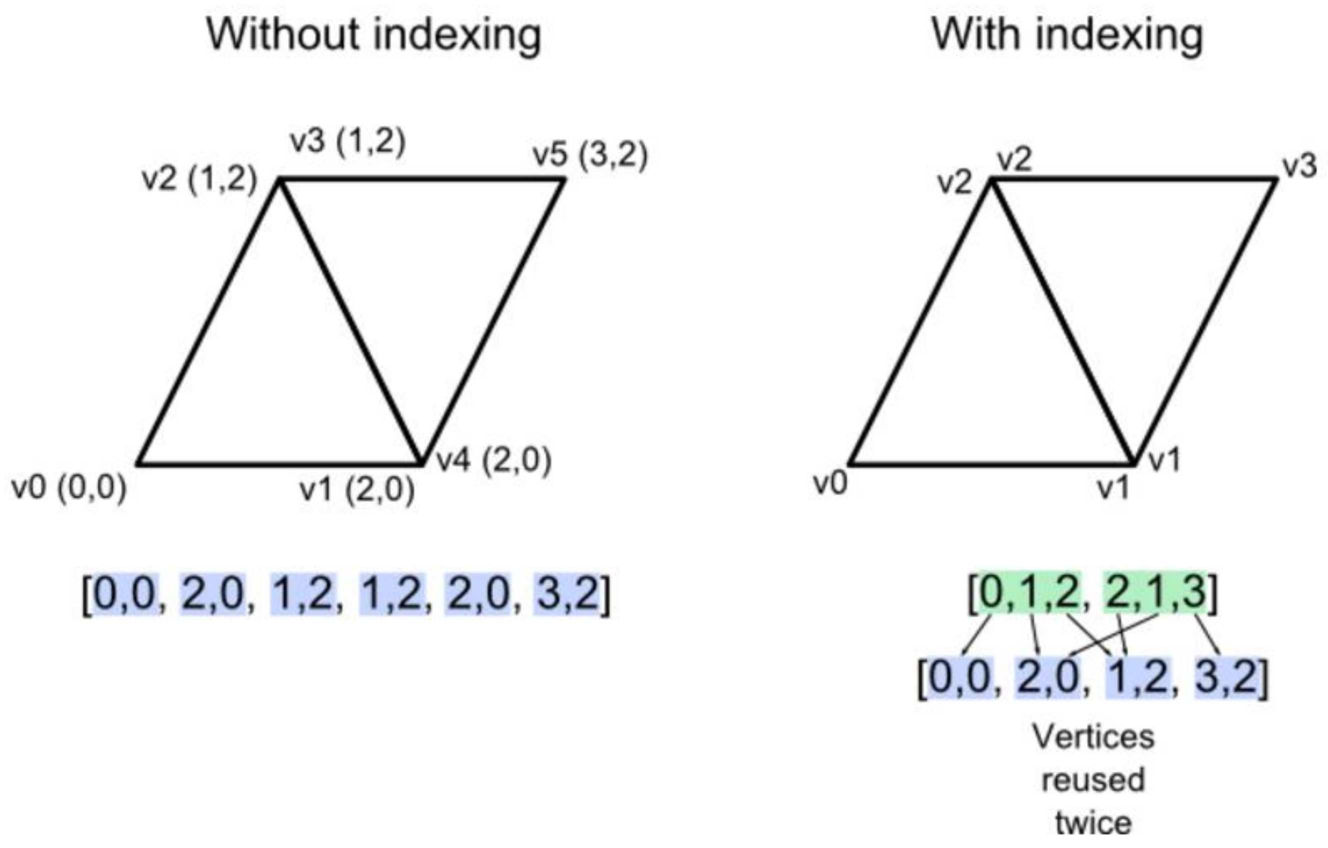
COMPUTER GRAPHICS

California State University, Fresno

VBO

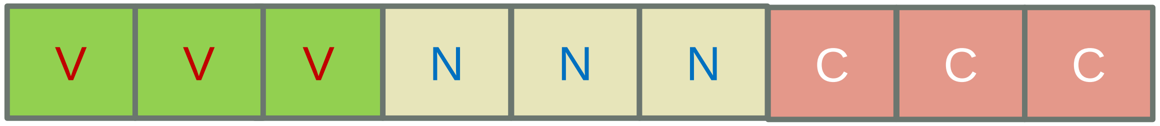
The principle of indexing



Formatting VBO Data

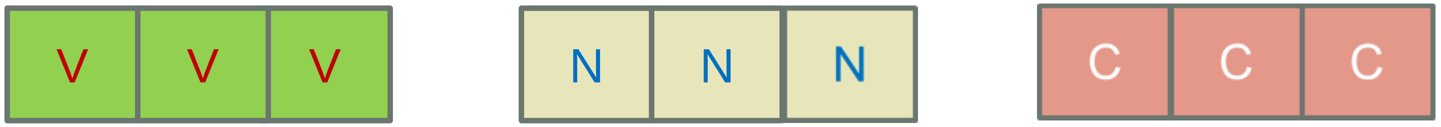
- VBOs are quite flexible in how you use them
- There are a number of ways you can represent vertex attribute data in VBOs
    - *Let V - vertices*
    - *Let C - color*
    - *Let N – normal*

VBO Buffer sample

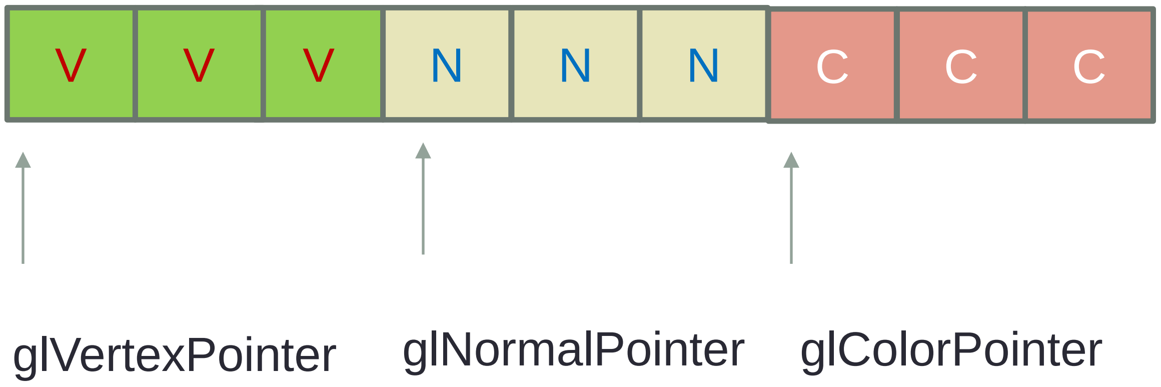


Storage Patterns

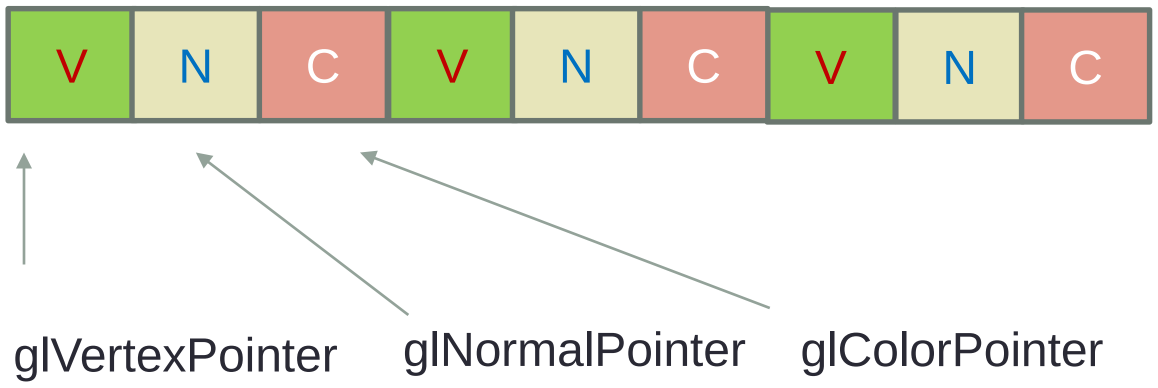
- (VVVV) (NNNN) (CCCC)
  - Allocate a separate VBO per vertex attribute
  - Same as using arrays of data in OBJ loader assignement
  - Seperate VBO per each type



- (VVVVNNNNCCCC)
  - Store the vertex attribute blocks in a batch
  - Same block and pack them all in the same VBO
  - Specifying the vertex attributes via glVertexAttribPointer
  - Pass byte offsets into the VBO to the pointer parameters



- (VNCVNCVNCVNC)
  - Interleave the vertex attributes for each vertex in a batch
  - Store each of these interleaved vertex blocks sequentially
  - Specifying the vertex attributes via glVertexAttribPointer
  - Pass byte offsets into the VBO to the pointer parameters



Best Practice

- Minimize the number of glVertexAttribPointer calls (or glVertexAttribFormat where available)

- This will make glDrawArrays and other array-style rendering faster

- Meshes with less than 65536 vertices can be stored sequentially in the same vertex buffer
- Since indices (GLushort) can be used for indexing
  16 bit number ➔ 2^{16} = **65535**

Multiple VBO

- Try using minimum number of VBOs possible
- This will enhance performances
- In Case of using Dynamic VBOs this may be differ

Example of multiple VBOs

```
//Binding the vertex
 glBindBuffer(GL_ARRAY_BUFFER, vertexVBOID);
 glVertexPointer(3, GL_FLOAT, sizeof(float)*3, NULL);
//Vertex start position address

 //Bind normal and texcoord
 glBindBuffer(GL_ARRAY_BUFFER, otherVBOID);
 glNormalPointer(GL_FLOAT, sizeof(float)*6, NULL);

 //Normal start position address
 glTexCoordPointer(2, GL_FLOAT, sizeof(float)*6, sizeof(float*3));
 //Texcoord start position address
```

Creating VBO

- First Three Steps

1. Generate a new buffer object with glGenBuffersARB()
2. Bind the buffer object with glBindBufferARB()
3. Copy vertex data to the buffer object with glBufferDataARB()

glGenBuffersARB()

- creates buffer objects and returns the identifiers of the buffer objects

  - Parameters

    1. The number of buffer objects to create
    2. The address of a GLuint variable or array to store a single ID or multiple IDs

Ex: `void glGenBuffersARB(GLsizei n, GLuint* ids)`

glBindBufferARB()

- Connect the buffer object with the corresponding ID before using the buffer object

Parameters

1. Target to tell VBO whether this buffer object will store vertex array data or index array data.
   - *target* flag assists VBO to decide the most efficient locations of buffer objects. Ex: system memory, video memory etc.

2. The address of a GLuint variable or array to store a single ID or multiple IDs

Ex: `void glBindBufferARB(GLenum target, GLuint id)`

glBufferDataARB()

- Copy the data into the buffer object when the buffer has been initialized

Parameters

1. *Target: target* would be GL_ARRAY_BUFFER_ARB or GL_ELEMENT_ARRAY_BUFFER_ARB
2. *Size:* number of bytes of data to transfer
3. *Source data:* pointer to the array of source data
4. *Usage flag:* hint for VBO to provide how the buffer object is going to be used: *static*, *dynamic* or *stream*, and *read*, *copy* or *draw*

```
Flags :

GL_STATIC_DRAW_ARB
GL_STATIC_READ_ARB
GL_STATIC_COPY_ARB
GL_DYNAMIC_DRAW_ARB
GL_DYNAMIC_READ_ARB
GL_DYNAMIC_COPY_ARB
GL_STREAM_DRAW_ARB
GL_STREAM_READ_ARB
GL_STREAM_COPY_ARB
```

Ex:

```
void glBufferDataARB(GLenum target, GLsizei size, const void* data, GLenum usage)
```

Flags

- *Static:* The data in VBO will not be changed
  (specified once and used many times)
- *Dynamic:* The data will be changed frequently
  (specified and used repeatedly)
- *Stream: T*he data will be changed every frame
  (specified once and used once)
- *Draw: T*he data will be sent to GPU in order to draw
- *Read: T*he data will be read by the client's application
- *Copy: T*he data will be used both drawing and reading

glBufferSubDataARB()

- Same as glBufferDataARB()
- Used to copy data into VBO
- Replaces a range of data into *the existing buffer*, starting from the given offset

Ex:

```
void glBufferSubDataARB(GLenum target, GLint offset, GLsizei size, void* data)
```

glDeleteBuffersARB()

- Can delete a single VBO or multiple VBOs
- After a buffer object is deleted, its contents will be lost

Ex:

```
void glDeleteBuffersARB(GLsizei n, const GLuint* ids)
```

- The following code is an example of creating a single VBO for vertex coordinates.
- We can delete the memory allocation for vertex array in your application after you copy data into VBO.

```
// ID of VBO
GLuint vboId;

// create vertex array ...
GLfloat* vertices = new GLfloat[vCount*3];

// generate a new VBO and get the associated ID
glGenBuffersARB(1, &vboId);

// bind VBO in order to use
glBindBufferARB(GL_ARRAY_BUFFER_ARB, vboId);

 // upload data to VBO
glBufferDataARB(GL_ARRAY_BUFFER_ARB, dataSize, vertices, GL_STATIC_DRAW_ARB);

// it is safe to delete after copying data to VBO delete [] vertices;

...

// delete VBO when program terminated glDeleteBuffersARB(1, &vboId);
```