**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

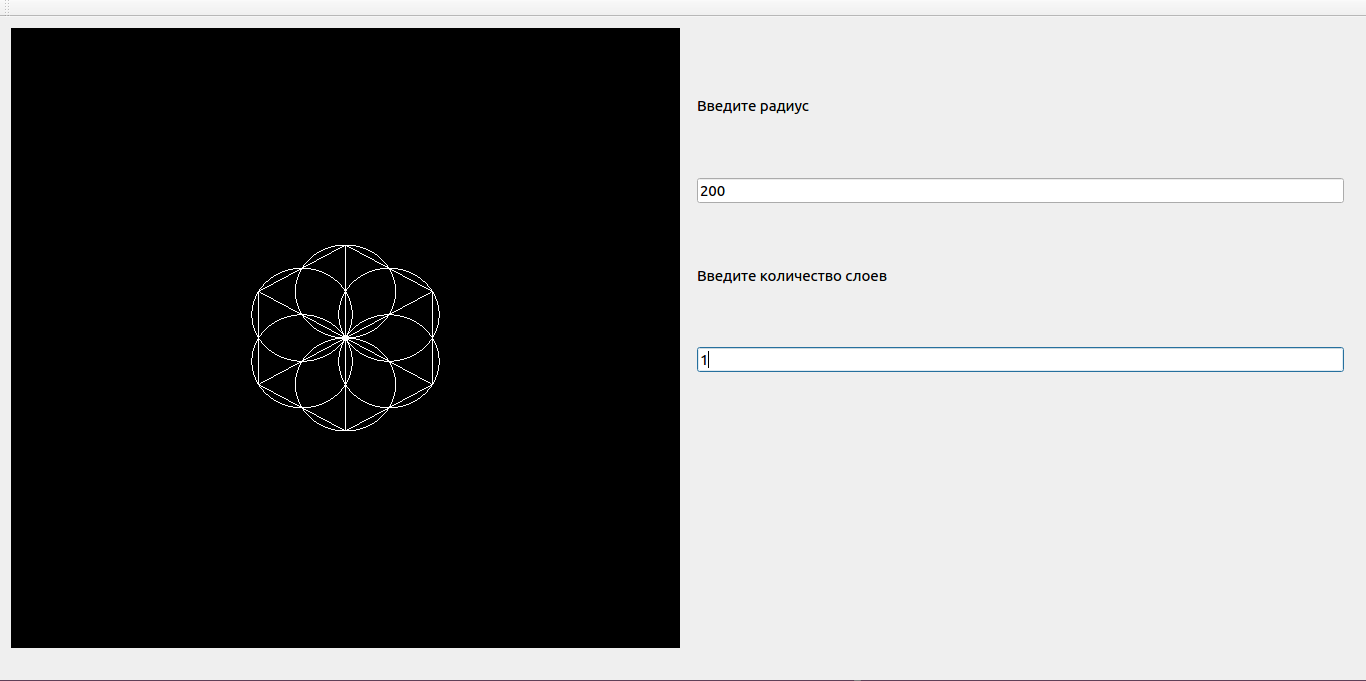
L-система - это грамматика некоторого языка (достаточно простого), которая описывает инициатор и преобразование, выполняемое над ним, при помощи средств, аналогичных средствам языка Лого (аксиоматическое описание простейших геометрических фигур и допустимых преобразований на плоскости и в пространстве).

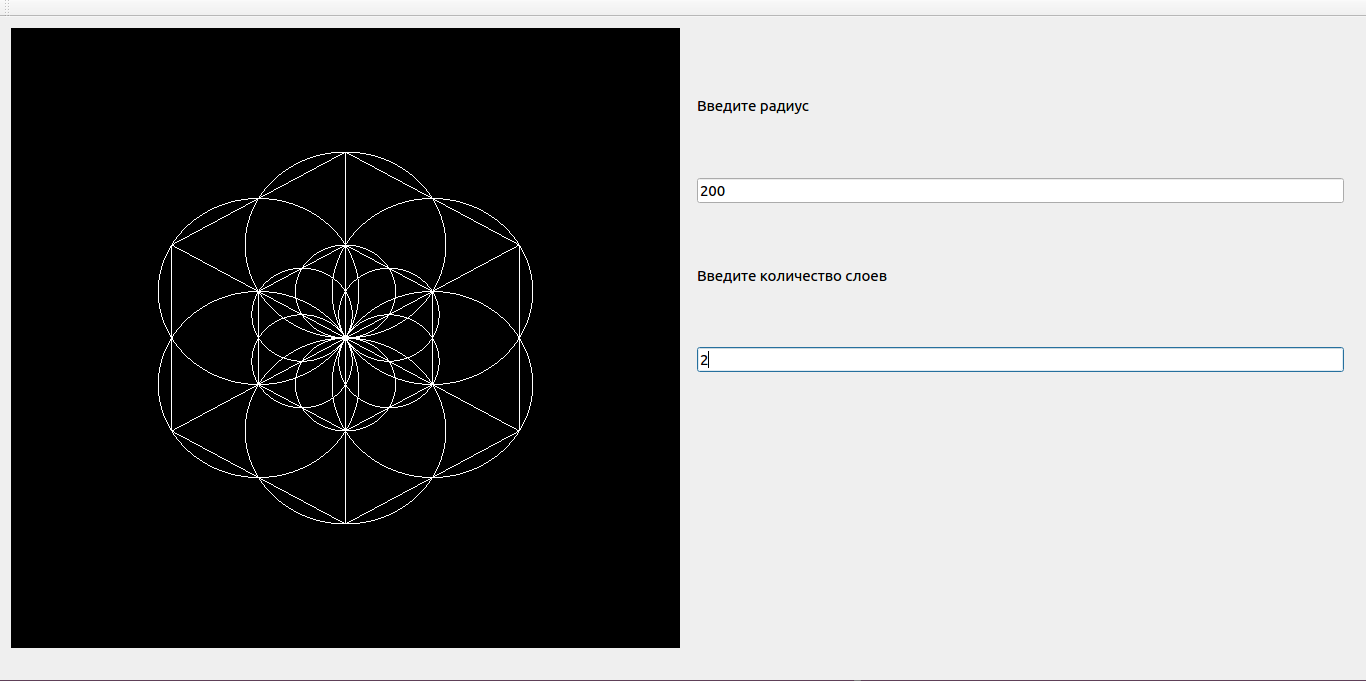
Система итерирующих функций IFC

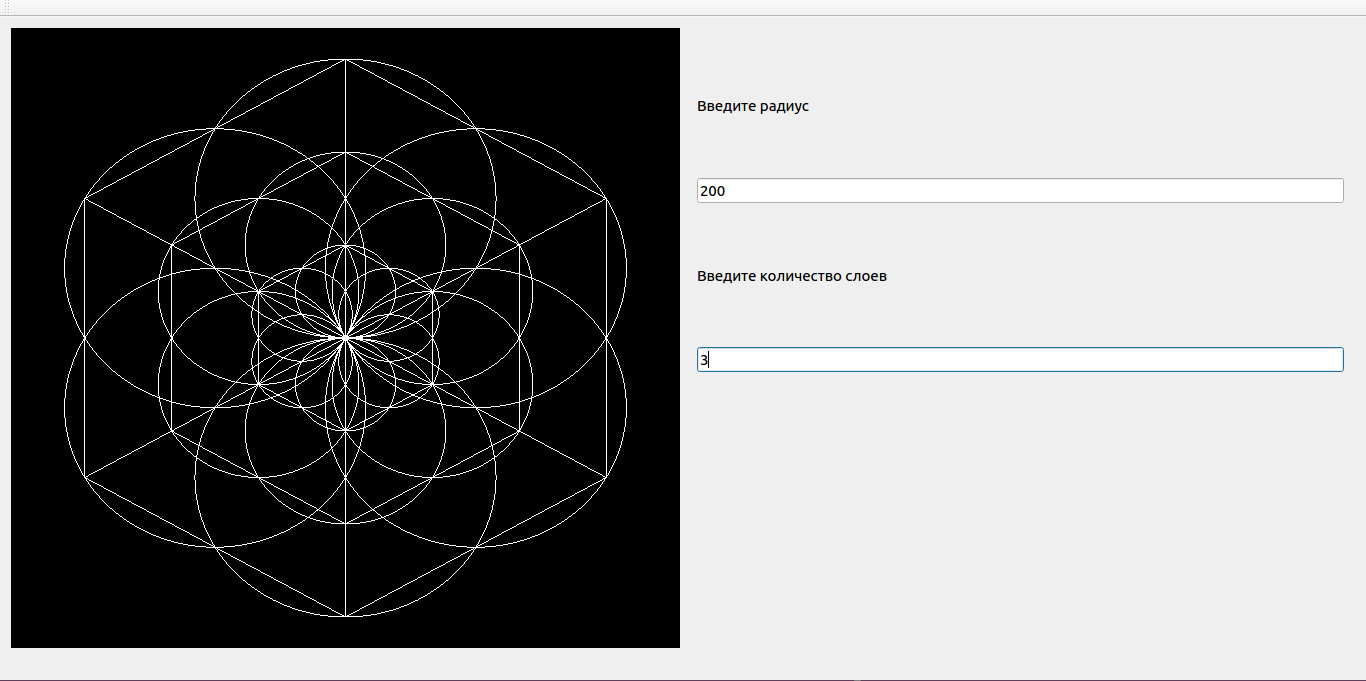
Система итерирующих функций IFC Применение таких преобразований, которые дают ту фигуру которую необходимо. Система итерирующих функций - это совокупность сжимающих аффинных преобразований. Как известно, аффинные преобразования включают в себя масштабирование, поворот и параллельный перенос. Аффинное 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);

}