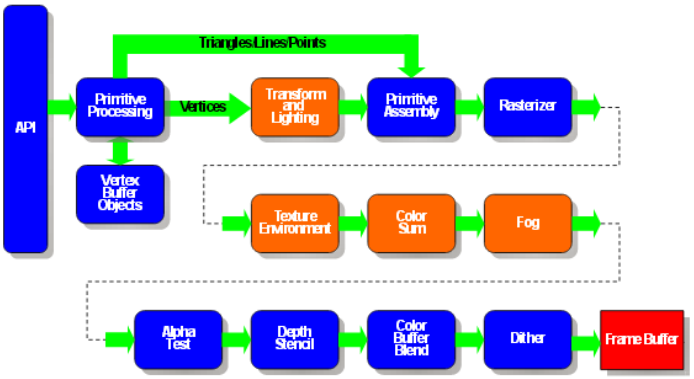
12. Эффект огненного вакуума/фаерболла (Fire Vacuum)

Смещает пиксели от краёв к середине, создавая эффект "всасывания"огня.

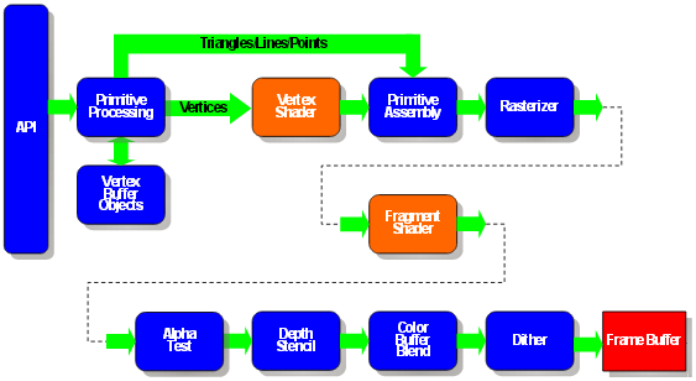


Общие теоретические сведения

Конвейер с фиксированной функциональностью



Программированный графический конвейер



Программируемый графический конвейер позволяет обойти фиксированную функциональность стандартного графического конвейера OpenGL.

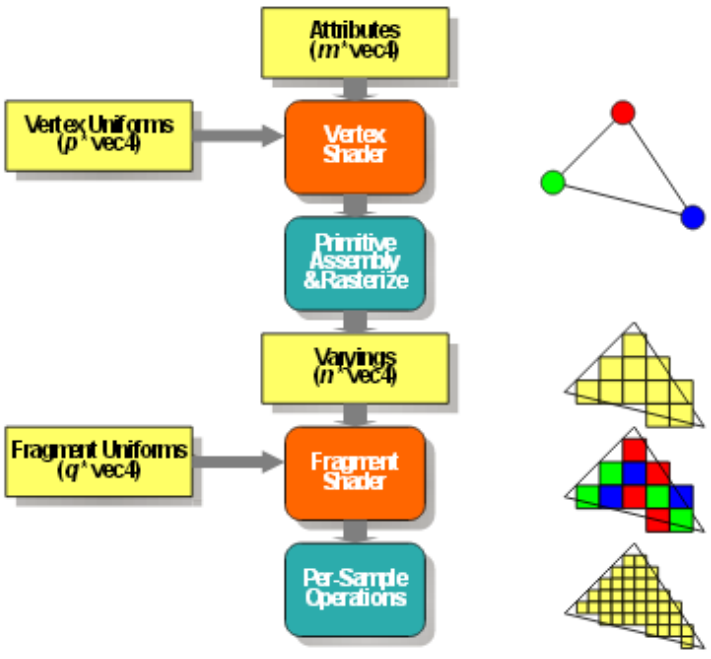
Для вершин – задать необычное преобразование вершин (обычное – это просто умножение координат вершин на модельную и видовую матрицу). Типичное применение - скелетная анимация, анимация волн.

Для геометрических примитивов, таких как треугольник, позволяет сформировать несколько иную геометрию, чем было, например, разбить треугольник на несколько более мелких.

Для фрагментов – позволяет определить цвет фрагмента (пикселя) в обход стандартных моделей освещения. Например, реализовать процедурные текстуры: дерева или мрамора.

Программа, используемая для расширения фиксированной функциональности OpenGL называется **шейдер,** соответственно различают 3 типа шейдеров вершинный, геометрический и фрагментный

Программируемая модель



Совместно с библиотекой OpenGL, могут быть использованы шейдеры, написанные на языке высокого и низкого уровня. Код шейдеров на языке низкого уровня сходен с кодом ассемблера, однако в действительности вы не кодируете на уровне ассемблера, поскольку каждый производитель аппаратного обеспечения предлагает уникальную структуру графического процессора с собственным представлением инструкций и наборов команд. Все эти процессоры вводят собственные пределы числа регистров констант и команд. Низкоуровневые расширения можно назвать наименьшим общим знаменателем функциональных возможностей, доступных у всех производителей.

Программирование графических процессоров на языке высокого уровня означает меньше кода, более читабельный вид, а, следовательно, более высокую производительность труда.

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.

Разные шаги графического конвейера накладывают разные ограничения на работу шейдеров. Поэтому у каждого типа шейдеров есть своя специфика.

Геометрический и тессеяляционные шейдеры не являются обязательными. Современный OpenGL требует наличия только вершинного и фрагментного шейдера. Хотя существует сценарий, при котором фрагментный шейдер может отсутствовать

Выполнение работы.

Программа, реализующая поставленную задачу, была написана на языке программирования Python 3.10. Интерфейс программы был написан с помощью PyQt6, была использована программа QtDesigner для создания окна.

Приложение включает в себя окно, в котором отображается эффект огненного вакуума. Был написан вершинный шейдер, который принимает двумерные координаты вершины и преобразует их в четырехмерные координаты, устанавливая z в 0.0 и w в 1.0. Это позволяет представить вершину в трехмерном пространстве и передать координаты для дальнейшего рендеринга объекта на экране. А также был написан фрагментный шейдер, который использует шум Перлина для создания текстурированного эффекта. Он принимает на вход текущее время и разрешение экрана, а затем использует сложные математические вычисления для определения цвета каждого пикселя на экране.

Листинг 1 — Шейдеры

```
vertex_shader_source = """
#version 330
```

```
in vec2 position;
void main() {
    gl_Position = vec4(position, 0.0, 1.0);
}
"""

fragment_shader_source = """
#version 330
out vec4 fragColor;

uniform float time;
uniform vec2 resolution;

float snoise(vec3 uv, float res)
{
    const vec3 s = vec3(1e0, 1e2, 1e3);
    uv *= res;
    vec3 uv0 = floor(mod(uv, res)) * s;
    vec3 uv1 = floor(mod(uv + vec3(1.0), res)) * s;
    vec3 f = fract(uv);
    f = f * f * (3.0 - 2.0 * f);
    vec4 v = vec4(uv0.x + uv0.y + uv0.z, uv1.x + uv0.y + uv0.z,
                  uv0.x + uv1.y + uv0.z, uv1.x + uv1.y + uv0.z);

    vec4 r = fract(sin(v * 1e-1) * 1e3);
    float r0 = mix(mix(r.x, r.y, f.x), mix(r.z, r.w, f.x), f.y);
    r = fract(sin((v + uv1.z - uv0.z) * 1e-1) * 1e3);
    float r1 = mix(mix(r.x, r.y, f.x), mix(r.z, r.w, f.x), f.y);
    return mix(r0, r1, f.z) * 2.0 - 1.0;
}

void mainImage(out vec4 fragColor, in vec2 fragCoord)
{
    vec2 p = -0.5 + fragCoord.xy / resolution.xy;
    p.x *= resolution.x / resolution.y;
    float color = 3.0 - (3.0 * length(2.0 * p));
     vec3 coord = vec3(atan(p.x, p.y) / 6.2832 + 0.5, length(p) * 0.4,
0.5);
    for (int i = 1; i <= 7; i++)
    {
        float power = pow(2.0, float(i));
         color += (1.5 / power) * snoise(coord + vec3(0.0, time * 0.05,
time * 0.01), power * 16.0);
    }
        fragColor = vec4(color, pow(max(color, 0.0), 2.0) * 0.4,
pow(max(color, 0.0), 3.0) * 0.15, 1.0);
}

void main()
{
    mainImage(fragColor, gl_FragCoord.xy);
}
"""
```

Тестирование.

Запуск программы представлен на рисунке 1.

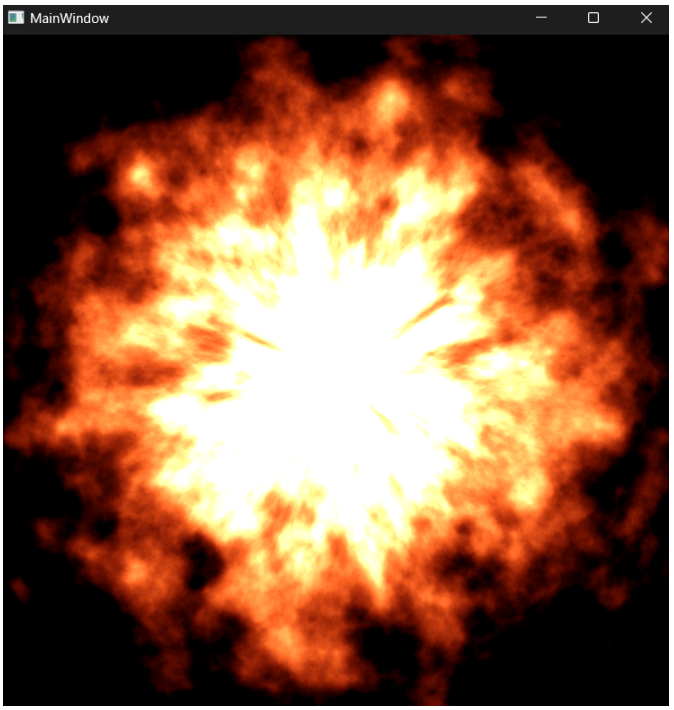


Рисунок 1 — Запуск программы

Вывод

В результате выполнения лабораторной работы была разработана программа, создающая эффект огненного вакуума, который был написан на языке шейдеров GLSL. При выполнении работы были приобретены навыки работы с графической библиотекой OpenGL.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

design.py

```
from controller import MyGLWidget
from PyQt6 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(600, 600)
        self.centralwidget = QtWidgets.QWidget(parent=MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.widget = MyGLWidget(parent=self.centralwidget)
        self.widget.setGeometry(QtCore.QRect(0, 0, 600, 600))
        self.widget.setObjectName("openGLWidget")
        MainWindow.setCentralWidget(self.centralwidget)
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec())
```

controller.py

```
import numpy as np
from OpenGL.GL import *
from OpenGL.GLUT import *
from PyQt6.QtCore import QTimer
from PyQt6.QtOpenGLWidgets import QOpenGLWidget
from OpenGL.GL import shaders

class MyGLWidget(QOpenGLWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.program = None
        self.time = 0.0
        self.width = 750
        self.height = 750
```

```
    def initializeGL(self):
        glClearColor(0.0, 0.0, 0.0, 1.0)
        glEnable(GL_DEPTH_TEST)
        self.create_shaders()

    def paintGL(self):
        self.time += 0.01  # Обновляем время для эффекта огня
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glUseProgram(self.program)  # Активируем программу шейдеров
        glUniform1f(glGetUniformLocation(self.program, "time"),
self.time)  # Устанавливаем uniform переменную
        # Устанавливаем uniform переменную для размера экрана
        glUniform2f(glGetUniformLocation(self.program, "resolution"),
self.width, self.height)

        # Рисуем квадрат
        glBegin(GL_QUADS)
        glVertex3f(-1.0, -1.0, 0.0)
        glVertex3f(1.0, -1.0, 0.0)
        glVertex3f(1.0, 1.0, 0.0)
        glVertex3f(-1.0, 1.0, 0.0)
        glEnd()
        self.update()

    def create_shaders(self):
        vertex_shader = shaders.compileShader(vertex_shader_source,
GL_VERTEX_SHADER)
        fragment_shader = shaders.compileShader(fragment_shader_source,
GL_FRAGMENT_SHADER)
        self.program = shaders.compileProgram(vertex_shader,
fragment_shader)

        if not self.program:
            error = glGetProgramInfoLog(self.program)
            print(f"Ошибка при создании программы шейдеров: {error}")
            return
```