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

Компьютерная графика:
лабораторный практикум.

Лабораторная работа № 3

"Построение фракталов"

Вариант 52

Студент гр. 5383 _____ Допира В. Е.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург
2018

Лабораторная работа № 3

Задание

На базе предыдущей лабораторной работы разработать программу реализующую фрактал по индивидуальному заданию.

Общие сведения

Фрактал (лат. fractus — дробленый) — термин, означающий геометрическую фигуру, обладающую свойством самоподобия, то есть составленную из нескольких частей, каждая из которых подобна всей фигуре целиком.

Геометрические фракталы

Фракталы этого класса самые наглядные. В двухмерном случае их получают с помощью ломаной (или поверхности в трехмерном случае), называемой генератором. За один шаг алгоритма каждый из отрезков, составляющих ломаную, заменяется на ломаную-генератор в соответствующем масштабе. В результате бесконечного повторения этой процедуры получается геометрический фрактал.

Алгебраические фракталы

Это самая крупная группа фракталов. Получают их с помощью нелинейных процессов в n-мерных пространствах. Наиболее изучены двухмерные процессы.

Стохастические (случайные) фракталы

Еще одним известным классом фракталов являются стохастические фракталы, которые получаются в том случае, если в итерационном процессе хаотически менять какие-либо его параметры. При этом получаются объекты очень похожие на природные - несимметричные деревья, изрезанные береговые линии и т.д. Двумерные стохастические фракталы используются при моделировании рельефа местности и поверхности моря. Примерами стохастических фракталов являются фрактальные кривые, возникающие в критических двумерных моделях

статистической механики, траектория броуновского движения на плоскости и в пространстве, плазма.

L-система

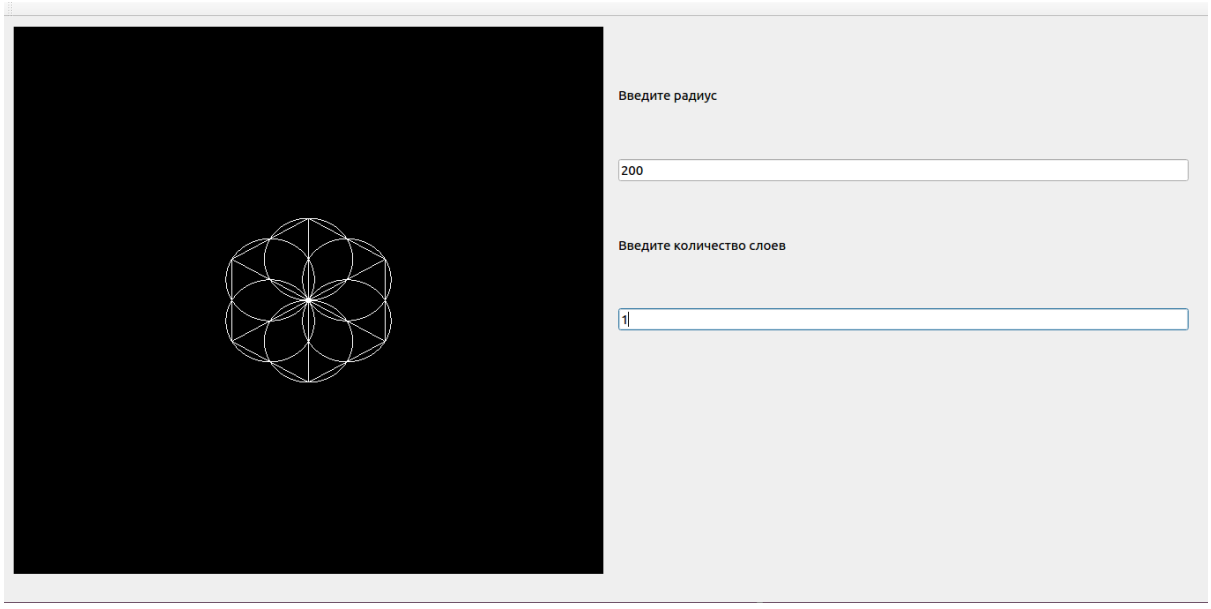
L-система - это грамматика некоторого языка (достаточно простого), которая описывает инициатор и преобразование, выполняемое над ним, при помощи средств, аналогичных средствам языка Лого (аксиоматическое описание простейших геометрических фигур и допустимых преобразований на плоскости и в пространстве).

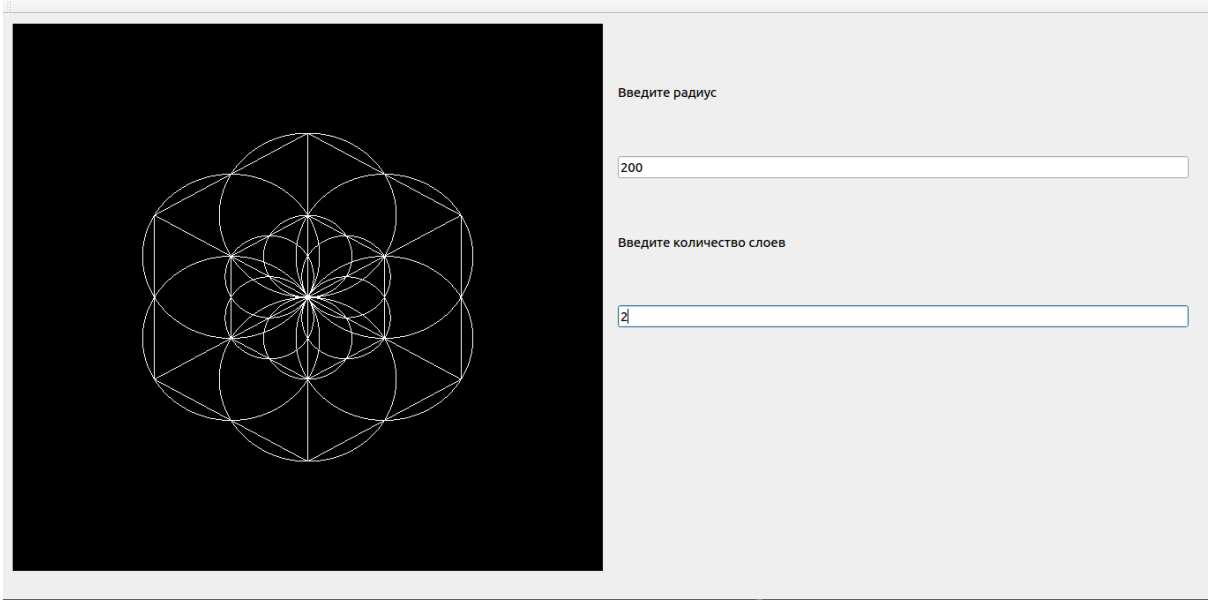
Система итерирующих функций IFC

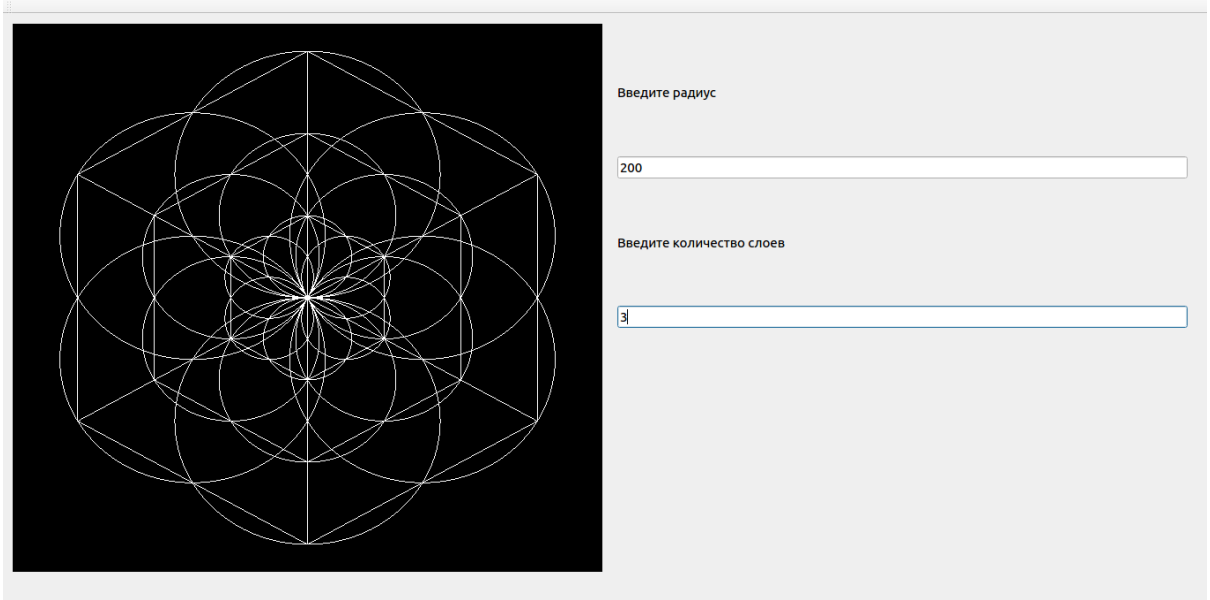
Система итерирующих функций IFC Применение таких преобразований, которые дают ту фигуру которую необходимо. Система итерирующих функций - это совокупность сжимающих аффинных преобразований. Как известно, аффинные преобразования включают в себя масштабирование, поворот и параллельный перенос. Аффинное преобразование считается сжимающим, если коэффициент масштабирования меньше единицы.

Тестирование

Результаты тестирования представлены на снимках экрана.







Вывод

В результате выполнения лабораторной работы разработана программа, реализующая представление заданного фрактала с помощью системы функций.

Приложение. Код генератора фрактала

```
GL_Widget::GL_Widget(QWidget *parent):
    QGLWidget(parent)
{
    setGeometry(20, 20, 550, 500);
}
void GL_Widget::initializeGL(){
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void GL_Widget::paintGL(){
    glClear(GL_COLOR_BUFFER_BIT);
    //Задаем режим матрицы
    glMatrixMode(GL_PROJECTION);
    //Загружаем матрицу
    glLoadIdentity();
    move();
    glScalef(m_scale, m_scale, m_scale);
    drawFractal(m_count);
}
void GL_Widget::drawFractal(int count)
{
    glViewport(0, 0, this->width(), this->height());
    for(int i = 1; i <= count; i++)
    {
        drawCircle(m_startPoint.m_x, m_startPoint.m_y + i * m_radius / 2, i * m_radius / 2, 300);
        drawCircle(m_startPoint.m_x, m_startPoint.m_y - i * m_radius / 2, i * m_radius / 2, 300);
        drawCircle(m_startPoint.m_x + i * sqrt(3) * m_radius / 4, m_startPoint.m_y + i * m_radius / 4, i
* m_radius / 2, 300);
        drawCircle(m_startPoint.m_x - i * sqrt(3) * m_radius / 4, m_startPoint.m_y + i * m_radius / 4, i
* m_radius / 2, 300);
        drawCircle(m_startPoint.m_x + i * sqrt(3) * m_radius / 4, m_startPoint.m_y - i * m_radius / 4, i
* m_radius / 2, 300);
        drawCircle(m_startPoint.m_x - i * sqrt(3) * m_radius / 4, m_startPoint.m_y - i * m_radius / 4, i *
m_radius / 2, 300);
        drawLine(m_startPoint.m_x, m_startPoint.m_y, m_startPoint.m_x, m_startPoint.m_y + i *
m_radius);
        drawLine(m_startPoint.m_x, m_startPoint.m_y, m_startPoint.m_x, m_startPoint.m_y - i *
m_radius);
        drawLine(m_startPoint.m_x, m_startPoint.m_y, m_startPoint.m_x + i * sqrt(3) * m_radius / 2,
m_startPoint.m_y + i * m_radius / 2);
        drawLine(m_startPoint.m_x, m_startPoint.m_y, m_startPoint.m_x + i * sqrt(3) * m_radius / 2,
m_startPoint.m_y - i * m_radius / 2);
        drawLine(m_startPoint.m_x, m_startPoint.m_y, m_startPoint.m_x - i * sqrt(3) * m_radius / 2,
m_startPoint.m_y + i * m_radius / 2);
        drawLine(m_startPoint.m_x, m_startPoint.m_y, m_startPoint.m_x - i * sqrt(3) * m_radius / 2,
m_startPoint.m_y - i * m_radius / 2);
        drawLine(m_startPoint.m_x, m_startPoint.m_y + i * m_radius, m_startPoint.m_x + i * sqrt(3) *
m_radius / 2, m_startPoint.m_y + i * m_radius / 2);
        drawLine(m_startPoint.m_x, m_startPoint.m_y - i * m_radius, m_startPoint.m_x + i * sqrt(3) *
m_radius / 2, m_startPoint.m_y - i * m_radius / 2);
        drawLine(m_startPoint.m_x, m_startPoint.m_y + i * m_radius, m_startPoint.m_x - i * sqrt(3) *
m_radius / 2, m_startPoint.m_y + i * m_radius / 2);
```

```
        drawLine(m_startPoint.m_x, m_startPoint.m_y - i * m_radius, m_startPoint.m_x - i * sqrt(3) *
m_radius / 2, m_startPoint.m_y - i * m_radius / 2);
        drawLine(m_startPoint.m_x + i * sqrt(3) * m_radius / 2, m_startPoint.m_y + i * m_radius / 2,
m_startPoint.m_x + i * sqrt(3) * m_radius / 2, m_startPoint.m_y - i * m_radius / 2);
        drawLine(m_startPoint.m_x - i * sqrt(3) * m_radius / 2, m_startPoint.m_y + i * m_radius / 2,
m_startPoint.m_x - i * sqrt(3) * m_radius / 2, m_startPoint.m_y - i * m_radius / 2);
    }
}
void GL_Widget::drawCircle(float x, float y, float r, int amountSegments)
{
    glBegin(GL_LINE_LOOP);
    for(int i = 0; i < amountSegments; i++)
    {
    float angle = 2.0 * 3.1415926 * float(i) / float(amountSegments);
    float dx = r * cosf(angle);
    float dy = r * sinf(angle);
    glVertex2f(x + dx, y + dy);
    }
    glEnd();
}
void GL_Widget::drawLine(float x, float y, float a, float b)
{
    glBegin(GL_LINES);
        glVertex2d(x, y);
        glVertex2d(a, b);
    glEnd();
}
void GL_Widget::wheelEvent(QWheelEvent *wheelEvent)
{
    scaling(wheelEvent->delta());
}
void GL_Widget::scaling(int delta)
{// если колесико вращаем вперед -- умножаем переменную масштаба на 1.1
    if (delta > 0)
    {
        m_scale *= 1.1;
    }
    else // иначе - делим на 1.1
    {
        if (delta < 0)
        {
            m_scale /= 1.1;
        }
    }
    updateGL();
}
void GL_Widget::move()
{
    glTranslatef(0.5f * m_positionX, -(0.5f * m_positionY), 0);
}
void GL_Widget::mouseMoveEvent(QMouseEvent *mouseEvent)
{
    double dx = (mouseEvent->x() - m_mousePositionX) / 10;
    double dy = (mouseEvent->y() - m_mousePositionY) / 10;
    if (mouseEvent->buttons() == Qt::LeftButton)
    {
```

```
        setPositionX(m_positionX + dx/1000);
        setPositionY(m_positionY + dy/1000);
    }
    updateGL();
}
void GL_Widget::mousePressEvent(QMouseEvent *mouseEvent)
{
    m_mousePositionX = mouseEvent->x();
    m_mousePositionY = mouseEvent->y();
}
void GL_Widget::setPositionX(double value)
{
    m_positionX = value;
}
void GL_Widget::setPositionY(double value)
{
    m_positionY = value;
}
void GL_Widget::setStartPoint(double x, double y)
{
    m_startPoint = Point2Df(x, y);
    updateGL();
}
void GL_Widget::setRadius(int value)
{
    m_radius = value;
    updateGL();
}
void GL_Widget::setCount(int value)
{
    m_count = value;
    updateGL();
}
size_t GL_Widget::radius() const
{
    return m_radius;
}
int GL_Widget::count() const
{
    return m_count;
}
std::pair<double, double> GL_Widget::startPoint() const
{
    return std::make_pair(m_startPoint.m_x, m_startPoint.m_y);
}
```