Chapter II.

The Sierpinski Gasket.

We will try to draw the Sierpinski Gasket using the following algorithm, assuming we have the three vertices of a triangle:

1) Pick an initial point at random inside the triangle.
2) Select one of the three vertices at random.
3) Find the point halfway between the line joining the initial point and the selected vertex.
4) Display the new point by putting some sort of marker, like a small circle, at its location.
5) Replace the initial point with this new point.
6) Return to step 2.

Representation:

Of the points of the triangle (and all the other points). As a triplet \( p = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \). If we are at the 2D level, we assume that \( z = 0 \), or we can write \( p = \begin{bmatrix} x \\ y \end{bmatrix} \). Either way will be accepted by most of the Graphic System. In our case, we will conceptually work with a 3D representation, at least momentarily.

In OpenGL we use a point or a vertex. A single vertex represents a point, two vertices determine a line segment, three vertices will determine a triangle or a circle, four will imply a quadrilateral, etc. If we use the vertex function, we have several implementations, as seen below.

In OpenGL we find glVertexnfv (n stands for the dimension, 2, 3 or 4; t stands for the data type, integer (i), float (f), or double (d); v, if present, indicates that the variables are specified by a pointer to an array).

Also in OpenGL we can use basic OpenGL types, like GLint, GLfloat.

To determine the kind of object that we want to create using some vertices, OpenGL uses the glBegin function, with its correspondent glEnd. So for a line segment we use:

```
glBegin(GL_LINES)
   glVertex3i(x1, y1, z1);
   glVertex3i(x2, y2, z2);
glEnd();
```
On the other hand, for two points we use:

```c
glBegin(GL_POINTS)
    glVertex3i(x1, y1, z1);
    glVertex3i(x2, y2, z2);
glEnd();
```

With these ideas, we could write the core of the program to display the Sierpinski gasket, but we still have to worry about other issues:

1) Colors?
2) Where on the screen will the image appear?
3) How big will the image be?
4) How do we create a window on the screen?
5) How much of the real world will appear on the screen?
6) How long will the image remain on the screen?

These questions are relevant to every image we create. The answers we obtain could be applied to the rest of our programs.

**Coordinate system:**

Real world: Problem coordinate system, using floating-point numbers.

Normalized device: Coordinates (almost) independent of the device. Still floating-point.

Device: Integer coordinates, with maximum values matching the resolution of the device

Viewing transformation: The process of going from the real world coordinates to the device coordinates. This can be done by the graphics system – i.e. someone made the program for this transformation.

The OpenGL API.

Our goal is to study computer graphics, and we use the OpenGL API to obtain this goal. Primarily we use the API to be able to display lines on a “window” on the screen. Lines and points are going to be our main tools in the course.

Perhaps we could use attribute functions, to modify how a line is displayed (thickness, dot and dashes or connected, etc.) or to display text.

We will look at the minimal amount of information we need to obtain our goal.

Example:
/*
 * txtlines.c
 * This is another simple, introductory OpenGL program.
* It draws lines on the window and displays the name of the author */

#include <GL/glut.h>

void display(void)
{
  /* clear all pixels */
  glClear (GL_COLOR_BUFFER_BIT);

  /* draw lines parallel from upper left corner. */
  glColor3f (0.5, 1.0, 0.5);
  glBegin(GL_LINES);
    glVertex3f (0.02, 1.0 , 0.0);
    glVertex3f (0.5, 0.52,0.0);
  glEnd();
  glColor3f (0.0, 1.0, 0.5);
  glBegin(GL_LINES);
    glVertex3f (0.0, 1.0 , 0.0);
    glVertex3f (0.5, 0.5,0.0);
  glEnd();
  glColor3f (0.5, 1.0, 0.0);
  glBegin(GL_LINES);
    glVertex3f (0.0, 0.98 , 0.0);
    glVertex3f (0.5, 0.48,0.0);
  glEnd();

  /* Go to the lower left corner */
  glRasterPos3f(0.05,0.05, 0.0);

  /* Display the letters given by the ASCII codes 68, 82, etc. */
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 68);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 82);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 46);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 71);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 85);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 84);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 73);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 69);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 82);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 82);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 69);
  glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 90);

  /* don't wait!
   * start processing buffered OpenGL routines */
  glFlush ();
}
}
void init (void)
{
/* select clearing color */
    glClearColor (0.0, 0.0, 1.0, 0.0);

/* initialize viewing values */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
}

/*
 * Declare initial window size, position, and display mode
 * (single buffer and RGBA). Open window with "lines"
 * in its title bar. Call initialization routines.
 * Register callback function to display graphics.
 * Enter main callback loop and process events.
 */

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("lines");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;   /* ANSI C requires main to return int. */
}

When you run this program, you get the following picture:
Example of the Sierpinski Gasket:

/*
 * Sierpinski Gasket
 * This is another simple, introductory OpenGL program.
 * It draws the Sierpinski Gasket on the window.
 */
#include <GL/glut.h>
#include <math.h>

void display(void)
{

typedef GLfloat point2[2];

point2 vertices[3] = {{25.0, 25.0}, {50.0, 75.0}, {75.0, 25.0}};

int i,j, k;

point2 p = {55, 55};

glClear (GL_COLOR_BUFFER_BIT);

/* Compute and display 1000 new points */
for(k = 0; k <= 4000;k++)
{
    j = rand()%3; /* pick vertex at random */

    /* Compute halfway point between vertex and p */

    p[0] = (p[0] + vertices[j][0])/2.0;
    p[1] = (p[1] + vertices[j][1])/2.0;

    /* plot the point */

    glBegin(GL_POINTS);
    glVertex2fv(p);
    glEnd();
}

/* don't wait!
 * start processing buffered OpenGL routines
 */
    glFlush ();
}

void init (void)
{
    /* select clearing color */
    glClearColor (0.0, 0.0, 1.0, 0.0);

    /* initialize viewing values */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 100.0, 0.0, 100.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
}

/* Declare initial window size, position, and display mode
 * (single buffer and RGBA). Open window with "Sierpinski Gasket"
 * in its title bar. Call initialization routines.
 * Register callback function to display graphics.
 * Enter main loop and process events.
 */
int main(int argc, char** argv)
{  
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("Sierpinski Gasket");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;  /* ANSI C requires main to return int. */
}

When you run this program, you get the following picture:

![Sierpinski Gasket](image)

So we have used the OpenGL Primitives, like GL_POINTS, GL_LINES, and this is the minimum that we will need to work our displays.

We also have used the colors - glColor**(…), and we have learned how to initialized the glut library - glutInit(…), setup the display mode - glutInitDisplayMode(…), determine the window size - glutInitWindowSize(…) and position - glutInitWindowPosition , etc.

We learned how to move the position of the Raster to an specific place – glRasterPos***(..) , and how to display text..
Finally we learned how to define a window on the real world through a set of instructions, the init() functions on our programs.

Finally we should comment that we had taken care of the following fact: The aspect ratio of the 2 dimensional window – glutInitWindowSize(...) and the 2 dimensional window created in the 3 dimensional world through the glOrtho(..) function should be the same, in order to avoid distorting the picture. (Remark: If the pixel density is different in the horizontal and vertical rows, we need an adjustment of size, based on this difference of densities). One solution is to use a viewport, a rectangular area of the display window, which can be specified through an OpenGL function. We can make the specification directly from our program without using the OpenGL function.

An example of a 3D program will be the 3D gasket:

```c
/*
 * 3D Sierpienski Gasket
 * This is another simple, introductory OpenGL program.
 * It draws the 3D Sierpinski Gasket on the window.
 */
#include <GL/glut.h>
#include <math.h>

void display(void)
{
    typedef GLfloat point3[3];
    point3 vertices[4] = {{25.0,25.0,25.0},{25.0, 75.0, 25.0},{75.0,25, 25},{75,75,75}};

    int i,j, k;
    point3 p = {35, 35, 35};
    glClear (GL_COLOR_BUFFER_BIT);
    /* Compute and display 1000 new points */
    for(k = 0; k <= 10000;k++)
    {
        j = rand()%4; /* pick vertex at random */
    
        /* Compute halfway point between vertex and p */
```
p[0] = (p[0] + vertices[j][0])/2.0;
p[1] = (p[1] + vertices[j][1])/2.0;

/* plot the point */

glBegin(GL_POINTS);
    glColor3f(p[0]/100,p[1]/100,p[2]/100);
    glVertex3fv(p);
glEnd();

} /* don't wait!
* start processing buffered OpenGL routines */

glFlush ();

void init (void)
{
    /* select clearing color */
    glClearColor (0.0, 0.0, 1.0, 0.0);

    /* initialize viewing values */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 100.0, 0.0, 100.0, -100.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

/* Declare initial window size, position, and display mode
* (single buffer and RGBA). Open window with "Sierpinski Gasket"
* in its title bar. Call initialization routines.
* Register callback function to display graphics.
* Enter main loop and process events. */

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (600, 600);
    glutInitWindowPosition (100, 100);
glutCreateWindow ("3D Gasket");
init ();
glutDisplayFunc(display);
glutMainLoop();
return 0;  /* ANSI C requires main to return int. */
}

Now the display is more difficult to understand. Although we have used colors to
distinguish the different depth of the 3D points, we have not taken care of the hidden
points removal. The points are drawn (redrawn) as the execution of the program dictates,
without taking into consideration the geometric relationships among the points.