```
/* sierpinski gasket with vertex arrays */
/* This version uses filled triangles. */

#include "Angel.h"

using namespace std;

const int NumTimesToSubdivide = 5;
const int NumTriangles = 243;  // 3^5 triangles generated
const int NumVertices  = 3 * NumTriangles;

vec2 points[NumVertices];
int Index = 0;

//----------------------------------------------------------------------------

void
triangle( const vec2& a, const vec2& b, const vec2& c )
{
    points[Index++] = a;
    points[Index++] = b;
    points[Index++] = c;
}

//----------------------------------------------------------------------------

void
divide_triangle( const vec2& a, const vec2& b, const vec2& c, int count )
{
    if ( count > 0 ) {
        vec2 v0 = ( a + b ) / 2.0;
        vec2 v1 = ( a + c ) / 2.0;
        vec2 v2 = ( b + c ) / 2.0;
        divide_triangle( a, v0, v1, count - 1 );
        divide_triangle( c, v1, v2, count - 1 );
        divide_triangle( b, v2, v0, count - 1 );
    }
    else {
        triangle( a, b, c );    // draw triangle at end of recursion
    }
}

//----------------------------------------------------------------------------

void
init( void )
{
    vec2 vertices[3] = {
        vec2( -0.9, -0.9 ), vec2( 0.0, 0.9 ), vec2( 0.9, -0.9 )
    };

    // Subdivide the original triangle
    divide_triangle( vertices[0], vertices[1], vertices[2],
                     NumTimesToSubdivide );

    // Create a vertex array object
    GLuint vao;
    glGenVertexArrays( 1, &vao );
    glBindVertexArray( vao );

    // Create and initialize a buffer object
    GLuint buffer;
    glGenBuffers( 1, &buffer );
```

```
    glBindBuffer( GL_ARRAY_BUFFER, buffer );
    glBufferData( GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW );

    // Load shaders and use the resulting shader program
    GLuint program = InitShader( "vshader22.glsl", "fshader22.glsl" );
    glUseProgram( program );

    // Initialize the vertex position attribute from the vertex shader
    GLuint loc = glGetAttribLocation( program, "vPosition" );
    glEnableVertexAttribArray( loc );
    glVertexAttribPointer( loc, 2, GL_FLOAT, GL_FALSE, 0,
                           BUFFER_OFFSET(0) );

    glClearColor( 1.0, 1.0, 1.0, 1.0 ); /* white background */
}

//----------------------------------------------------------------------------

void
display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glDrawArrays( GL_TRIANGLES, 0, NumVertices );
    glFlush();
}

//----------------------------------------------------------------------------

void
keyboard( unsigned char key, int x, int y )
{
    switch ( key ) {
    case 033:
        exit( EXIT_SUCCESS );
        break;
    }
}

//----------------------------------------------------------------------------

int
main( int argc, char **argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGBA );
    glutInitWindowSize( 512, 512 );
    glutInitContextVersion( 3, 2 );
    glutInitContextProfile( GLUT_CORE_PROFILE );
    glutCreateWindow( "Simple GLSL example" );

    glewExperimental = GL_TRUE;
    glewInit();

    init();

    glutDisplayFunc( display );
    glutKeyboardFunc( keyboard );

    glutMainLoop();
    return 0;
}
```