

OpenGL 3&4 and Vulkan

CSCI 4239/5239

**Advanced Computer Graphics
Spring 2026**

What is new in OpenGL 3&4

- Additional shaders
 - Geometry (OpenGL 3.2)
 - Tessellation (OpenGL 4.0)
 - Compute (OpenGL 4.3)
- New syntax for passing variables
 - “in” from previous stage
 - “out” to next stage
 - Deprecating most predefined variables
- Building objects from vertex arrays
- Deprecating OpenGL transformations

Deprecated Features

- glBegin() glEnd()
 - Use vertex buffer objects instead
- glTranslate() glRotate() glScale()
 - Use vmath or glm or roll your own
- Display lists
- *Deprecated features remain available through the compatibility profile, but are not available in the core profile which is common with OpenGL ES*

Vertex Arrays

- Pass all the vertex values to OpenGL as a single array of values rather than numerous calls to `glVertex`, `glColor`, etc.
- Draw objects using `glDrawArrays()` or `glDrawElements()`

Vertex Buffer Objects (VBO)

- Stored on the GPU
- Addressed analogous to textures
 - `glGenBuffers()` - generate unique names
 - `glBindBuffer()` - select buffer
 - `glBufferData()` - copy data to buffer
 - `glBufferSubData()` - copy partial data
 - `glEnableVertexAttribArray()` - enable array
 - `glVertexAttribPointer()` - map attribute

glVertexAttribPointer(index, size, type, normalized, stride, pointer)

- index: 0,1,.. must match layout
- size: dimension of variable (1,2,3,4)
- type: variable type (e.g. GL_FLOAT)
- normalize: if true map integers to 0-1
- stride: bytes between data values
- pointer: offset of data values (in bytes)
- *The data comes from the current vertex buffer selected using glBindBuffer()*
- *Activate glEnableVertexAttribArray(index)*

Vertex Array Objects (VAO)

- Stored on the GPU
- Remembers how the buffers and arrays map so you don't have to do it again
- Create with `glGenVertexArrays`
- Apply with `glBindVertexArray`
- Added in GL3

OpenGL 3&4 Adoption

- On the desktop, you can do gradual adoption
 - OpenGL 3&4 style shaders with glBegin()
 - VBOs with OpenGL 2 shaders
 - The compatibility profile supports both
- With OpenGL ES it is all or nothing
 - Smaller footprint
 - Fewer legacy implementations
- Use whatever combination you want
 - My examples use whatever is convenient

Vulkan

- Vulkan is what would have been GL5
- Breaks backwards compatibility, but strongly resembles OpenGL
- Requires you to be very explicit
- Close to the metal, little abstractions
- Super verbose, very steep learning curve
- Requires tons of scaffolding