CSCD 396

Beginning Graphics

Other function used by OpenGL to update the drawing

- glutPostReDisplay();
  - In the next iteration through **glutMainLoop**, the window's display callback will be called to redisplay the window.
  - You need to call this function if there’s an update via any other callback functions, i.e., **glutKeyboardFunc**;

Wireframe demo!!

glPointSize/ glLineWidth

- void glPointSize(GLfloat size):
  - Specifies the diameter of rasterized points. The initial value is 1.

- void glLineWidth(GLfloat width):
  - Specifies the width of rasterized lines. The initial value is 1.

Varying Point Size in Shader

- Use the following command in your cpp file:
  - glEnable(GL_PROGRAM_POINT_SIZE)
- You need set the point size in shader as follows:
  - gl_PointSize = 50.0;
- **This will overwrite any command regarding setting point size from the application.**

Vector and Matrix Type used in Shader

Table 2.3 GLSL Vector and Matrix Types

Base Type	2D vec	3D vec	4D vec	Matrix Types		

`float`	`vec2`	`vec3`	`vec4`	`mat2`
				`mat2x2`
				`mat3x2`
				`mat4x2`
					`mat3`
					`mat2x3`
					`mat3x3`
					`mat4x3`
						`mat4`
						`mat2x4`
						`mat3x4`
						`mat4x4`

`double`	`dvec2`	`dvec3`	`dvec4`	`dmat2`
				`dmat2x2`
				`dmat3x2`
				`dmat4x2`
					`dmat3`
					`dmat2x3`
					`dmat3x3`
					`dmat4x3`
						`dmat4`
						`dmat2x4`
						`dmat3x4`
						`dmat4x4`

`int`	`ivec2`	`ivec3`	`ivec4`	—		
`uint`	`uvec2`	`uvec3`	`uvec4`	—		
`bool`	`bvec2`	`bvec3`	`bvec4`	—		

Vector and Matrix Type used in Shader

- mat3 M;
  - This will generate an identity matrix:

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

- mat3 M= mat3(4.0)
  - This will generate the following matrix:

$$\begin{bmatrix} 4.0 & 0.0 & 0.0 \\ 0.0 & 4.0 & 0.0 \\ 0.0 & 0.0 & 4.0 \end{bmatrix}$$

Vector and Matrix Type used in Shader

- A 3X3 matrix can be initialized in the following ways:
  - mat3 m = mat3(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,8.0, 9.0);

- This will generate the following matrix

$$\begin{bmatrix} 1.0 & 4.0 & 7.0 \\ 2.0 & 5.0 & 8.0 \\ 3.0 & 6.0 & 9.0 \end{bmatrix}$$

A Few Basic GLSL Type Modifiers

- const : variables are treated as read-only
- in : variable is an input to the shader
- out : variable is an output from the shader
- uniform :variable can’t be changed by the shaders; shared between all the shader stages in a program; can’t be written (We’ll learn about this later)

Rendering

```
void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    glBindVertexArray(rect_vao);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
    glFlush();
}
```

Clears the color buffer to its current clearing value

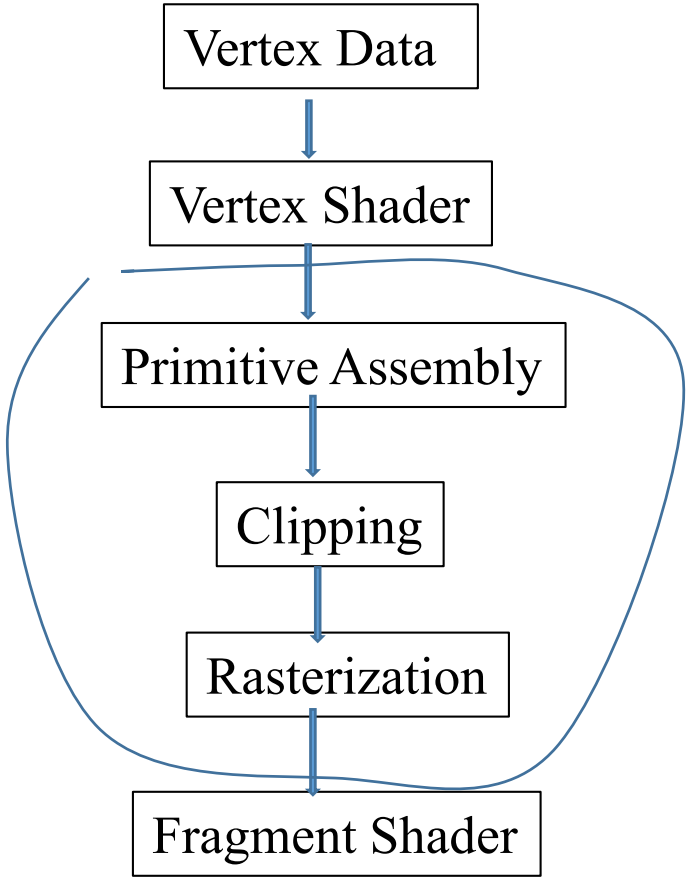
Makes the vertex array to be used as current vertex data

void **glDrawArrays**(GLenum *mode*, GLint *first*, GLsizei *count*);

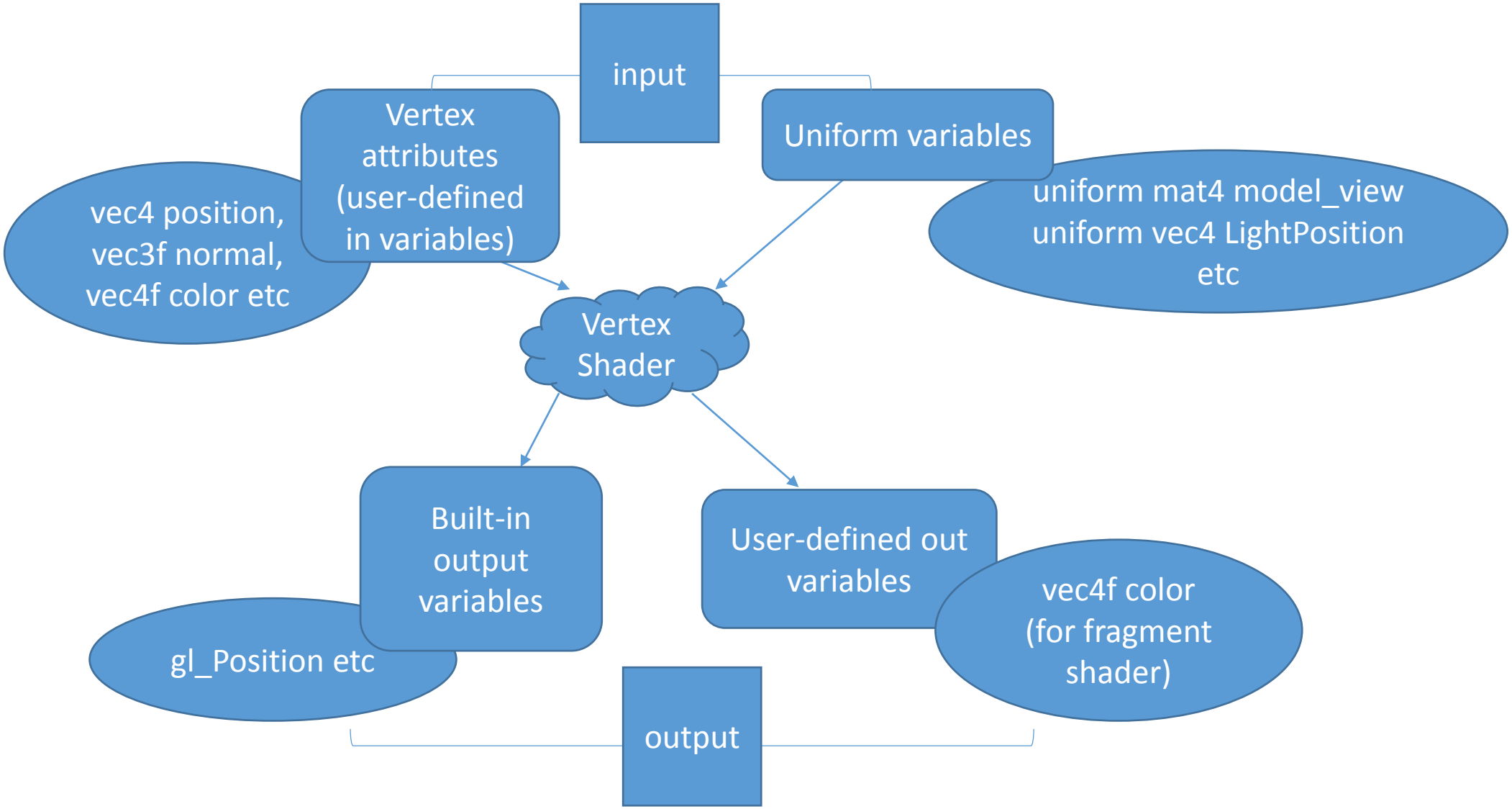
'mode' implies different primitives, i.e. GL_POINTS, GL_LINES, GL_TRIANGLES, GL_TRIANGLE_FAN etc.
'first' is the vertex array starting index
'count' is the number of vertices.

Forces previously issued OpenGL commands to begin execution

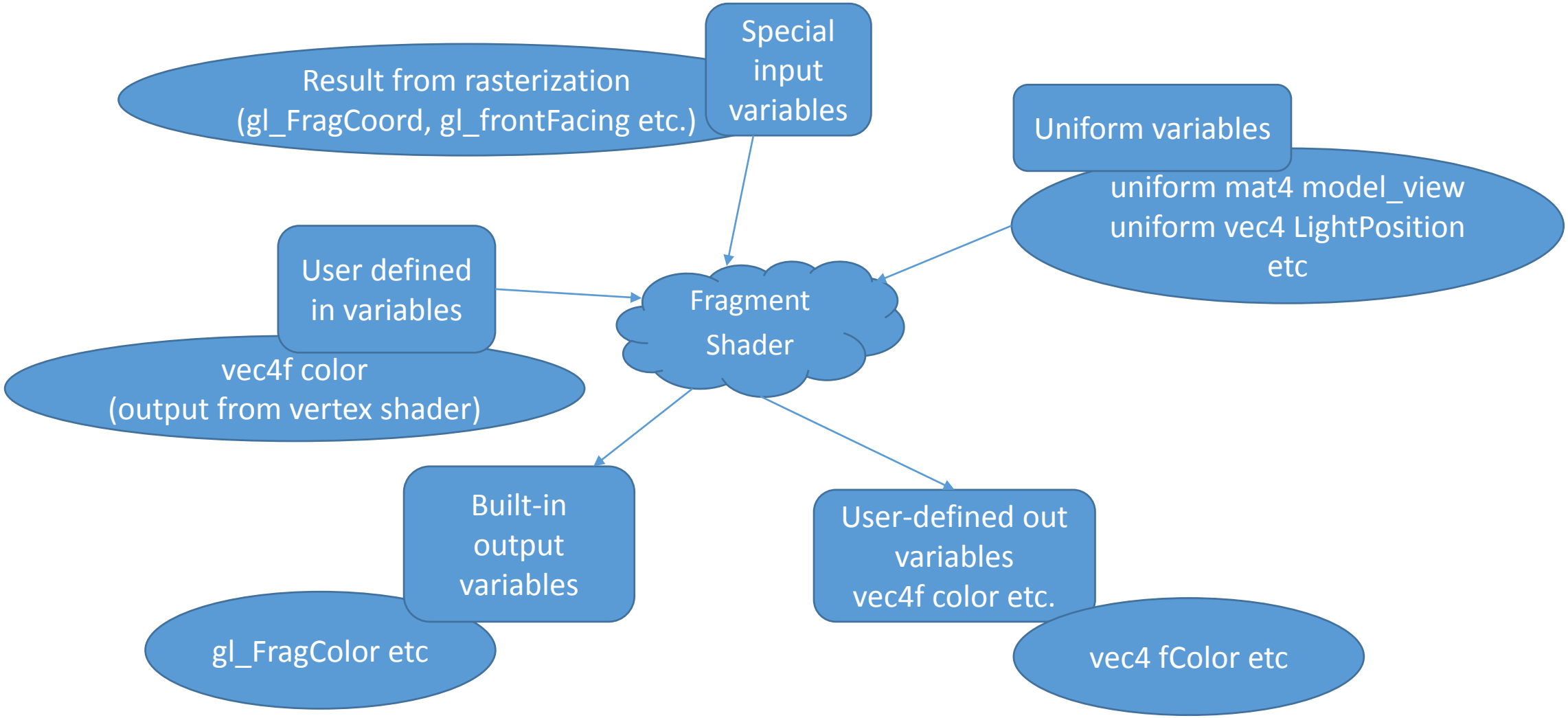
OpenGL Pipeline Using Shaders



A Simplified Vertex Processor



A Simplified Fragment Processor



Vertex Attributes

- Sent from application to the vertex shader ;
- Global variables defined in vertex shader as ‘in’ variables;

```
#version 430 core
layout (location = 0) in vec4 position;
layout (location = 1) in vec4 color;
out vec4 vs_fs_color;
void main(void)
{
    vs_fs_color = color;
    gl_Position =  position;
}
```

- Values are provided per vertex; i.e. not constant across primitives;
- Holds per vertex information like position, color, normal etc.

Vertex Attributes

- When the program object and shaders are linked (see ‘initShaders’), attributes are bound to corresponding attribute indices;
  - For example, the first parameter in **glVertexAttribPointer**, takes the index of a vertex attribute variable in shader
    - *void **glVertexAttribPointer**(GLuint index, GLint size, GLenum type, GLboolean normalized,GLsizei stride, const GLvoid *pointer);*
    - Later that vertex attribute is enabled by ***glEnableVertexAttribArray(index)***
- There can be a number of vertex attributes;
- Each attribute needs to have corresponding index.

Vertex Attribute Index

- Attribute indices can be assigned by
  - OpenGL program API;
  - Shader using ‘layout’ qualifier;

Vertex Attribute Index

Define indices in cpp file

- glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, NULL);
- glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, NULL);

Use layout qualifier in the shader

```
// "position" and "normal" are regular vertex attributes

    layout (location = 0) in vec4 position;
    layout (location = 1) in vec4 color;
```

OpenGL Drawing Commands

- void **glDrawArrays**(GLenum *mode*, GLint *first*, GLsizei *count*);
- Parameters

- **mode** specifies what kind of primitives to render. Symbolic constants GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_TRIANGLES etc.

- **first** specifies the starting index in the arrays.

- **count** specifies the number of indices to be rendered.

OpenGL Drawing Commands

- void **glDrawElements**(GLenum *mode*, GLsizei *count*, GLenum *type*, const GLvoid **indices*);

- Parameters
  - **mode**: specifies what kind of primitives to render. Symbolic constants GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_LINE_STRIP_ADJACENCY, GL_LINES_ADJACENCY, GL_TRIANGLE_STRIP, GL_TRIANGLE etc.
  - **count** specifies the number of elements to be rendered.
  - **type** specifies the type of the values in indices. Must be one of GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, or GL_UNSIGNED_INT.
  - **indices** specifies a pointer to the location where the indices are stored.