

Computer Graphics (CS 543)

Lecture 2a: Introduction to OpenGL/GLUT (Part 2)

Prof Emmanuel Agu

*Computer Science Dept.
Worcester Polytechnic Institute (WPI)*

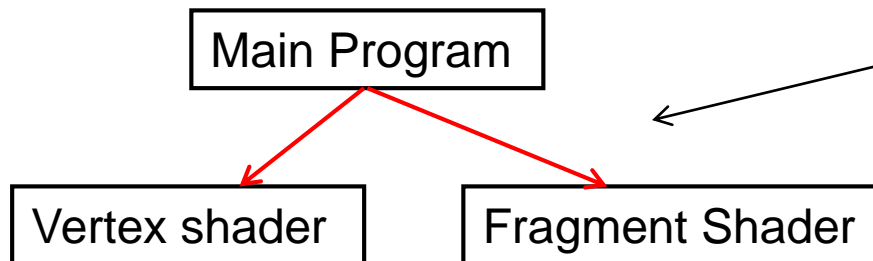




OpenGL Program: Shader Setup

- Modern OpenGL programs have 3 parts:
 - Main **OpenGL program** (.cpp file), **vertex shader** (e.g. vshader1.glsl), and **fragment shader** (e.g. fshader1.glsl) in same Windows directory
 - In main program, need to link names of vertex, fragment shader
 - **initShader()** is homegrown shader initialization function. More later

```
GLuint program = InitShader( "vshader1.glsl", fshader1.glsl" );  
glUseProgram(program) ;
```



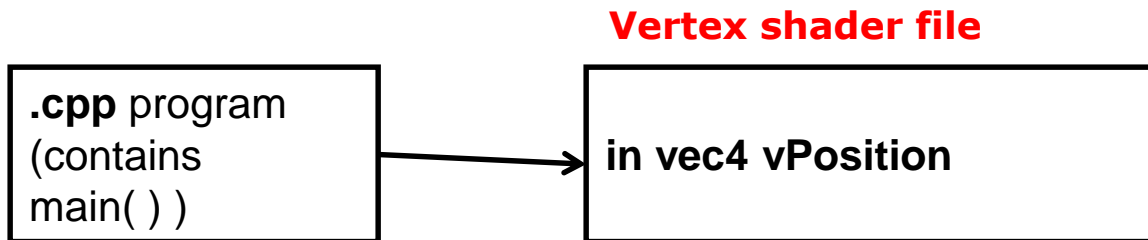
initShader()

Homegrown, connects main Program to shader files
More on this later!!



Vertex Attributes

- Want to make 3 dots (vertices) accessible as variable **vPosition** in vertex shader
- First declare vPosition in vertex shader, get its address



- Compiler puts all variables declared in shader into a table
- Need to find location of vPosition in table of variables

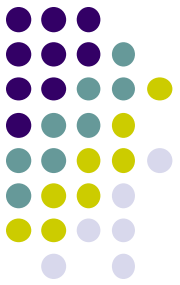
Location of
vPosition



Variable
Variable 1
vPosition
.....
Variable N

```
GLuint loc = glGetAttribLocation( program, "vPosition" );
```

Vertex Attributes



Location of
vPosition



Variable
Variable 1
vPosition
.....
Variable N

Get location of vertex attribute **vPosition**



```
GLuint loc = glGetAttribLocation( program, "vPosition" );  
glEnableVertexAttribArray( loc );
```



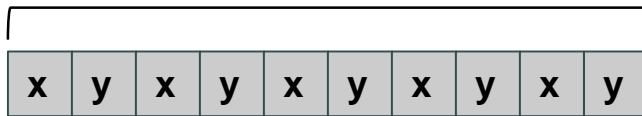
Enable vertex array attribute
at location of **vPosition**

glVertexAttribPointer



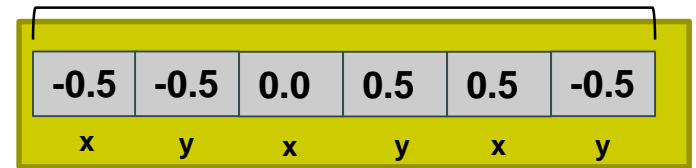
- Data now in VBO on GPU, but need to specify meta format (using `glVertexAttribPointer`)
- Vertices are packed as array of values

Vertices stored in array



vertex 1 vertex 2

E.g. 3 dots stored in array on VBO



dot 1 dot 2 dot 3

```
glVertexAttribPointer( loc, 2, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0) );
```

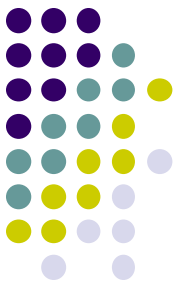
Location of `vPosition` in table of variables

2 (x,y) floats per vertex

Data not normalized to 0-1 range

Padding between Consecutive vertices

Data starts at offset from start of array



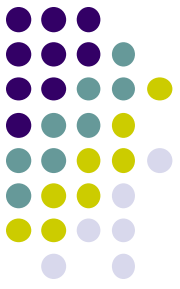
Put it Together: Shader Set up

```
void shaderSetup( void )
{
    // Load shaders and use the resulting shader program
    program = InitShader( "vshader1.glsl", "fshader1.glsl" );
    glUseProgram( program );

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

    // sets white as color used to clear screen
    glClearColor( 1.0, 1.0, 1.0, 1.0 );
}
```

OpenGL Skeleton: Where are we?



```
void main(int argc, char** argv){
    glutInit(&argc, argv);    // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );
```

// ... now register callback functions

```
glutDisplayFunc(myDisplay);
glutReshapeFunc(myReshape);
glutMouseFunc(myMouse);
glutKeyboardFunc(myKeyboard);
```

```
glewInit( );
```

```
generateGeometry( );
```

```
initGPUBuffers( );
```

```
void shaderSetup( );
```

```
glutMainLoop( );
```

```
}
```

```
void shaderSetup( void )
```

```
{
```

```
    // Load shaders and use the resulting shader program
```

```
    program = InitShader( "vshader1.glsl", "fshader1.glsl" );
```

```
    glUseProgram( program );
```

```
    // Initialize vertex position attribute from vertex shader
```

```
    GLuint loc = glGetAttribLocation( program, "vPosition" );
```

```
    glEnableVertexAttribArray( loc );
```

```
    glVertexAttribPointer( loc, 2, GL_FLOAT, GL_FALSE, 0,
```

```
                           BUFFER_OFFSET(0) );
```

```
    // sets white as color used to clear screen
```

```
    glClearColor( 1.0, 1.0, 1.0, 1.0 );
```

```
}
```



Vertex Shader

- We write a simple “pass-through” shader
- Simply sets **output vertex position = input position**
- **gl_Position** is built-in variable (already declared)

```
in vec4 vPosition
```

```
void main( )
```

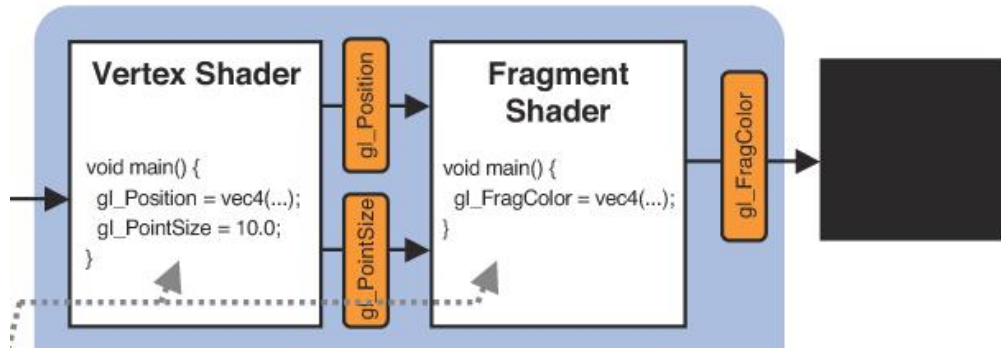
```
{
```

```
    gl_Position = vPosition;
```

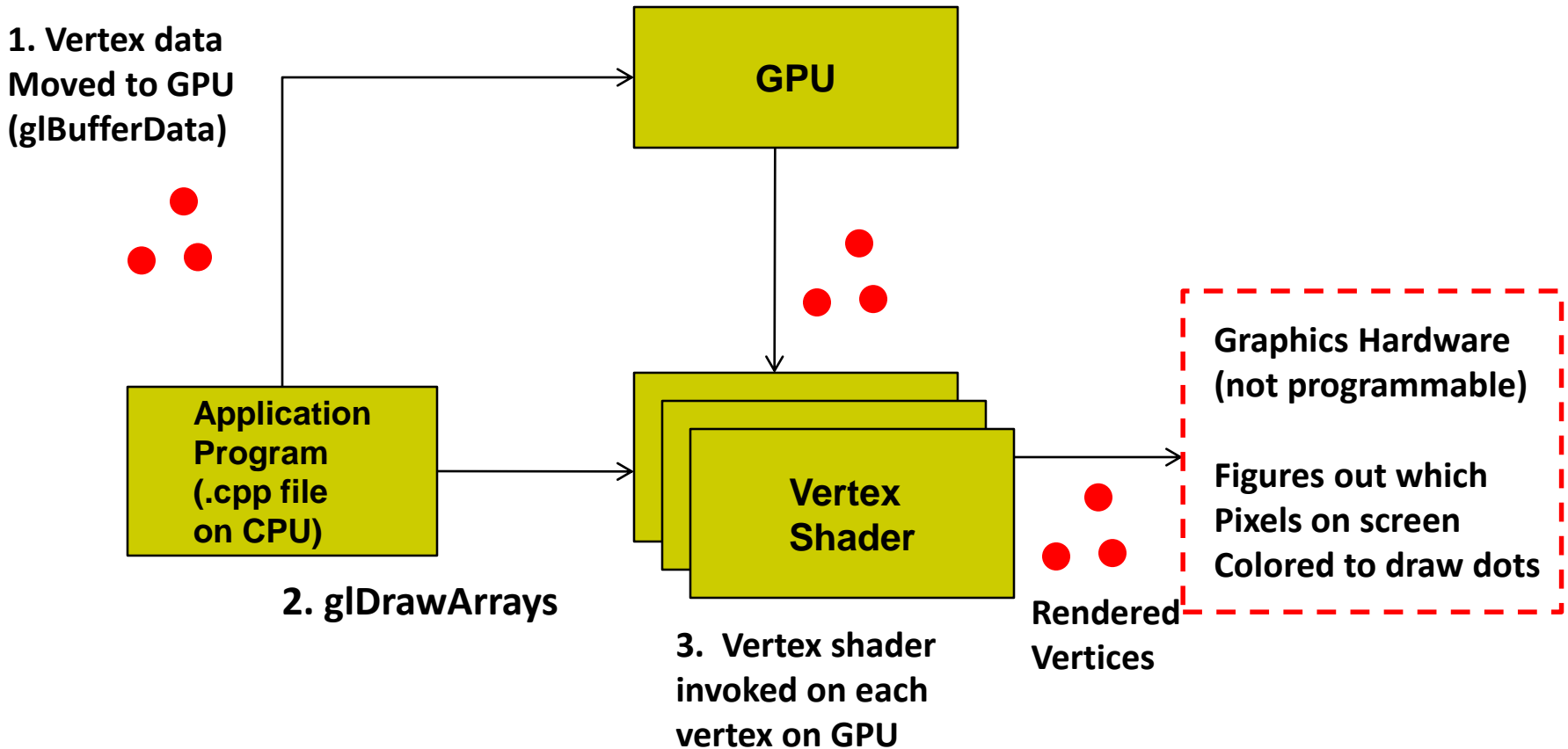
```
}
```

output vertex position

input vertex position
(from .cpp file)



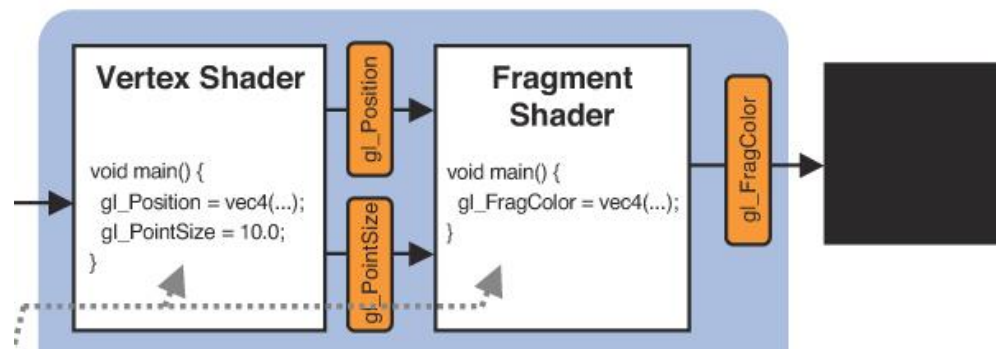
Execution Model





Fragment Shader

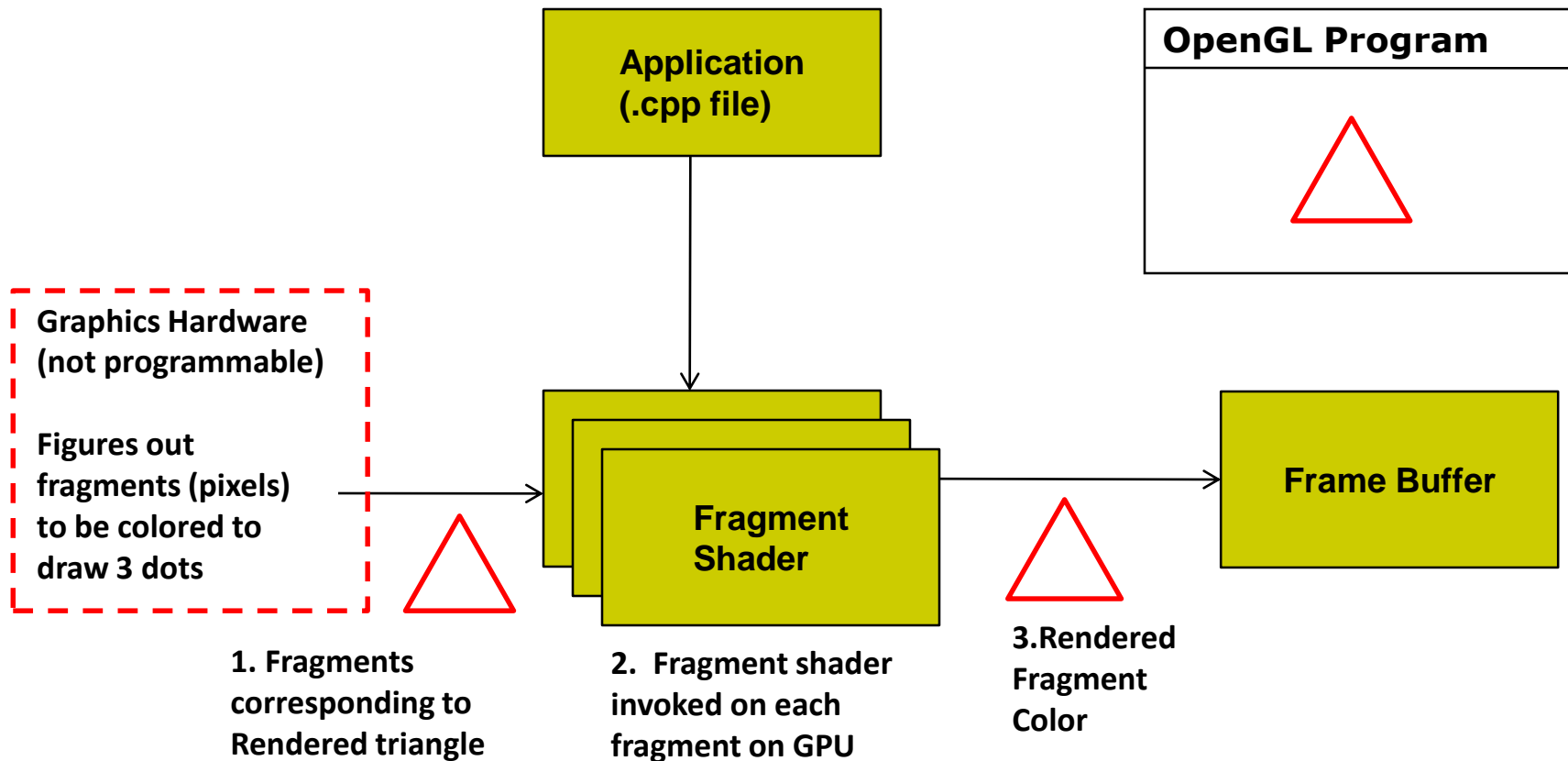
- We write a simple fragment shader (sets color of dots to red)
- `gl_FragColor` is built in variable (already declared)



```
void main( )  
{  
    R    G    B  
    gl_FragColor = vec(1.0, 0.0, 0.0, 1.0);  
}
```

Set each drawn fragment color to red

Execution Model





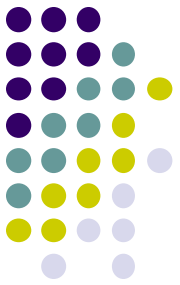
Recall: OpenGL Skeleton

```
void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);    // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );

    // ... now register callback functions
    glutDisplayFunc(myDisplay); ←--Next... how to draw in myDisplay
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit( );
    glutMainLoop( );
}
```

Recall: Draw points (from VBO)



```
glDrawArrays (GL_LINE_LOOP, 0, N);
```

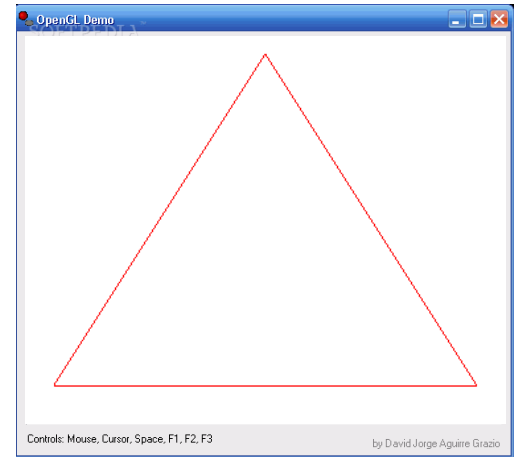
Render buffered data as line loop

Starting index

Number of points to be rendered

- Display function using `glDrawArrays`:

```
void mydisplay(void) {  
    glClear(GL_COLOR_BUFFER_BIT);           // clear screen  
    glDrawArrays(GL_LINE_LOOP, 0, 3);      // draw the points  
    glFlush( );                            // force rendering to show  
}
```

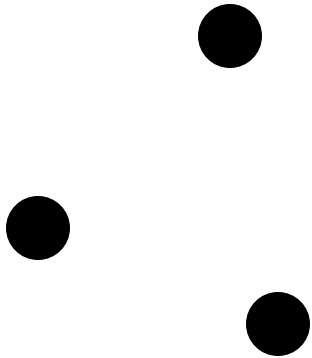


Other possible arguments to `glDrawArrays` instead of `GL_LINE_LOOP`?



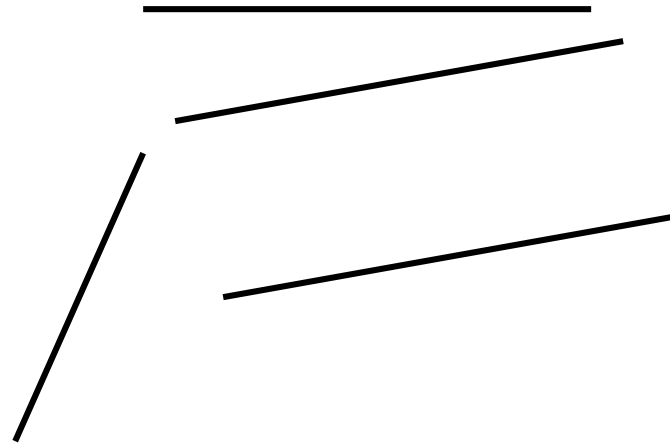
`glDrawArrays(GL_POINTS, ...)`

– draws dots



`glDrawArrays(GL_LINES, ...)`

– Connect vertex pairs to draw lines

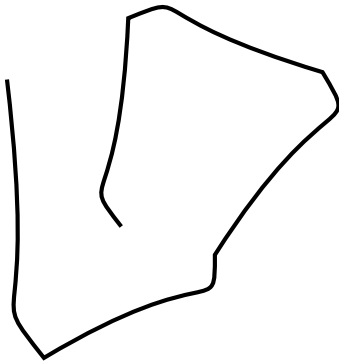




glDrawArrays() Parameters

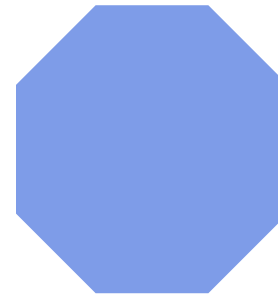
glDrawArrays(GL_LINE_STRIP,..)

– polylines



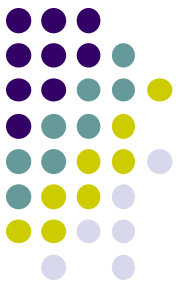
glDrawArrays(GL_POLYGON,..)

– convex filled polygon



glDrawArrays(GL_LINE_LOOP)

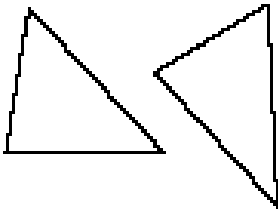
– Close loop of polylines
(Like GL_LINE_STRIP but closed)



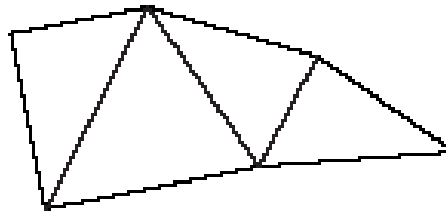
glDrawArrays() Parameters

- Triangles: Connect 3 vertices
 - GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN
- Quad: Connect 4 vertices
 - GL_QUADS, GL_QUAD_STRIP

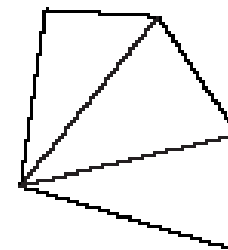
GL_TRIANGLES



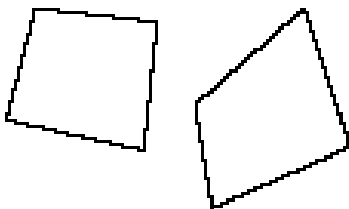
GL_TRIANGLE_STRIP



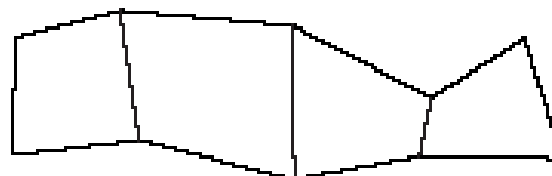
GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



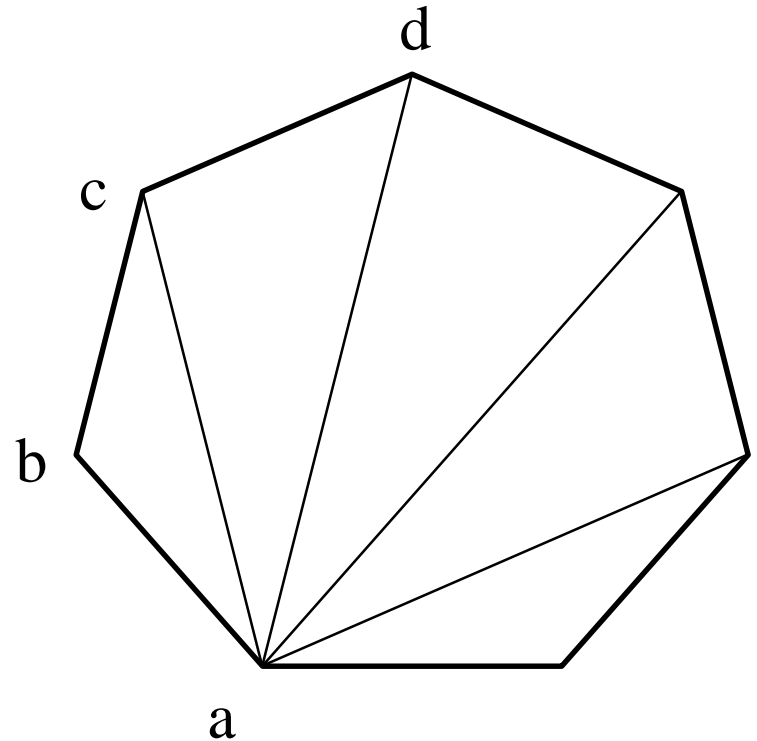
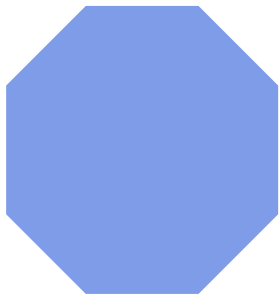


Triangulation

- Generally OpenGL breaks polygons down into triangles which are then rendered. Example

`glDrawArrays(GL_POLYGON,..)`

– convex filled polygon





Previously: Generated 3 Points to be Drawn

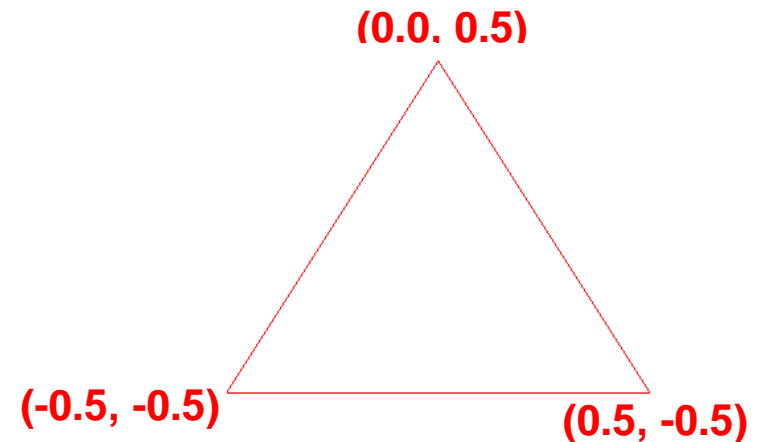
- Stored points in array `points[]`, moved to GPU, draw using `glDrawArray`

```
point2 points[NumPoints];
```

```
points[0] = point2( -0.5, -0.5 );
```

```
points[1] = point2( 0.0, 0.5 );
```

```
points[2] = point2( 0.5, -0.5 );
```

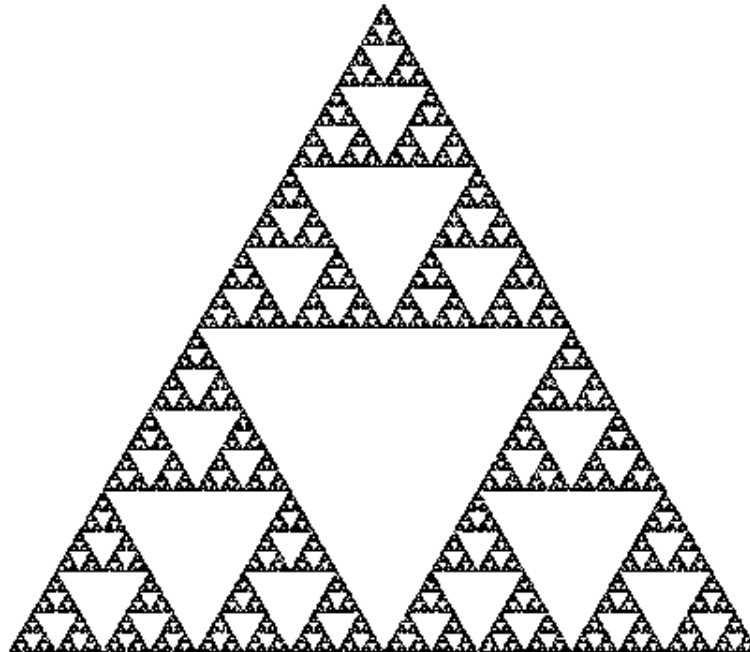


- Once drawing steps are set up, can generate more complex sequence of points algorithmically, drawing steps don't change
- Next: example of more algorithm to generate more complex point sequences



Sierpinski Gasket Program

- Any sequence of points put into array `points[]` will be drawn
- Can generate interesting sequence of points
 - Put in array `points[]`, draw!!
- Sierpinski Gasket: Popular fractal



Sierpinski Gasket



Start with initial triangle with corners (x_1, y_1) , (x_2, y_2) and (x_3, y_3)

1. Pick initial point $\mathbf{p} = (x, y)$ at random inside a triangle
2. Select one of 3 vertices at random
3. Find \mathbf{q} , halfway between \mathbf{p} and randomly selected vertex
4. Draw dot at \mathbf{q}
5. Replace \mathbf{p} with \mathbf{q}
6. Return to step 2



Actual Sierpinski Code

```
#include "vec.h"    // include point types and operations
#include <stdlib.h> // includes random number generator

void Sierpinski( )
{
    const int NumPoints = 5000;
    vec2 points[NumPoints];

    // Specify the vertices for a triangle
    vec2 vertices[3] = {
        vec2( -1.0, -1.0 ), vec2( 0.0, 1.0 ), vec2( 1.0, -1.0 )
    };
};
```



Actual Sierpinski Code

```
// An arbitrary initial point inside the triangle
points[0] = point2(0.25, 0.50);

// compute and store N-1 new points
for ( int i = 1; i < NumPoints; ++i ) {
    int j = rand() % 3;    // pick a vertex at random

    // Compute the point halfway between the selected vertex
    // and the previous point
    points[i] = ( points[i - 1] + vertices[j] ) / 2.0;
}
```



References

- Angel and Shreiner, Interactive Computer Graphics, 6th edition, Chapter 2
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition, Chapter 2