AIM-To implement sierpenski using open GL

THEORY:-

The goal of this case study is to measure the performance of graph transformation tools constructing Sierpinski triangles. The Sierpinski triangle is a fractal named after Waclaw Sierpinski who described it in 1915. Originally constructed as a mathematical curve, this is one of the basic examples of self-similar sets, i.e. it is a mathematically generated pattern that can be reproduced at any magnification or reduction. An algorithm for obtaining arbitrarily close approximations to the Sierpinski triangle is as follows: 1. Start with an equilateral triangle with a base parallel to the horizontal axis. 2. Shrink the triangle by 1 2 , make two copies, and position the three shrunk triangles so that each triangle touches each of the two other triangles at a corner. 3. Repeat step 2 with each of the smaller triangles

The field of graph transformation was set up over 30 years ago, but the development of supporting tools started with considerable delay. Currently, a number of tool environments for different graph transformation approaches is available and the activity in tool development has increased considerably. Thus, a comparison of tools with respect to both functional and non-functional issues is becoming more and more important. Graph transformation tools can serve very different purposes. The case study we consider in this paper allows us to compare the efficiency of graph representations and the performance of repeated rule applications. For this comparison we have chosen the generation of Sierpinski triangles. Due to its exponential nature, the problem involves graphs which are getting huge within a few generation steps. Theses graphs need not be typed and attributed; hence very simple graph models may be used. Furthermore, the generation process is very regular and can be performed with only a few rules. In the context of the AGTIVE tool contest, the response to the call for this case study has been impressive. Twelve solutions with variants have been submitted, differing heavily in the underlying graph transformation approaches and tools, the graph representation, and the application control for rules. At the end of this paper, we categorize the given solutions and compare their runtime performance. This paper is structured as follows: The case study used for competition is presented in Section 2. It comprises the generation of Sierpinski triangles. Section 3 gives an overview on the dimensions of solutions, while Section 4 presents a variety of concrete solutions. In Section 5, we briefly compare the presented solutions and draw some conclusion

#include <GL/glut.h>

void myinit()

{

/\* attributes \*/

glClearColor(0.0, 0.0, 0.0, 0.0); /\* white background \*/

glColor3f(1.0, 0.0, 0.0); /\* draw in red \*/

/\* set up viewing \*/

/\* 500 x 500 window with origin lower left \*/

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 100.0, 0.0, 100.0);

glMatrixMode(GL\_MODELVIEW);

}

void display( void )

{

GLfloat vertices[3][2]={{0.0,0.0},{50.0,50.0},{100.0,0.0}}; /\* A triangle \*/

int j, k;

int rand(); /\* standard random number generator \*/

GLfloat p[2] ={7.5,5.0}; /\* An arbitrary initial point inside traingle \*/

glClear(GL\_COLOR\_BUFFER\_BIT); /\*clear the window \*/

/\* compute and plots 5000 new points \*/

glBegin(GL\_POINTS);

for( k=0; k<500000; k++)

{

j=rand()%3; /\* pick a vertex at random \*/

/\* Compute point halfway between selected vertex and old point \*/

p[0] = (p[0]+vertices[j][0])/2.0;

p[1] = (p[1]+vertices[j][1])/2.0;

/\* plot new point \*/

glVertex2fv(p);

}

glEnd();

glFlush(); /\* clear buffers \*/

}

void main(int argc, char\*\* argv)

{

/\* Standard GLUT initialization \*/

glutInit(&argc,argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500,500); /\* 500 x 500 pixel window \*/

glutInitWindowPosition(0,0); /\* place window top left on display \*/

glutCreateWindow("Sierpinski Gasket"); /\* window title \*/

glutDisplayFunc(display); /\* display callback invoked when window opened \*/

myinit(); /\* set attributes \*/

glutMainLoop(); /\* enter event loop \*/

}

CONCLUSION—Thus we have implemented sierpenski using open GL.



