셰이더프로그래밍

Lecture 2

이택희

개요

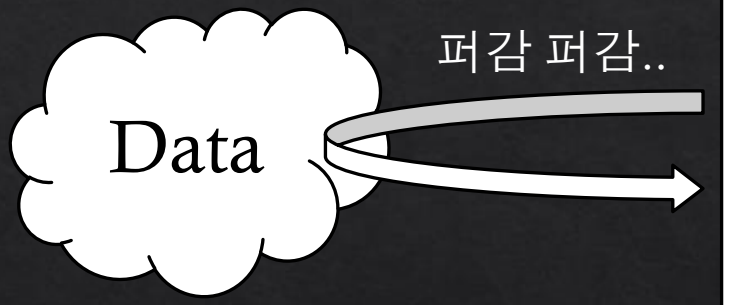
- OpenGL 렌더링 구조
- OpenGL 데이터 준비
  - Generate, Bind 구조 이해
    - Vertex Buffer Object
- OpenGL 데이터 사용
- 실습

OpenGL 렌더링 구조

OpenGL 렌더링 구조

- OpenGL 에서 렌더링은 일종의 State Machine 형태로 동작한다.

  Draw call을 줄이는 게 최적화에 관련됨. Draw 콜 한 번 자체가 밑의 렌더링 구조와 같음.



Data 설정은 렌더링
시작전 한번만 설정됨

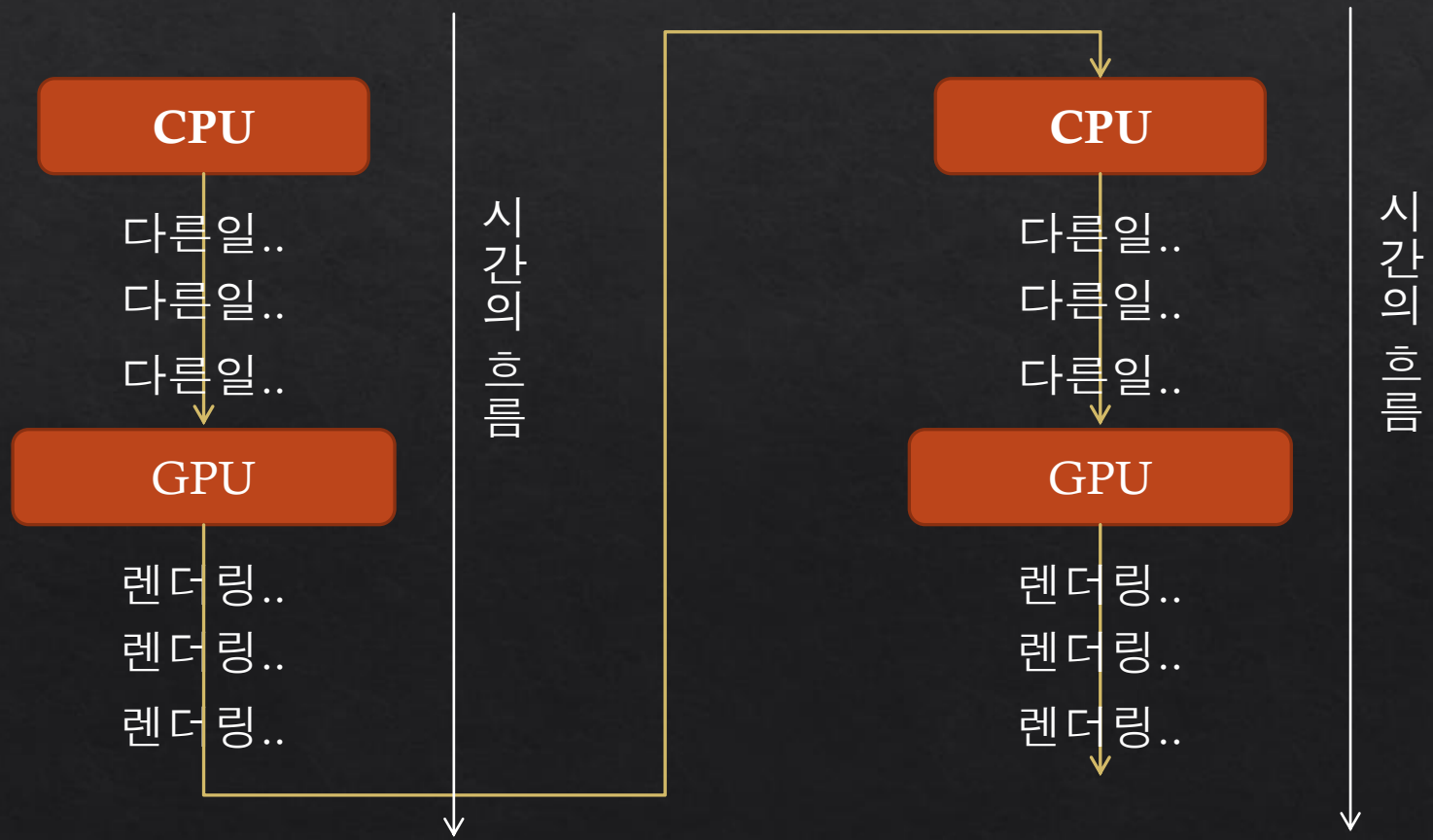
파이프라인은 설정된 데이터를
지속적으로 읽어가며 렌더링 수행

렌더링 도중에 데이터가 바뀌는 것을
허용하지 않음!

OpenGL 렌더링 구조

- 병렬화가 안되면 매우 비효율적임

게임할 때 CPU에서는 게임 로직, 물리 충돌 로직 등 담당. 아무리 빨라도 12mm/s 정도. 60hz 위해서는 16ms 안에 연산되어야. 이미 cpu에서 12mm/s을 차지한다음 GPU가 일을 한다면 GPU에서 10mm/s 만에 렌더링 한다고 해도 22mm/s 나옴. 그래서 병렬화하면 게임이 끊겨 보일 것.



OpenGL 렌더링 구조

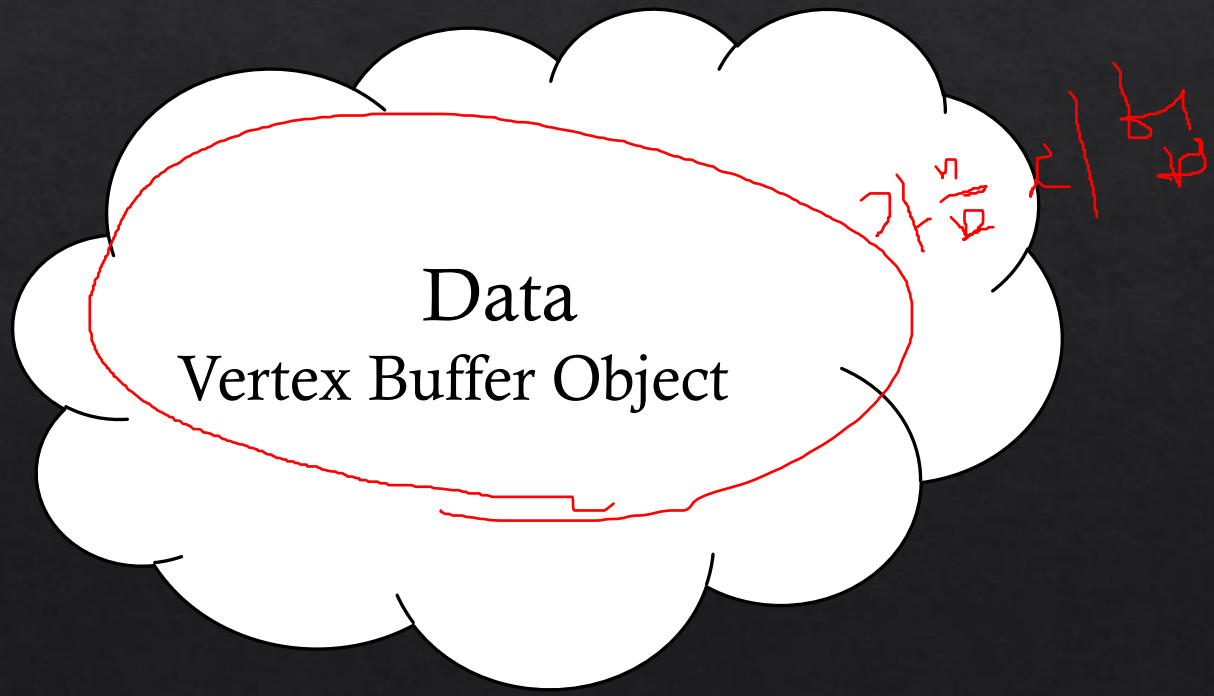
- 성능의 극대화를 위한 구조
  - CPU-GPU 병렬화에 최적화되어 있음

CPU 는 GPU에 일을 시킨 뒤에는 간섭을 안 해야 함. 간섭하면 GPU의 일이 끊어짐. 그래서 일을 시킨 뒤에는 다른 일 하는 스테이트 머신처럼 써야 함.

OpenGL 렌더링 구조

- 효율적인 렌더링을 위해 고유의 Data 형식을 가짐

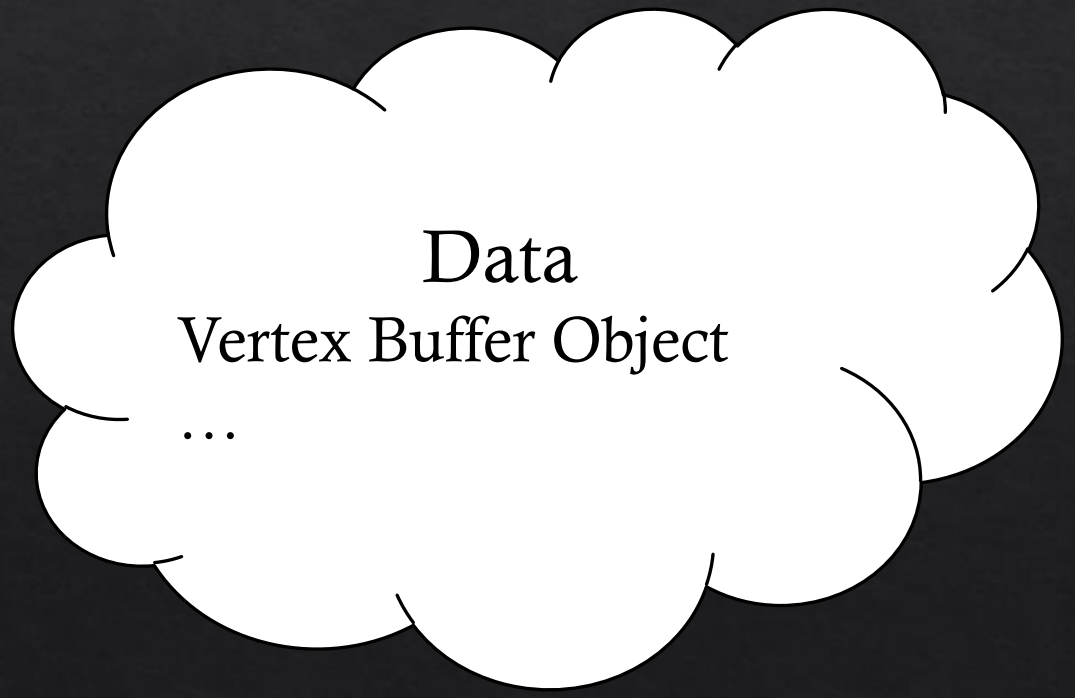
  데이터 자체는 동일한데 CPU, GPU의 자료구조가 다름. 그래서 오늘 배우는 내용이 필요하다.



OpenGL 렌더링 구조

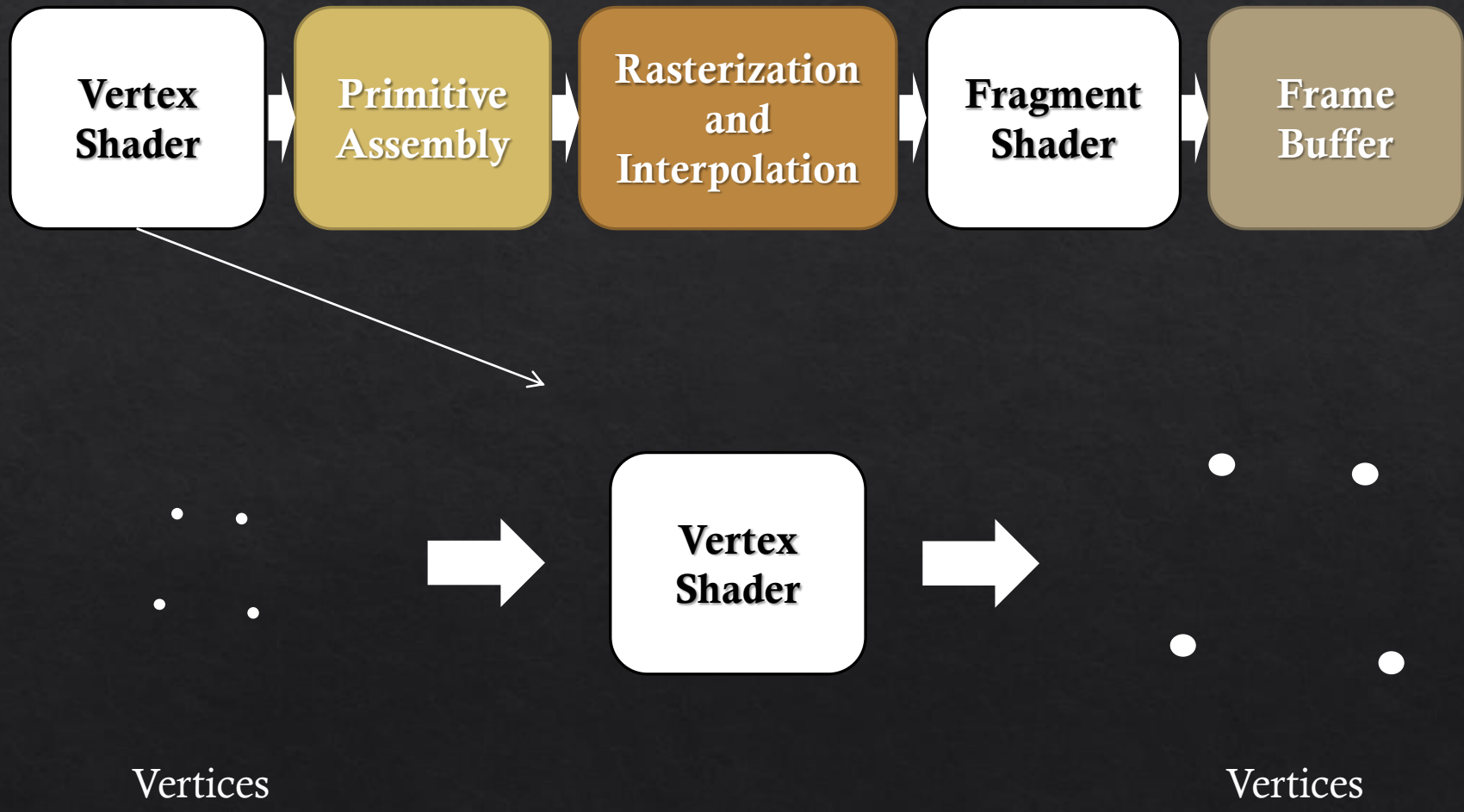
- 고유의 Data 형식을 생성하고 이를 설정하는 방법을 알아야 함

CPU에서 옮기면 화면으로 넘기면 굉장히 느리기 때문에 GPU 메모리로 넘기는 과정이 무조건 필요함. 게임의 로딩 시간이 GPU 넘기는 과정.



버텍스 쉐이더가 맨앞에 있다는 것은 최초의 입력은 버텍스일 수밖에 없다. 내부적으로 변환이 되면서 다른 스테이트에서 사용가능해짐.

OpenGL 렌더링 구조



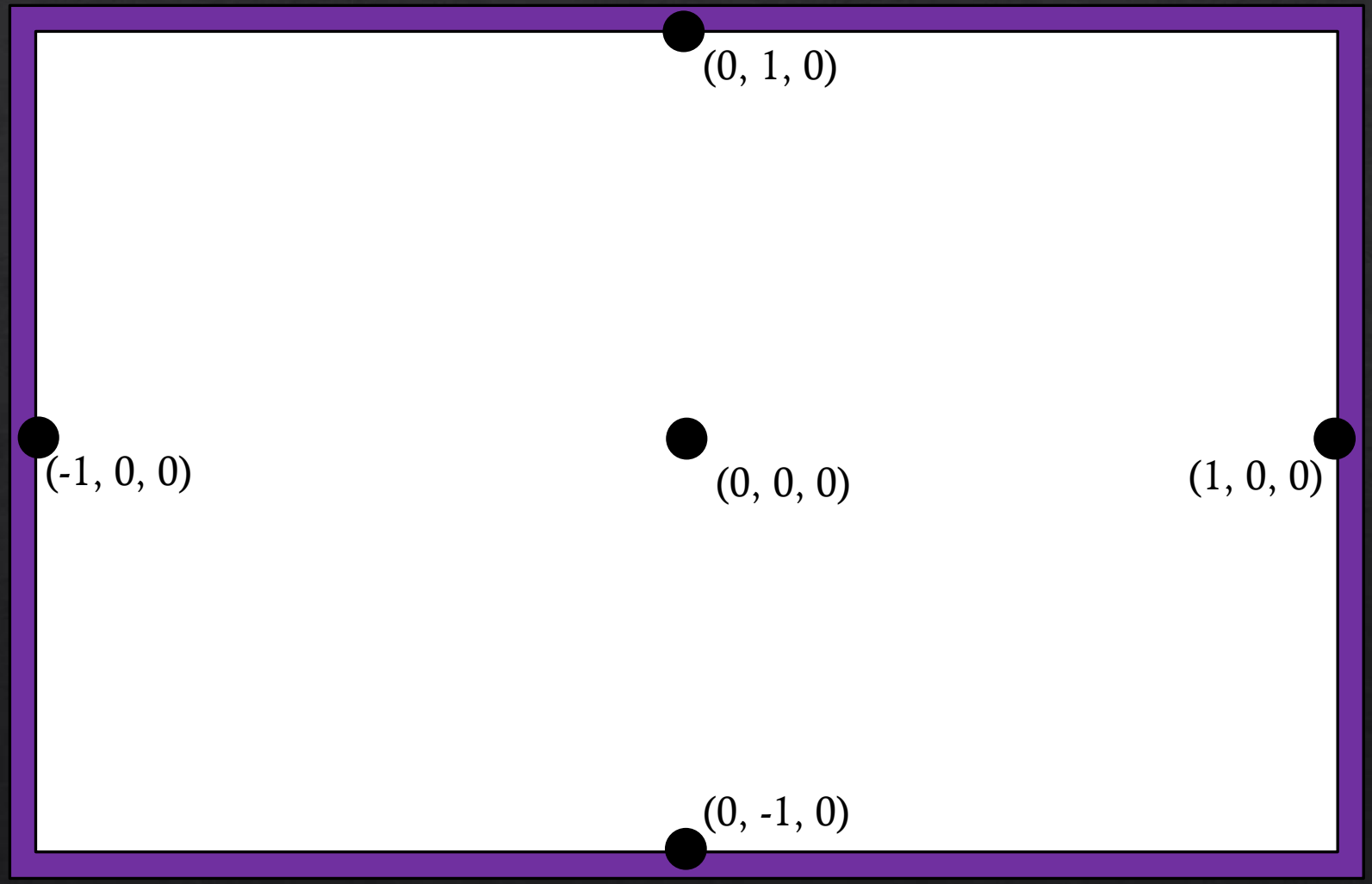
OpenGL 렌더링 구조

Vertex Shader 의 입력인 Vertices
설정 필요

OpenGL 데이터 준비

프로젝션 매트리스를 이후에는 결과적으로 (-1,1) 사이의 좌표값으로 떨어진다. 우리는 따로 프로젝션 매트릭스 안 쓸 것이기 때문에 -1, 1 사이로만 좌표를 쓸 것이다.

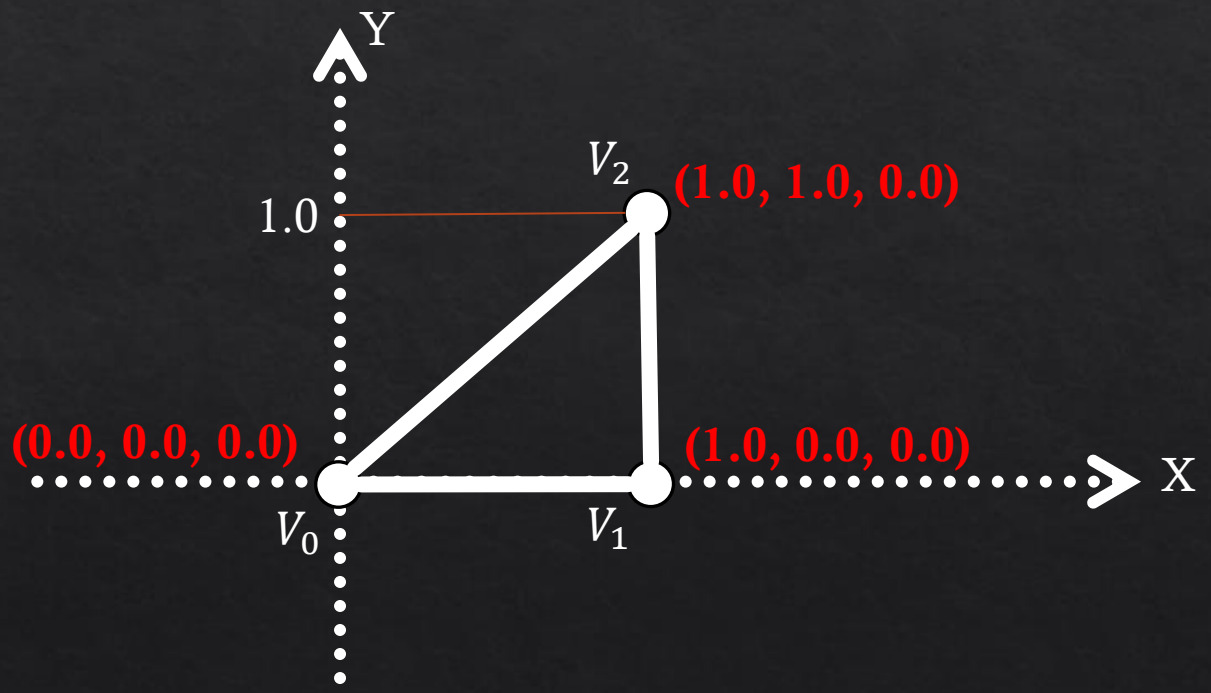
OpenGL 데이터 준비



투영 매트릭스에 따라 달라질 수 있으나 일단 기본 좌표계 기반으로 진행

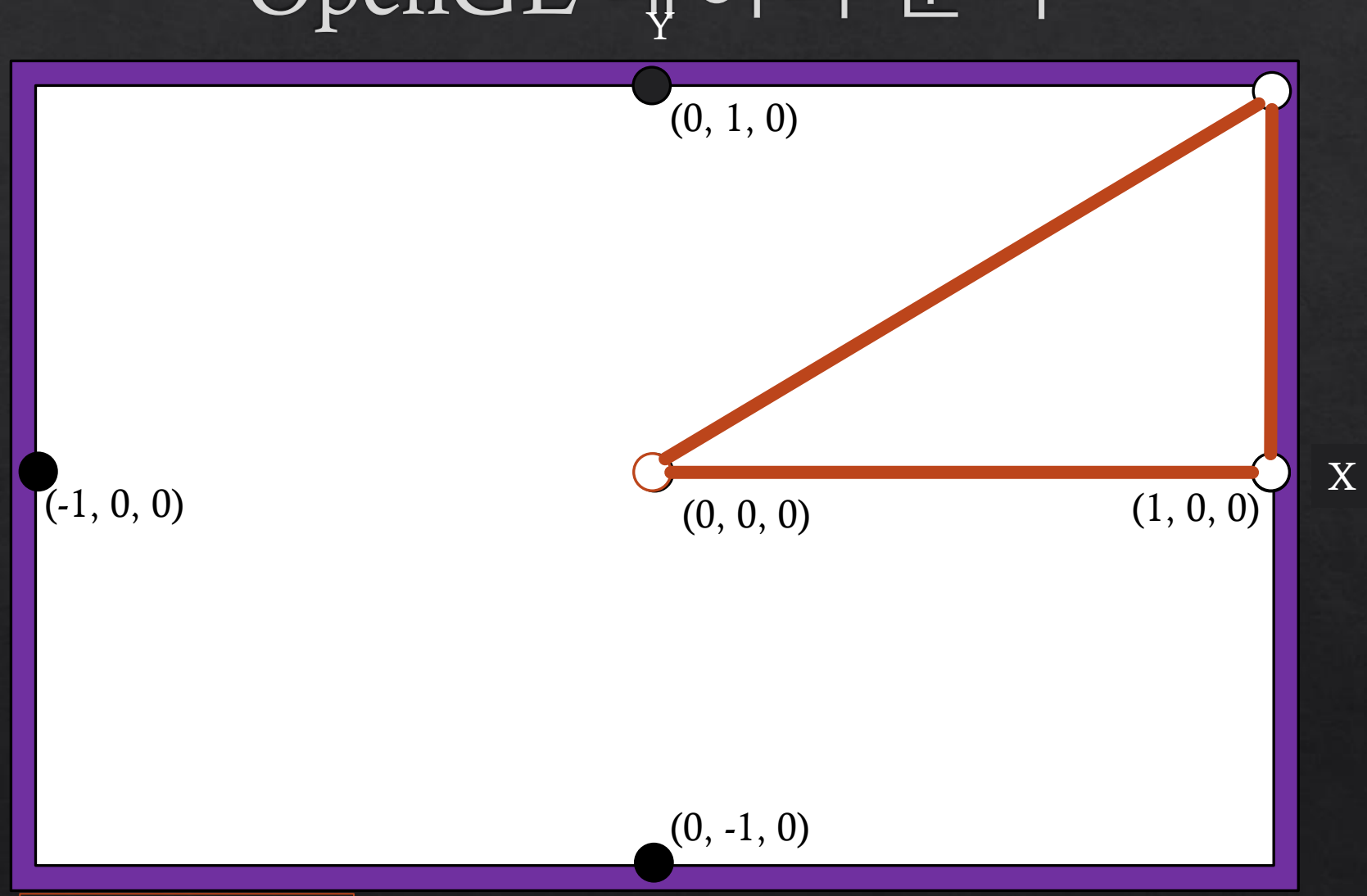
OpenGL 데이터 준비

Vertices !



시계 방향이냐 반시계 방향이냐에 따라 그리는 순서가 달라지는데, OpenGL은 오른손좌표계. 은면을 위해 필요한 과정이다.

OpenGL 데이터 준비



V_0 (0.0, 0.0, 0.0) V_1 (1.0, 0.0, 0.0) V_2 (1.0, 1.0, 0.0)

OpenGL 데이터 준비

- Vertex 데이터는 Array 형식으로 준비

$V_0(0.0, 0.0, 0.0)$			$V_1(1.0, 0.0, 0.0)$			$V_2(1.0, 1.0, 0.0)$		
0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0

vertex 순서가 중요함!

```
float vertices[] = {0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f};
```

CPU에 저장되어 있는 정보가 된다. 아직 GPU 이동 전.

OpenGL 데이터 준비

```
float vertices[] = {0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f};
```

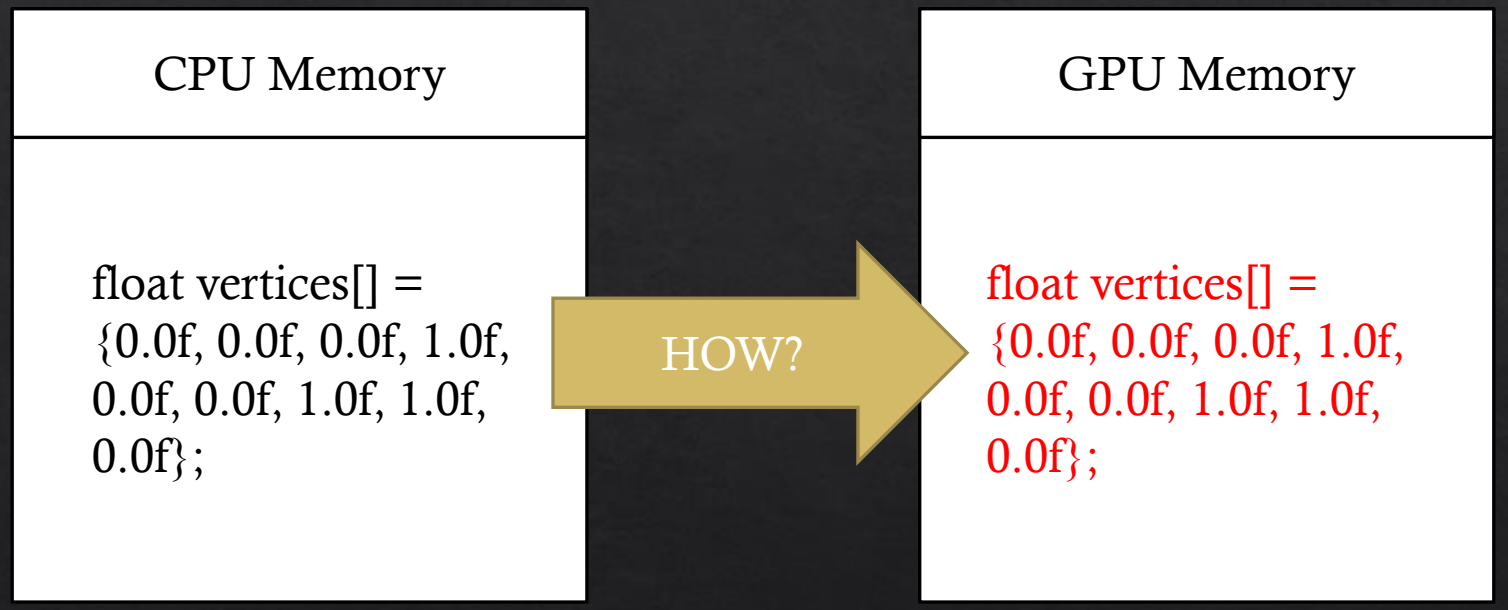
이게 바로 GPU 에서 사용이 가능할까? - 불가능. 가능해도 굉장히 비효율적.

위 array 는 어디에 저장이 되어 있을까?

위 array 를 OpenGL 에서 바로 사용 가능할까?

OpenGL 데이터 준비

GPU 로 옮기는 과정에서 Bind 가 필요하게 된다. 제너어쩌구도 필요.



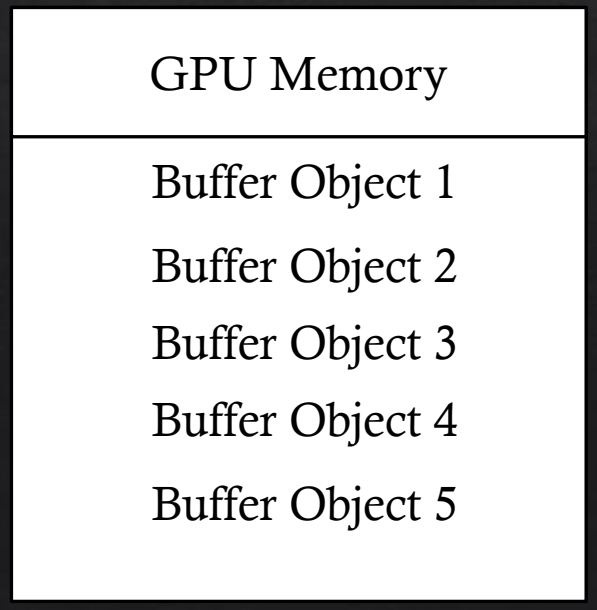
OpenGL 데이터 준비

- Vertices 를 저장하기 위한 OpenGL 고유의 형식
  - Vertex Buffer Object
  - 줄여서 VBO 라고 함

OpenGL 데이터 준비

- OpenGL Buffer Object
  - 다양한 목적으로 사용하기 위한 버퍼 오브젝트
  - Vertex 사용을 위한 용도로 생성하게 되면,
    - Vertex Buffer Object 라 칭한다

버퍼 오브젝트긴 한데 버텍스 저장 용도로 만들어서 버텍스 버퍼 오브젝트임.
GPU 메모리 외부에서는 아이디로 참조 가능. 버텍스 오브젝트의 아이디를 알면 된다.



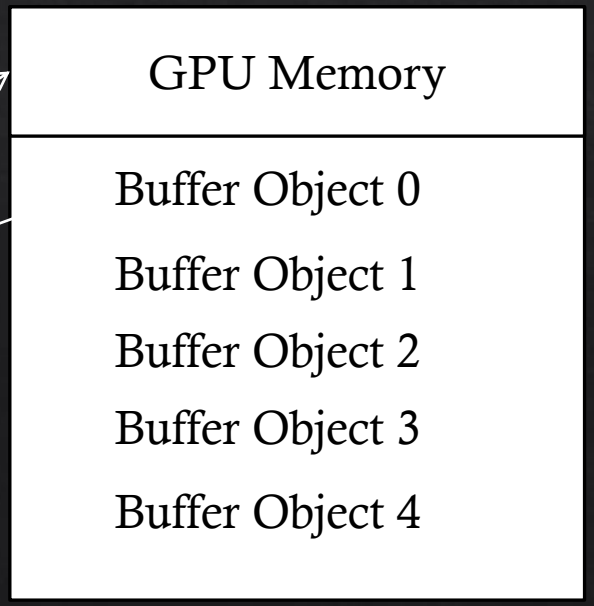
OpenGL 데이터 준비

- glGenBuffers(GLsizei n, GLuint *ids)
  - Buffer Object 를 생성하고 Object ID 를 ids 에 넣어줌
  - Object ID는 이후 실제 데이터를 CPU→GPU로 올릴 때 사용

이런식으로 사용하면 1개만 만들어라. 이후에는 0번 아이디, 1번 아이디 이런 식으로 리턴.

```
GLuint VBO;
glGenBuffers(1, &VBO);


// 1 ➔ VBO
```



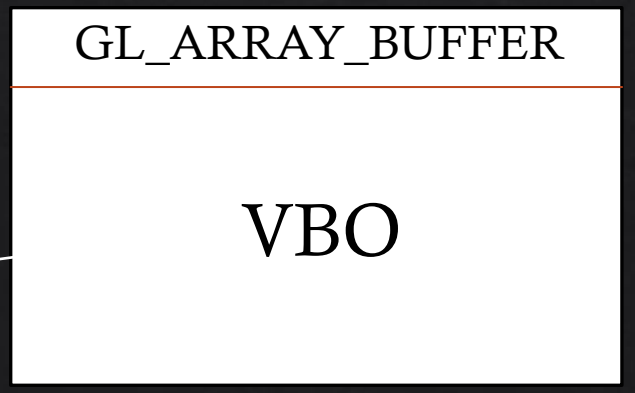
OpenGL 데이터 준비

- glBindBuffer(GLenum target, GLuint id);
  - 생성된 VBO 를 ID 를 사용하여 Bind 함
    - Bind 란?
      - 실제 OpenGL에서 작업할 대상의 형태와 용도를 구체화 해 주는 것
      - 데이터를 올리려고 하는데 그 데이터가 array 형식의 buffer를 가진다면 GL_ARRAY_BUFFER 를 사용

```
GLuint VBO;
glGenBuffers(1, &VBO);
// 0 ➔ VBO

glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

버퍼 오브젝트(VBO)를 어레이 버퍼로 바인드하면 이제부터 어레이버퍼로 사용가능하다는 뜻.
어레이 형태로 올리려면 바인드를 하고 사용해야 한다. 안 하면 마지막 배열에 이상한짓 됨.



OpenGL 데이터 준비

- glBufferData(GLenum target, GLsizeiptr size, const GLvoid *data, GLenum usage);
  - Bind된 VBO 에 데이터를 할당

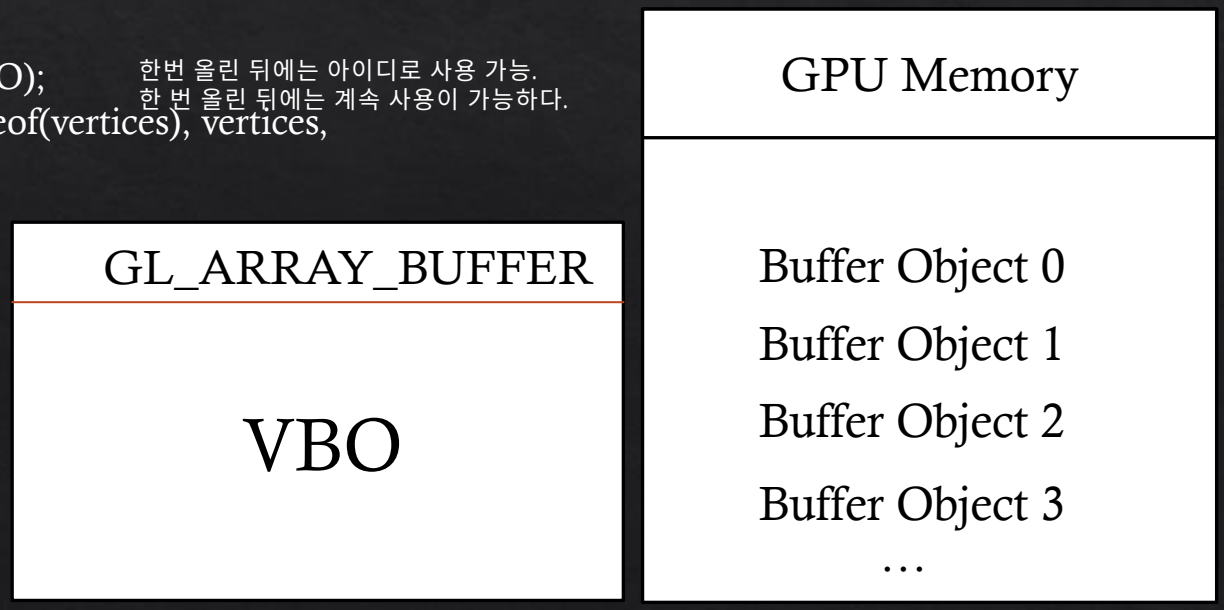
초당 4기가 정도로 데이터를 GPU에 넘기게 된다. 1억만개 정도의 엄청 큰 버텍스 버퍼를 올리게 되면 오래 걸릴 텐데, 오래 걸리는 것은 보통 glBufferData에서 오래 걸리게 된다. ?
보통 다들 간과하는데 연구가 필요한 부분이다. 노드 밸런싱 만드는 게 중요하다. 주변영역만 잘 보이도록. 필요한 부분만 로딩하고, 일단 플레이어에게 보여준 뒤에 나머지까지 로딩.

```
GLuint VBO;
glGenBuffers(1, &VBO);
// 0 ➔ VBO

glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);
```

한번 올린 뒤에는 아이디로 사용 가능.
한 번 올린 뒤에는 계속 사용이 가능하다.

```
float vertices[] =
{0.0f, 0.0f, 0.0f, 1.0f,
0.0f, 0.0f, 1.0f, 1.0f,
0.0f};
```



OpenGL 데이터 준비

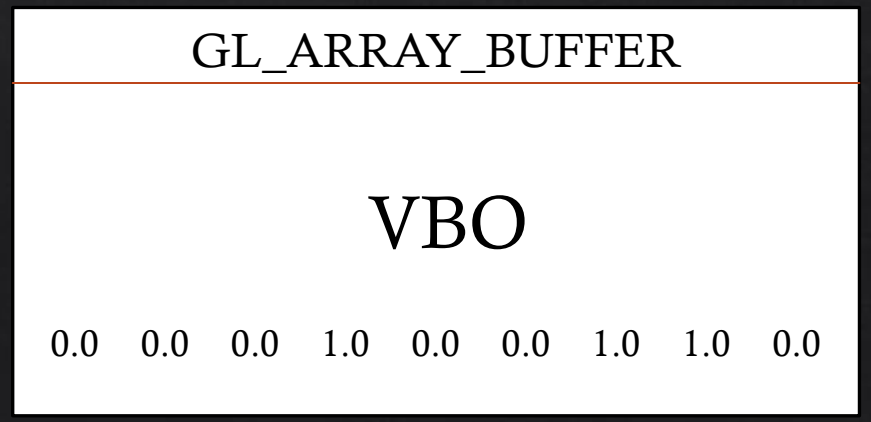
OpenGL 데이터 사용

OpenGL 데이터 사용

- 일단 BIND
  - OpenGL 데이터 생성 시 이미 Bind 했으나
    - 중간에 다른 오브젝트가 BIND 되었을 가능성 마지막으로 bind 된 애만 남아있는 위험성이 존재함. 바인드가 잘못되면 스키닝이 잘못돼서 갑자기 뚝딱이는 경우 생김.
  - OpenGL 은 종류 당 (GL_ARRAY_BUFFER 같은) 하나의 오브젝트 만 Bind 허용

바인드가 되어있는지 확인해야 함.
어트리뷰트란 버텍스 셰이더 요소를 말함.

```
glEnableVertexAttribArray(….);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

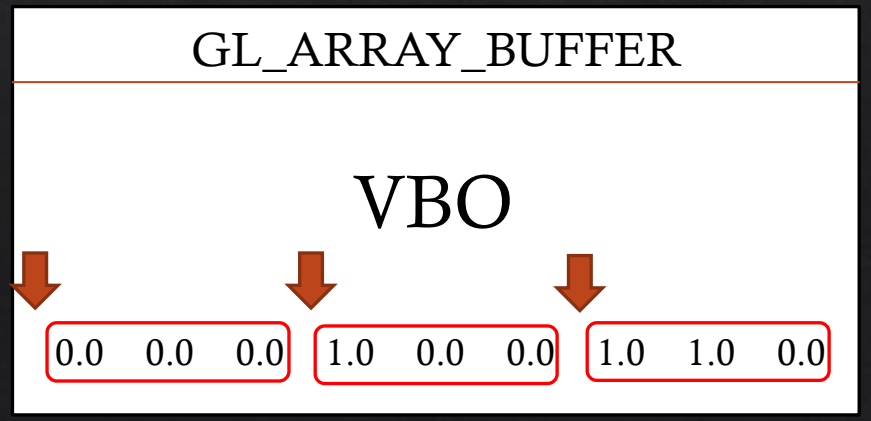


glEnableVertexAttribArray → 이 함수에 대한 설명은 다음 시간에..

OpenGL 데이터 사용

- glVertexAttribPointer(GLuint index, GLint size, GLenum type, GLboolean normalized, GLsizei stride, const GLvoid *pointer);
- Draw 시 데이터를 읽어갈 단위의 크기 및 시작점 설정

GPU에서 어떤 형식으로 해석할 건지를 여기서 정해줌. int인지, boolean인지, 등등등. 버텍스란 정보다. UV, ID, 좌표 등등 다양한 정보. 이 정보가 어떤 형식으로 넘어가는지 지정해 주는 곳이 여기다.

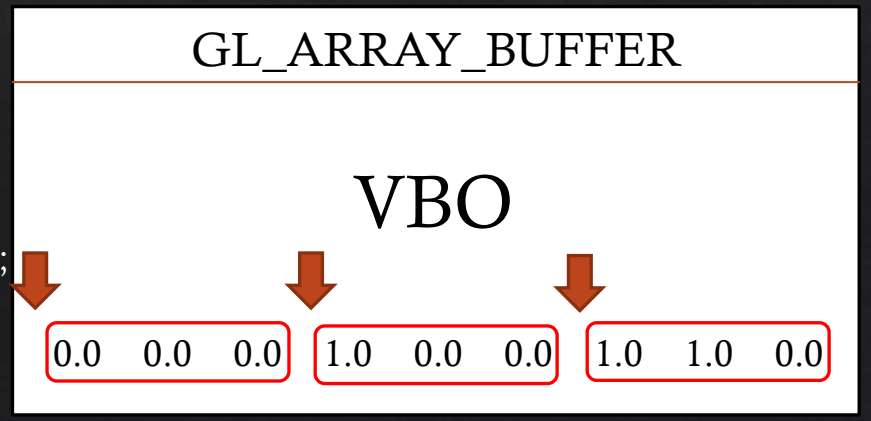



```
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
```

OpenGL 데이터 사용

- glDrawArrays(GLenum mode, GLint first, GLsizei count);
  - 어떠한 Primitive 로 구성할 것인지 선택
  - Vertex 몇 개를 그릴 것인지 입력
  - 이 함수 호출 즉시 GPU 가 동작

```
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
glDrawArrays(GL_POINTS, 0, 1);
```



실습

실습

- GLSLBase 프로젝트 사용
  - 오늘 배운 내용 구현