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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Компьютерная графика»
Тема: «Построение фракталов»

Студент гр. 6381 _____ Фиалковский М.С.

Студент гр. 6381 _____ Афийчук И.И.

Преподаватель _____ Герасимова Т.В.

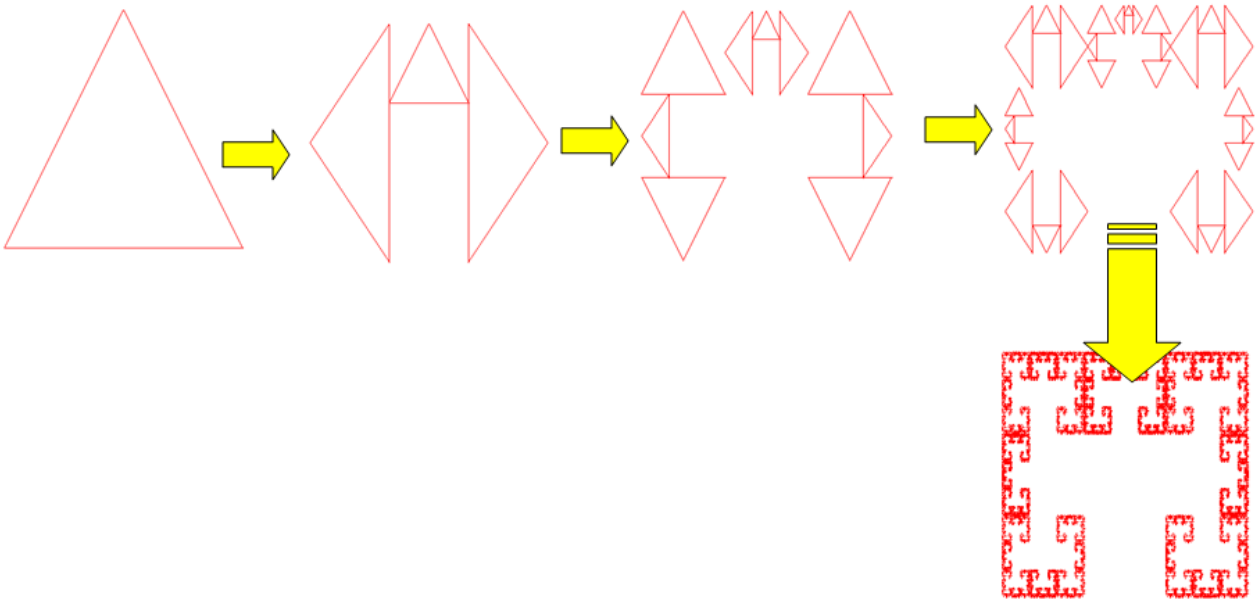
Санкт-Петербург
2019

Задание.

Реализовать программу, отображающую фрактал по индивидуальному заданию с помощью библиотеки OpenGL.

Задание 21

IFS-фракталы “Треугольники Кантора”



Общие сведения.

Фрактал— множество, обладающее свойством самоподобия (объект, в точности или приближённо совпадающий с частью себя самого, то есть целое имеет ту же форму, что и одна или более частей).

Существует большое число математических объектов называемых фракталами (треугольник Серпинского, снежинка Коха, кривая Пеано, множество Мандельброта). Фракталы с большой точностью описывают многие физические явления и образования реального мира: горы, облака, турбулентные (вихревые) течения, корни, ветви и листья деревьев, кровеносные сосуды, что далеко не соответствует простым геометрическим фигурам.

Ход работы.

Сборка и компиляция программы осуществляется с помощью утилиты cmake и компилятора gcc из пакета MinGW. Графический интерфейс выполнен с помощью библиотеки FLTK. В качестве обертки над библиотекой OpenGL, используется GLEW. FLTK также имеет интерфейс для обращения к командам OpenGL.

Логика построения и работы программы во многом схожа с 1 и 2 работами.

Первая фигура фрактала – равнобедренный треугольник – имеет три вершины. В каждый момент времени все вершины фрактала хранятся в одном массиве (в нашем случае с контейнере std::vector). Все дальнейшие манипуляции будут изменять именно его.

На каждом шаге глубины фрактала фигура, поданная на вход копируется в ещё 2 новых фигуры.

Первая фигура уменьшается в 3 раза и поднимается чуть вверх по оси Y.

Вторая фигура поворачивается на 90 градусов влево, сжимается в 3 раза вдоль оси X и двигается влево.

Третья фигура поворачивается на 90 градусов вправо, сжимается в 3 раза вдоль оси X и двигается вправо.

Далее снова вызывается это же функция, но с уменьшенной на 1 глубиной. Если переданное значение глубины достигло 0, то программа рисует получившийся набор вершин и выходит из функции.

Преобразования вектора вершин происходит с помощью умножения на матрицы сдвига, поворота и масштабирования, причём важно это делать в правильном порядке. Для совершения данных манипуляций используется библиотека glm.

Например, команда

```
glm::mat4 transformMatrix = glm::mat4(1.0)
transformMatrix = glm::scale(transformMatrix, glm::vec3(1.0, 1.0/3, 1.0))
```

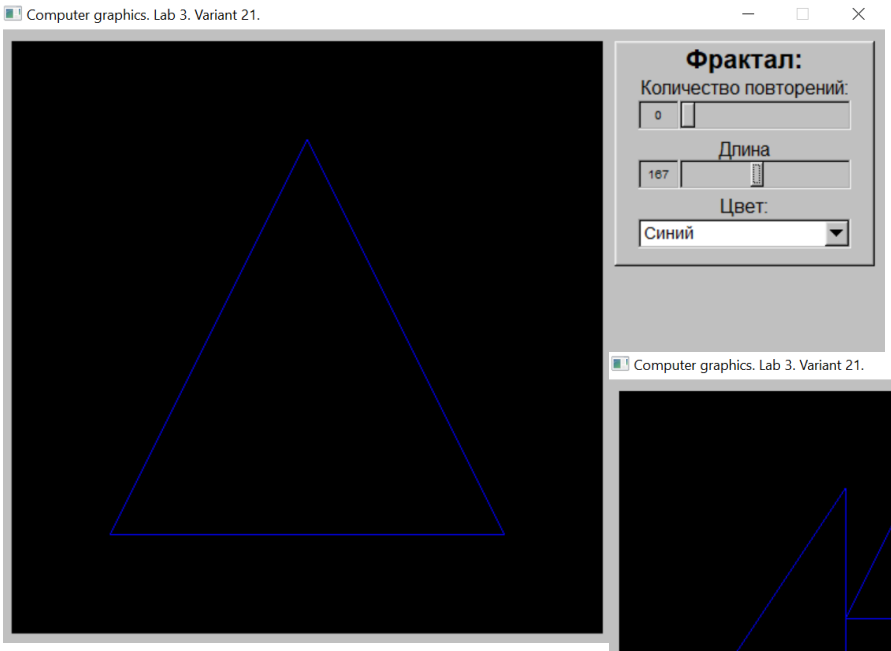
Создает единичную матрицу и умножает её на матрицу, полученную для операции масштабирования (сжатия в 3 раза по оси Y). Теперь мы можем умножить массив вершин на данную матрицу, тем самым получив сжатую фигуру (здесь *verteces* имеет тип *std::vector<glm::vec2>*):

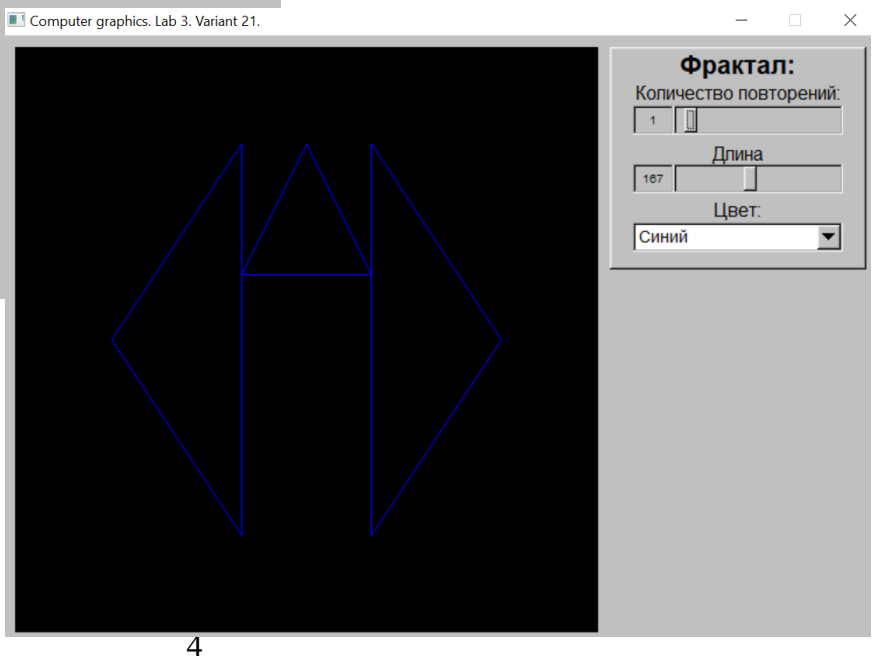
```
for (auto& i : verteces) {
    auto tmp = (transformMatrix * glm::vec4(i, 0.0f, 1.0f));
    i.x = tmp.x;
    i.y = tmp.y;
}
```

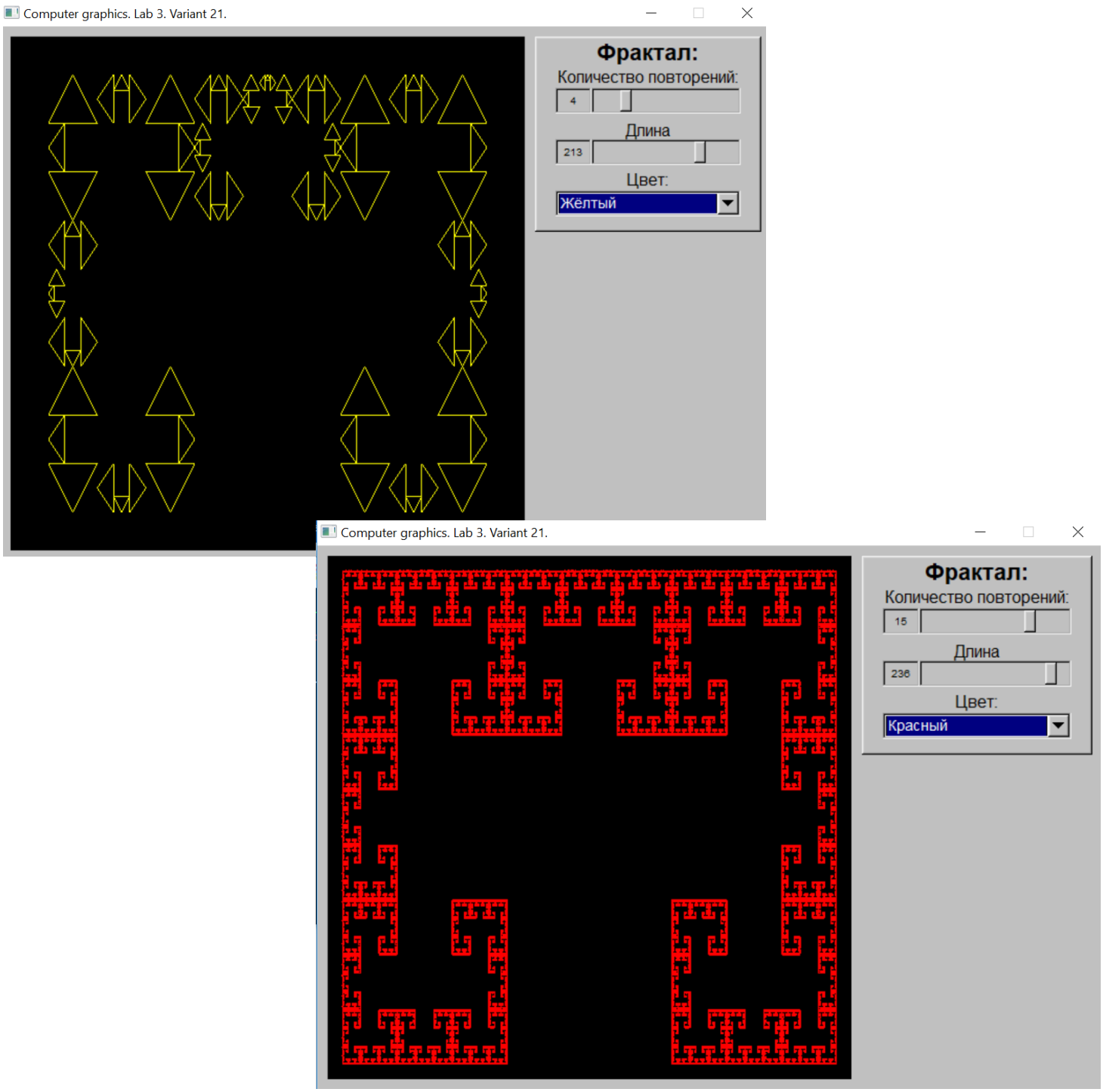
Тестирование.

Справа от области отрисовки находятся элементы UI, с помощью которых можно настраивать параметры выводимого фрактала. Все изменения применяются автоматически при изменении.

Нулевой и первый шаг в полученной программе выглядит следующим образом:







Вывод.

В процессе выполнения лабораторной работы была разработана программа, реализующую геометрическую форму «фрактал» заданного типа. При выполнении работы были приобретены навыки работы с графической библиотекой OpenGL, FLTK и GLM.

Исходный код метода Draw

```
void GlSubWin::draw() {
    if (!valid()){ // first time? init
        valid(1);
        FixViewport(w(), h());
    }
    glPolygonMode(GL_FRONT, GL_LINE);
    glClearColor(0,0,0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    applyColor(statePtr->getElemColor());

    sideLength = (GLdouble)statePtr->getLength();
    std::cout << statePtr->getLength() << "\n";
    std::vector<glm::vec2> triangleVertices = {
        { 0,  sideLength},
        {-sideLength, -sideLength},
        { sideLength, -sideLength}
    };
    try {
        Fractal(triangleVertices, statePtr->getDeep());
    }
    catch (const std::bad_alloc&) {
        statePtr->badAllocTrigger();
        return;
    }
}

void GlSubWin::Fractal(std::vector<glm::vec2> &verteces, int deep){
    if (deep == 0){
        glm::mat4 trans = glm::translate(glm::mat4(1.0), glm::vec3(w()/2, h()/2,
0.0f));
        glBegin(GL_TRIANGLES);
            for (auto &i :verteces){
                auto tmp = (trans * glm::vec4(i, 0.0f, 1.0f));
                i.x = tmp.x;
                i.y = tmp.y;
                glVertex2d(i.x, i.y);
            }
        glEnd();
        return;
    }

    auto left = verteces;
    auto right = verteces;

    figureCenter(verteces); // instead of center
    figureLeft(left);
    figureRight(right);
```

```
    verteces.insert(std::end(verteces), std::begin(left), std::end(left));
    verteces.insert(std::end(verteces), std::begin(right), std::end(right));
    left.clear();
    left.shrink_to_fit();
    right.clear();
    right.shrink_to_fit();
    Fractal(verteces, deep-1);
}

void GlSubWin::figureCenter(std::vector<glm::vec2> &verteces){
    glm::mat4 transformMatrix = glm::mat4(1.0);
    transformMatrix = glm::translate(transformMatrix, glm::vec3(0, sideLength*2/3,
0)); // Second
    transformMatrix = glm::scale(transformMatrix, glm::vec3(1.0/3, 1.0/3, 1)); //
First

    for (auto& i : verteces) {
        auto tmp = (transformMatrix * glm::vec4(i, 0.0f, 1.0f));
        i.x = tmp.x;
        i.y = tmp.y;
    }
}

void GlSubWin::figureLeft(std::vector<glm::vec2> &verteces){
    glm::mat4 transformMatrix = glm::mat4(1.0); // Единичная матрица.
    transformMatrix = glm::translate(transformMatrix, glm::vec3(-sideLength*2/3, 0,
0)); // Третье
    transformMatrix = glm::rotate(transformMatrix, glm::radians(90.0f),
glm::vec3(0.0f,0.0f,1.0f)); // Второе
    transformMatrix = glm::scale(transformMatrix, glm::vec3(1.0, 1.0/3, 1.0)); //
Первое

    for (auto& i : verteces) {
        auto tmp = (transformMatrix * glm::vec4(i, 0.0f, 1.0f)); // Умножаем
        i.x = tmp.x;
        i.y = tmp.y;
    }
}

void GlSubWin::figureRight(std::vector<glm::vec2> &verteces){
    glm::mat4 transformMatrix = glm::mat4(1.0);

    transformMatrix = glm::translate(transformMatrix, glm::vec3(sideLength*2/3, 0,
0)); // Third
    transformMatrix = glm::rotate(transformMatrix, glm::radians(-90.0f),
glm::vec3(0.0f,0.0f,1.0f)); // Second
    transformMatrix = glm::scale(transformMatrix, glm::vec3(1.0, 1.0/3, 1.0)); //
First

    for (auto& i : verteces) {
```

```
        auto tmp = (transformMatrix * glm::vec4(i, 0.0f, 1.0f));
        i.x = tmp.x;
        i.y = tmp.y;
    }
}

void GlSubWin::drawUpdated(){
    redraw();
}

void GlSubWin::applyColor(ElemColor color, double alpha){
    switch (color){
        case ElemColor::red:     glColor4f(1.0f, 0.0f, 0.0f, alpha); break;
        case ElemColor::green:   glColor4f(0.0f, 1.0f, 0.0f, alpha); break;
        case ElemColor::blue:    glColor4f(0.0f, 0.0f, 1.0f, alpha); break;
        case ElemColor::magneta: glColor4f(1.0f, 0.0f, 1.0f, alpha); break;
        case ElemColor::cyan :   glColor4f(0.0f, 1.0f, 1.0f, alpha); break;
        case ElemColor::yellow:  glColor4f(1.0f, 1.0f, 0.0f, alpha); break;
        case ElemColor::random: {
            auto r = static_cast<float>(std::rand() % 256) / 256;
            auto g = static_cast<float>(std::rand() % 256) / 256;
            auto b = static_cast<float>(std::rand() % 256) / 256;
            glColor4f(r,g,b, alpha);
            break;
        }
        default: assert("Invalid switch statement!\n" == nullptr); break;
    }
}
```