

OpenGL Compendium

Who should read this document?

1. Readers of book on Computer Graphics, by Sinha A. N. and Udai A. D., TMH
2. Beginners of OpenGL (OGL) programming language.

Requisites

1. Should have preliminary knowledge of C/C++.
2. Should have the following files at the location listed below

File	Location
Gl.h Glut.h Glu.h	[compiler]\include\gl
Opengl32.lib Glut32.lib Glu32.lib	[compiler]\lib
Opengl32.dll Glut32.dll Glu32.dll	[system]

For users of Visual C++, [compiler] is c:\Program Files\Microsoft Visual Studio\VC98 directory. For Windows 9x/NT/2000/XP users, [system] is C:\winnt\system32 or c:\windows\system directory.

What are OpenGL, GLU and GLUT?

OpenGL is software designed specially for graphics generation on a variety of hardware systems. The interface consists of about 250 distinct high level commands none of which performs windowing tasks or obtains a user input which ensures its implementation on many hardware platforms and a device independent approach. Various primitives like point, line, polyline or a polygon can be generated by small set of graphics routines with OpenGL. OpenGL Utility library (GLU) and OpenGL Utility Toolkit (GLUT) are sophisticated library of such routines which supports complicated graphics output like quadric surface or a NURBS surface. With OpenGL tools generation of wire frame model, depth-queued scene, antialiasing, shading, adding shadows and texture, fog generation, lighting or even rendering becomes very simple. Various other libraries like X Windows Extension to Open GL (GLX), Apple OpenGL (AGL) or Windows OpenGL interface (WGL) are also developed over OpenGL.

In this document the discussion is confined mostly to GLUT, which is a windows system independent toolkit, written by Mark Kilgard, as it serves the basic scope of amateur graphics programmers.

Where to obtain the OpenGL tools for program? Important Links!

Microsoft Visual C++ comes packaged with OpenGL Libraries and Header files for GL and GLU. Still it can be downloaded from Microsoft's Help and Support site for OpenGL Release Notes and Components at

<http://download.microsoft.com/download/win95upg/info/1/w95/en-us/opengl95.exe>

Nate Robins page for GLUT is an easiest way to download the latest version of GLUT 3.0 and its documentation files for Windows at

<http://www.xmission.com/~nate/glut.html>

For any further FAQ, documentation and Sample code the home page for OpenGL is an excellent resource at <http://www.opengl.org>

Why to Switch for OGL/GLUT?

OpenGL has proved to be an excellent tool in current day graphics software developers as well as amateur programmers. It provides an easy but powerful library set for graphics generation. It directs the graphics developers directly into the physics and mathematics of graphics rather than achieving programming excellence.

Command Syntax and Simple Commands.

All OpenGL commands are prefixed with appropriate sets of alphabets as detailed below

Prefix	Origin	Header files	Example
gl	OpenGL	GL/gl.h	glClearColor(); glEnable(); glBegin();
glut	GLUT	GL/glut.h	glutCreateWindow(); glutMainLoop();

Prefixes like glu, glX, agl, or wgl correspond to respective libraries as discussed before. Microsoft Windows version of gl.h or glut.h requires windows.h to be included before gl.h or glut.h because some macros are defined in windows.h which is used in the OpenGL headers.

Similarly, GLX requires the include files X11/Xlib.h to be included before glx.h. Additional headers should be included if the code is using GLX, AGL, WGL etc. It is common to use stdlib.h and stdio.h with OpenGL code as most of the applications also use non graphics standard C library headers. One can always try to compile without these headers if he/she finds it suitable for any particular code.

Data types and suffixes

Suffix	Data Type	C – Language Type	OpenGL Type definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating point	float	GLfloat, GLclampf
d	64-bit floating point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield
v	void	None	GLvoid, used for pointers to arrays of values.

Example:

glVertex2i(2, 3); will signify a OpenGL command with 2 int type parameters.

glColor3f(0.2, 0.5, 1.0); will signify a command with 3 float type parameters.

glColor3fv(color_array);

Skeleton of a basic OpenGL code

```
#include <windows.h>
#include <gl.h>
#include <glut.h>
#include <Program specific files>

void main ()
{
    Initialization routines
    Create a graphics output window
    glutDisplayFunc(a_Display_Function);
    glutReshapeFunc(a_Reshape_Function);
    glutMouseFunc(a_Mouse_Handler_Function);
    glutKeyboardFunc(a_Keyboard_Handler_Function);
    other initialization functions;
    call for other functions;
    glutMainLoop();
}
```

Callback function definitions

Sample codes for Output Primitives, 3D Transformations, Shading, Fractal, and Animation to start with.

1. The following sample code demonstrates the basic steps of an OpenGL program and introduces a simple object definition.

```

//A Sample code demonstrating OpenGL steps and defining a polygon
primitive

#include <GL/glut.h>

void display (void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.0);
        glVertex3f(0.75, 0.25, 0.0);
        glVertex3f(0.75, 0.75, 0.0);
        glVertex3f(0.25, 0.75, 0.0);
    glEnd();
    glFlush();
}
void init (void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
void main (void)
{
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(150, 100);
    glutCreateWindow("OGL1 - My First Code on OpenGL");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}

```

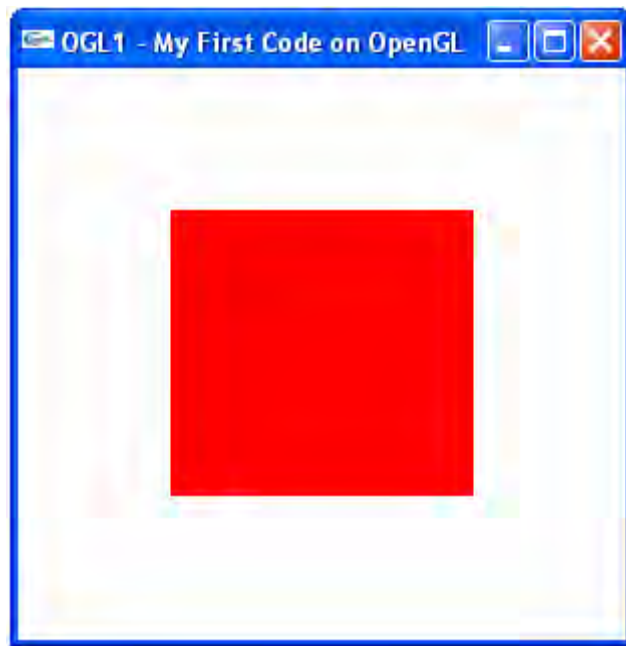


Fig. 1: Output of Sample Code 1.

2. The following sample code demonstrates the use of OpenGL in generating a simple fractal using a point primitive (Refer Fig. 14.11)

```
#include <stdlib.h>
#include <GL/glut.h>

struct GLintPoint
{
    GLint x, y;
};
GLint random(GLint m)
{
    return rand()%m;
}
void drawPoint(GLint x, GLint y)
{
    glBegin(GL_POINTS);
        glVertex2i(x, y);
    glEnd();
}
void Sierpinski(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    GLintPoint P[3] = {{10,10},{600,10},{300,450}};
    int index = random(3);
    GLintPoint point = P[index];
    drawPoint(point.x, point.y);
    for (GLint i = 0; i < 100000; i++)
    {
        glColor3f(1.0, 0.0, 0.0);
        index = random(3);
        point.x = (point.x + P[index].x)/2;
        point.y = (point.y + P[index].y)/2;
        drawPoint(point.x, point.y);
    }
    glFlush();
}
void init (void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(1.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void main (void)
{
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("OGL2 - Sierpinski");
    init();
    glutDisplayFunc(Sierpinski);
    glutMainLoop();
}
```



Fig 2: Output of Sample Code 2.

3. A sample code below implements Bresenham's algorithm for generation of a line primitive (Refer Chapter 3, Section 3.5 and Sample code 3.2)

```
#include <stdlib.h>
#include <GL/glut.h>

struct GLintPoint
{
    GLint x, y;
};
void drawPoint(GLint x, GLint y)
{
    glBegin(GL_POINTS);
        glVertex2i(x, y);
    glEnd();
}
void bresline(GLintPoint a, GLintPoint b)
{
    GLint x=a.x,y=a.y,xend,dx=abs(b.x-a.x),dy=abs(b.y-a.y);
    GLint p = 2 * dy - dx;
    if (a.x > b.x)
    {
        x = b.x;
        y = b.y;
        xend = a.x;
    }
    else
    {
        x = a.x;
        y = a.y;
        xend = b.x;
    }
    for (x = a.x; x <= xend; x++)
    {
        drawPoint(x, y);
        if (p < 0)
            p += 2 * dy;
        else
            p += 2 * dy - dx;
            y++;
    }
}
```

```

        {
            y++;
            p += 2*(dy - dx);
        }
    }
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    GLintPoint P[2] = {{10,10}, {600, 410}};
    bresline(P[0], P[1]);
    glFlush();
}
void init (void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void main (void)
{
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("OGL3 - Bresenham's Line");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}

```

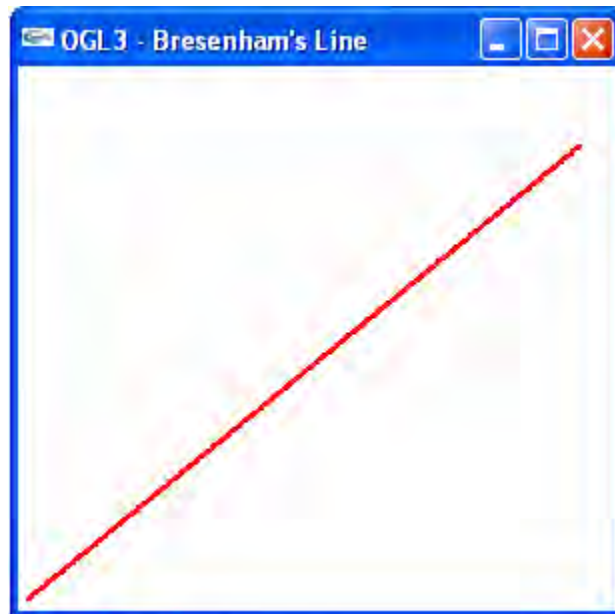


Fig 3: Output of Sample code 3

4. The following code demonstrates an application of OpenGL in 3D Transformations. (Refer Chapter 7, Section 7.3)

```

#include <windows.h>
#include <GL/glut.h>

#define draw_line(x1,y1,z1,x2,y2,z2) glBegin(GL_LINES); \
    glVertex3f(x1,y1,z1); glVertex3f(x2,y2,z2); glEnd();

void draw_axis(GLdouble length)
{
    glPushMatrix();
    draw_line(0, 0, 0, 0, 0, length);
    glTranslated(0, 0, length);
    glutWireCone(0.05, 0.2, 10, 10);
    glPopMatrix();
}

void display(void)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0*4/3, 2.0*4/3, -2.0, 2.0, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(-2.0,2.0,2.0,0.0,0.0,0.0,0.0,1.0,0.0);

    glClear(GL_COLOR_BUFFER_BIT);

    glColor3d(0, 0, 0);
    draw_axis(1.5);           //Draw Z-Axis

    glPushMatrix();
    glRotated(90, 0, 1, 0);
    draw_axis(1.5);           //Draw x-Axis
    glRotated(-90, 1, 0, 0);
    draw_axis(1.5);           //Draw y-Axis
    glPopMatrix();

    glPushMatrix();
    glTranslated(1.0, 0.25, 1.0);
    glutWireCube(0.5);
    glPopMatrix();

    glColor3d(0, 0, 1);
    glPushMatrix();
    glTranslated(1.0, 0.25, 1.0);
    glRotated(30, 0, 0, 1);
    glutWireCube(0.5);
    glPopMatrix();

    glFlush();
}

void main (void)
{
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("OGL4 - 3D Transformation");
    glutDisplayFunc(display);
}

```



```

glClearColor(1.0, 1.0, 1.0, 0.0);
glViewport(0, 0, 640, 480);
glutMainLoop();
}

```

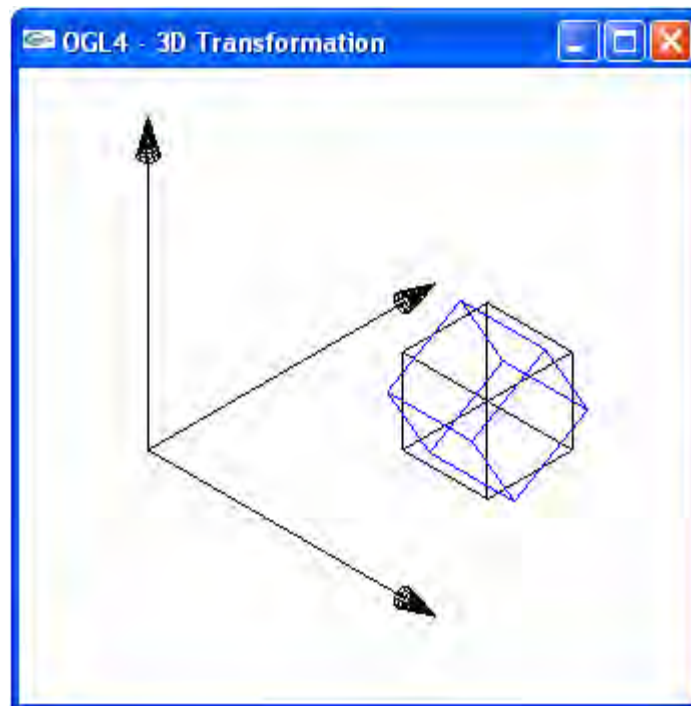


Fig 4: Output of Sample code 4.

5. The following code demonstrates the use of OpenGL for Smooth Shading of a Sphere using various illumination techniques as discussed in Chapter 11 and demonstrated in Sample code 11.1.

```

#include <windows.h>
#include <GL/glut.h>

void display_sphere(void)
{
    GLfloat mat_ambient[] = {0.7f, 0.7f, 0.7f, 1.0f};
    GLfloat mat_diffuse[] = {0.6f, 0.6f, 0.6f, 1.0f};
    GLfloat mat_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
    GLfloat mat_shininess[] = {50.0f};

    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    GLfloat light_Intensity[] = {1.0f, 0.4f, 0.0f, 1.0f};
    GLfloat light_Position[] = {2.0f, 6.0f, 3.0f, 0.0f};
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Intensity);
    glLightfv(GL_LIGHT0, GL_POSITION, light_Position);
}

```

```

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
GLdouble winHt = 1.0;
glOrtho(-winHt*4/3,winHt*4/3,-winHt,winHt,0.1, 100);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(2.3, 1.3, 2, 0, 0.25, 0, 0.0, 1.0, 0.0);

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glTranslated(0.4, 0.4, 0.4);
glutSolidSphere(0.5, 50, 50);
glPopMatrix();
glFlush();
}
void main (void)
{
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("OGL5 - Shading and Illumination Example");
    glutDisplayFunc(display_sphere);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_SMOOTH);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_NORMALIZE);
    glClearColor(0.1, 0.1, 0.1, 0.0);
    glViewport(0, 0, 640, 480);
    glutMainLoop();
}

```

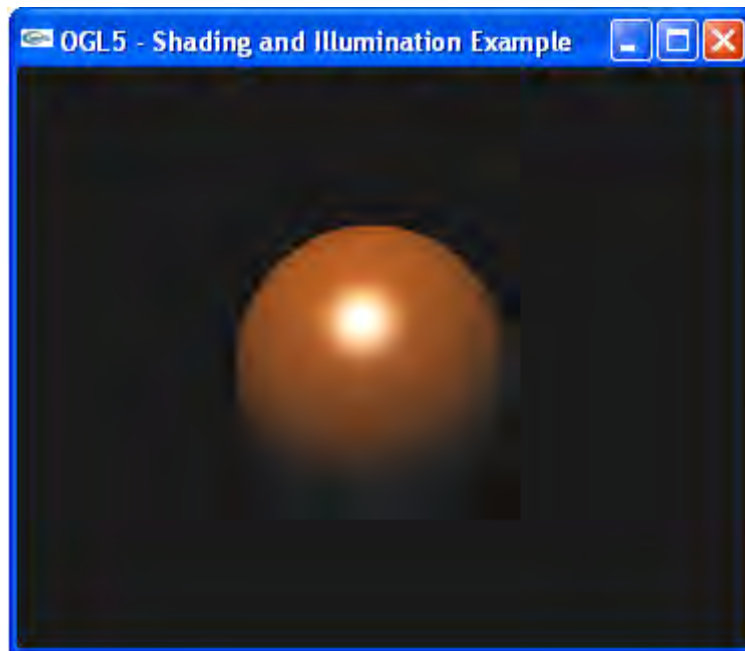


Fig 5: Output of Sample code 5

6. The following code demonstrates the use of OpenGL in Animation with Page Flipping technique (i.e. double buffering) discussed in Section 15.6 of the Ref. [1].

```
#include <stdlib.h>
#include <GL/glut.h>

static GLfloat height = 0.0, width = 150.0;
static GLfloat incx, incy;

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glTranslatef(width, height, 0);
    glColor3f(1.0, 0, 0);
    glRectf(-5.0, -5.0, 5.0, 5.0);
    glPopMatrix();
    glutSwapBuffers();
}

void bounceDisplay(void)
{
    height += incy;
    width += incx;
    if (height > 95.0)
        incy = -0.17;
    if (width > 95.0)
        incx = -0.1;
    if (height <= 5)
        incy = 0.13;
    if (width <= 5)
        incx = 0.1;

    glutPostRedisplay();
}

void reshape(GLint w, GLint h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 100.0, 0.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void mouse(GLint button, GLint state, GLint x, GLint y)
{
    switch(button){
        case GLUT_LEFT_BUTTON:
            if(state == GLUT_DOWN)
                glutIdleFunc(bounceDisplay);
            break;
        case GLUT_MIDDLE_BUTTON:
            if(state == GLUT_DOWN)
                glutIdleFunc(NULL);
            break;
        default:
            break;
    }
}
```

```

    }
}
void main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Left Mouse Button to Start & Middle Button to
Stop Animation");
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutMainLoop();
}

```

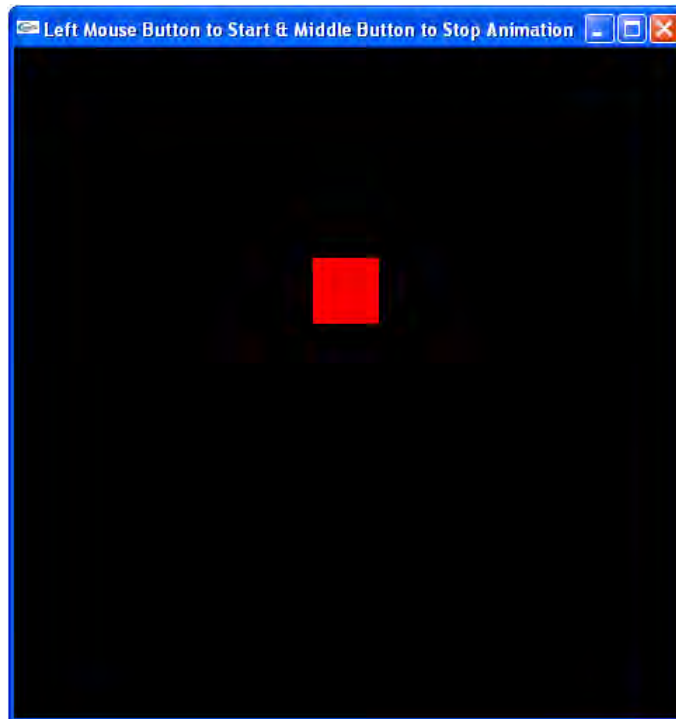


Fig 6: Output of Sample Code 6

7. The following code demonstrates the use of OpenGL in Fractal Generation through an affine fern fractal.

```

//An affine fractal fern generator program
#include <stdlib.h>
#include <GL/glut.h>

GLint xscale, yscale, xoffset, yoffset, p[3];
GLfloat a[4], b[4], c[4], d[4], e[4], f[4];

void drawPoint(GLint x, GLint y)
{
    glBegin(GL_POINTS);
        glVertex2i(x, y);
    glEnd();
}

```

```

}
void fern(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    GLint k, px, py;
    GLfloat x, y, newx;
    for (GLint i=1;i<=25000;i++)
    {

        GLint j = rand();
        k = (j<p[0]) ? 0:((j<p[1]) ? 1:((j<p[2]) ? 2:3 ));
        newx = (a[k] * x + b[k] * y + e[k]);

        y = (c[k] * x + d[k] * y + f[k]);
        x = newx;

        px = x * xscale + xoffset;
        py = y * yscale + yoffset;

        if ((px >= 0)&&(px < 640)&&(py >= 0)&&(py < 480))
            drawPoint (px, py) ;
    }
    glFlush();
}
void init (void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.2, 1.0, 0.5);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void main(void)
{
    a[0] = 0; a[1] = 0.20; a[2] = -0.15; a[3] = 0.85;
    b[0] = 0; b[1] = -0.26; b[2] = 0.28; b[3] = 0.04;
    c[0] = 0; c[1] = 0.23; c[2] = 0.26; c[3] = -0.04;
    d[0] = 0.16; d[1] = 0.22; d[2] = 0.24; d[3] = 0.85;
    e[0] = 0; e[1] = 0; e[2] = 0; e[3] = 0;
    f[0] = 0; f[1] = 1.6; f[2] = 0.44; f[3] = 1.6;

    p[0] = 480; p[1] = 2621; p[2] = 4915;

    xscale = 40;
    yscale = 40;
    xoffset = 300;
    yoffset = 50;

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("OGL8 - A Fractal Fern");
    init();
    glutDisplayFunc(fern);
    glutMainLoop();
}

```

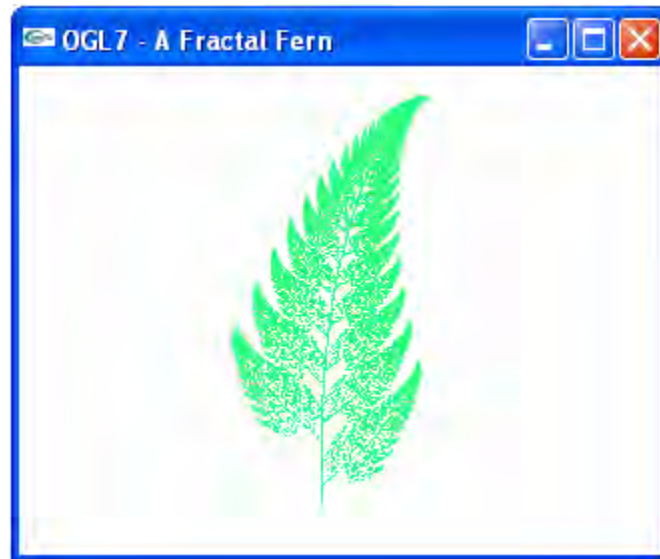


Fig. 7: Output of Sample Code 7.

References

1. Computer Graphics, Sinha A. N. and Udai A. D., Tata McGraw Hill, New Delhi, 2007.
2. OpenGL Programming Guide, Mason Woo, Jackie Neider, Tom Davis and Dave Shreiner, Pearson Education Asia, Delhi, 2000.