

셰이더 프로그래밍

Lecture 2

이택희

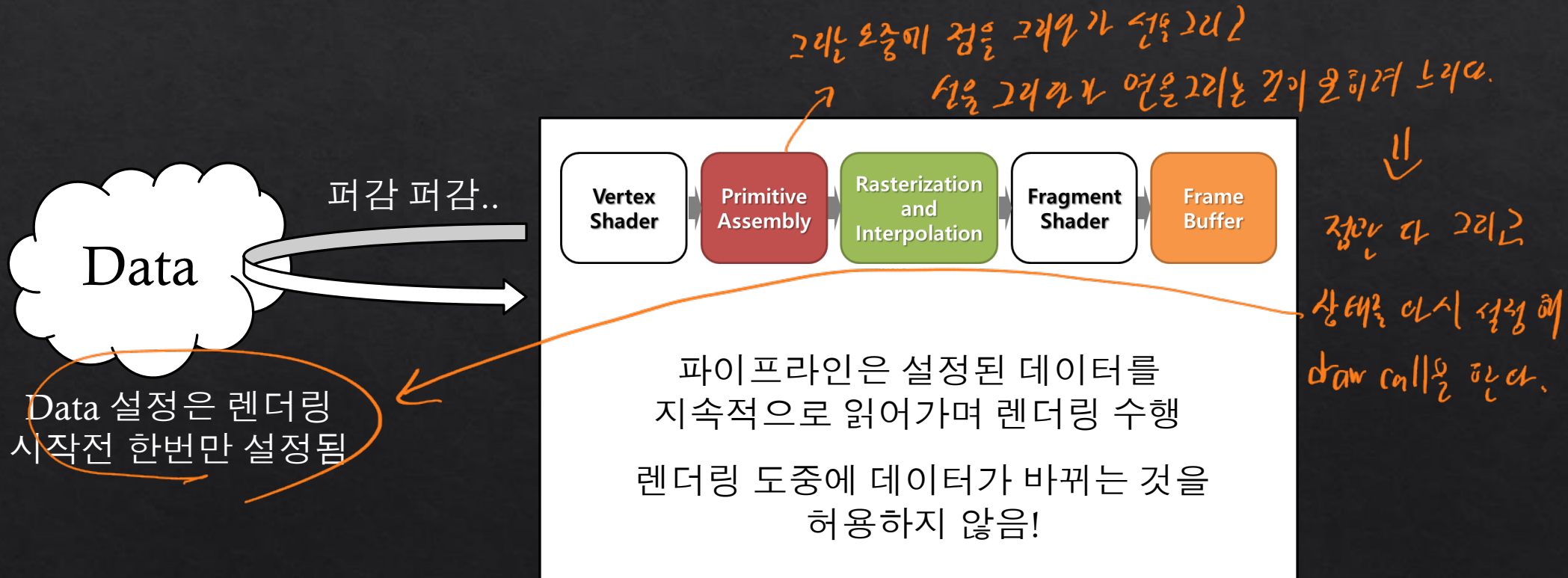
개요

- ◇ 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.



OpenGL 렌더링 구조

◇ 병렬화가 안되면 매우 비효율적임

60 FPS → 16ms 안에 그려야함

22ms
↓
60프레임이
안나옴

대략 12ms

대략 10ms

CPU

다른일..
다른일..
다른일..

GPU

렌더링..
렌더링..
렌더링..

시간의 흐름

CPU

다른일..
다른일..
다른일..

GPU

렌더링..
렌더링..
렌더링..

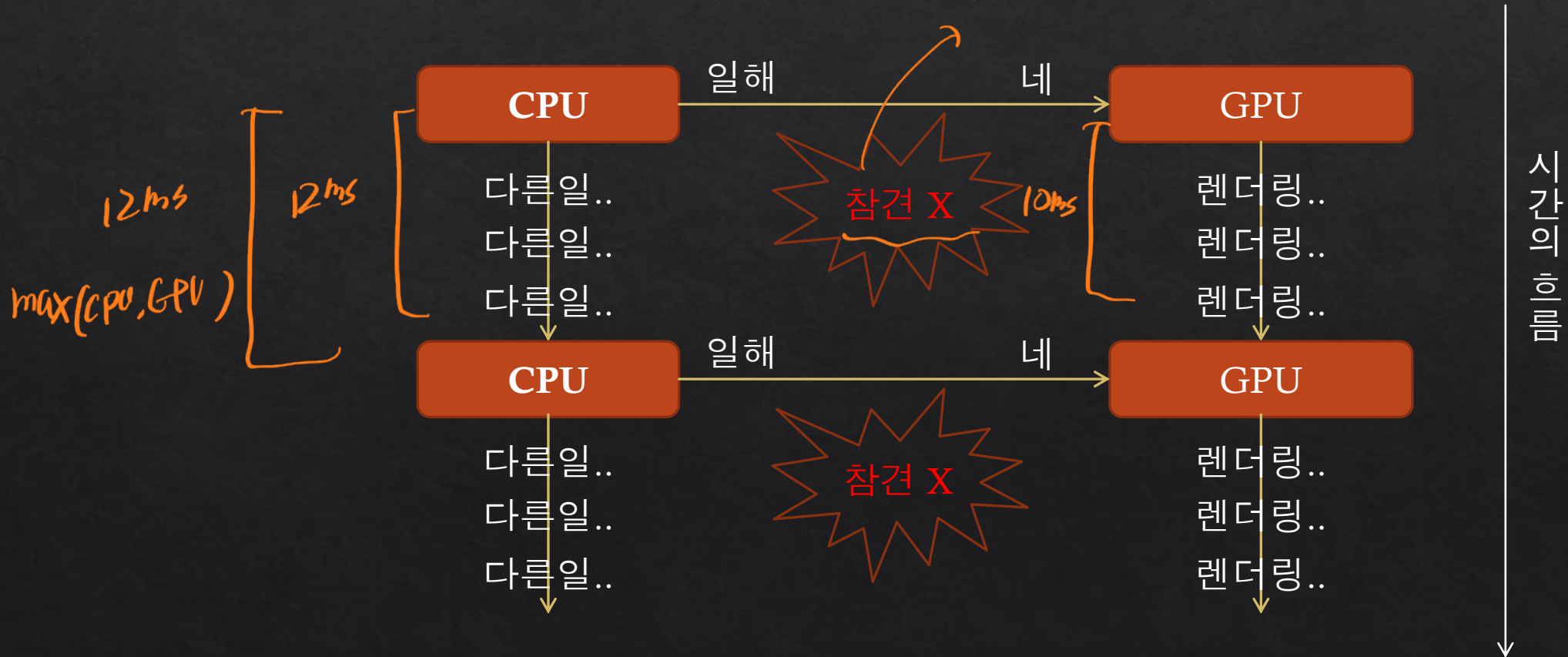
시간의 흐름

OpenGL 렌더링 구조

◇ 성능의 극대화를 위한 구조

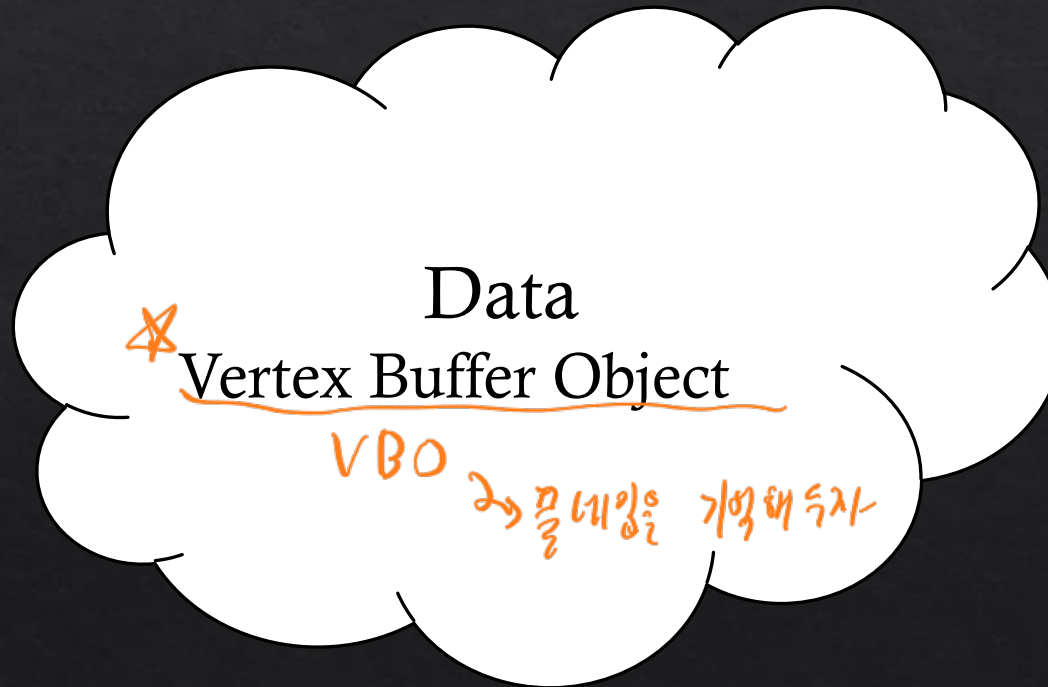
◇ CPU-GPU 병렬화에 최적화되어 있음

간섭을 하지 않기 위해 state machine 으로 작동시킨다



OpenGL 렌더링 구조

- ◆ 효율적인 렌더링을 위해 고유의 Data 형식을 가짐



OpenGL 렌더링 구조

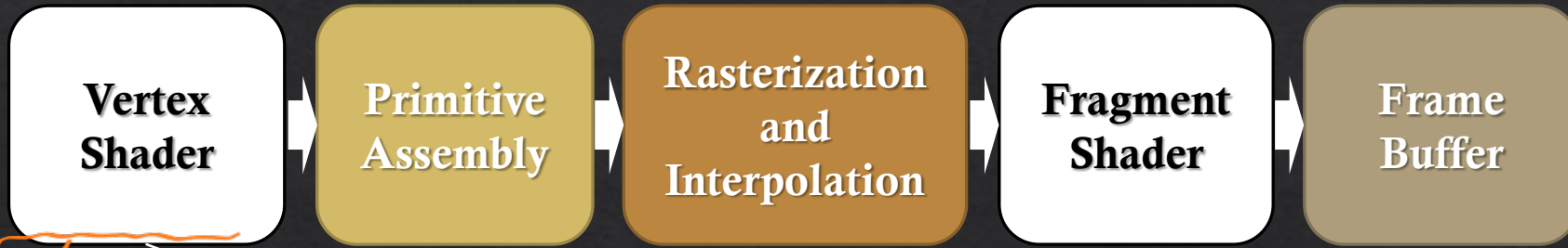
- ◆ 고유의 Data 형식을 생성하고 이를 설정하는 방법을 알아야 함



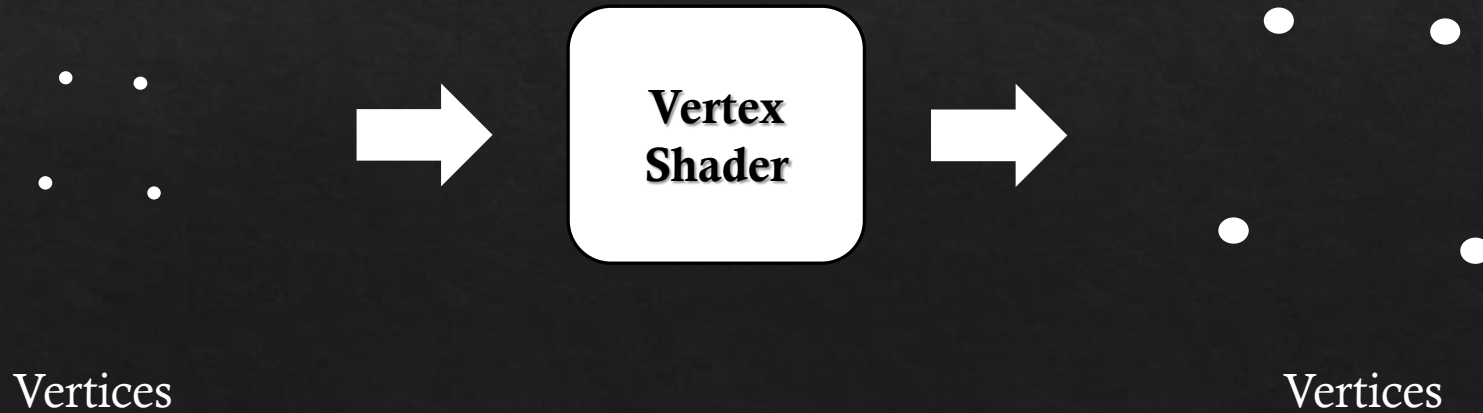
Data
Vertex Buffer Object

...

OpenGL 렌더링 구조



최초의 입력은 무조건 정점으로 시작하게 된다.



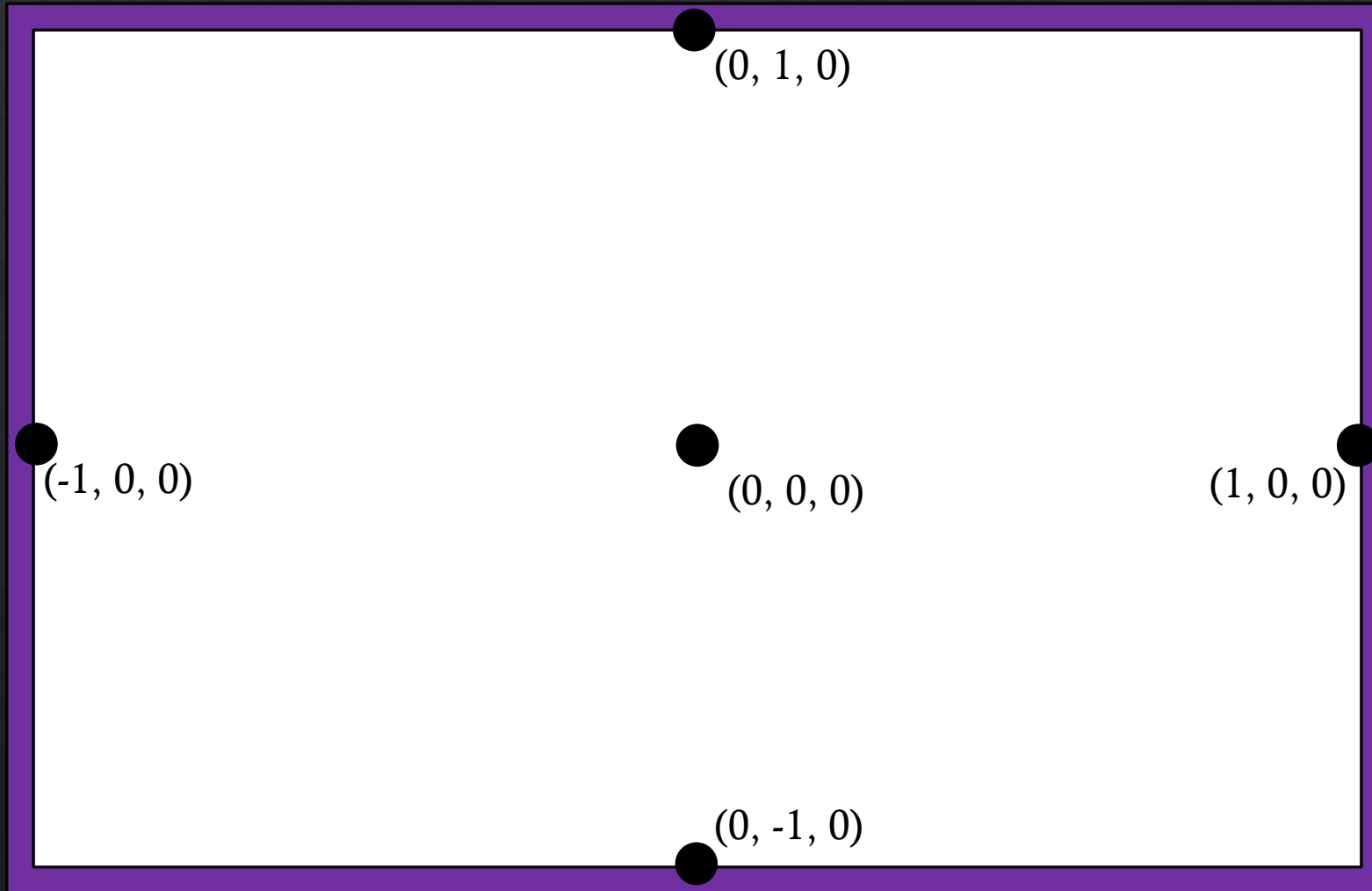
OpenGL 렌더링 구조



Vertex Shader 의 입력인 Vertices
설정 필요

OpenGL 데이터 준비

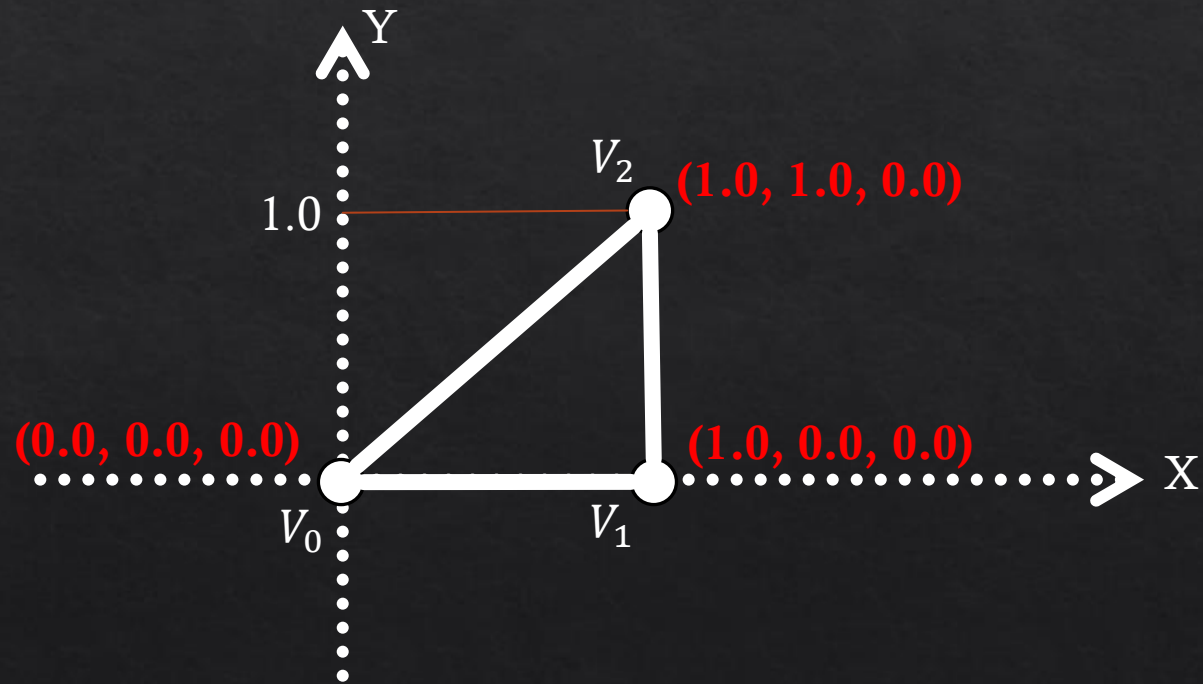
OpenGL 데이터 준비



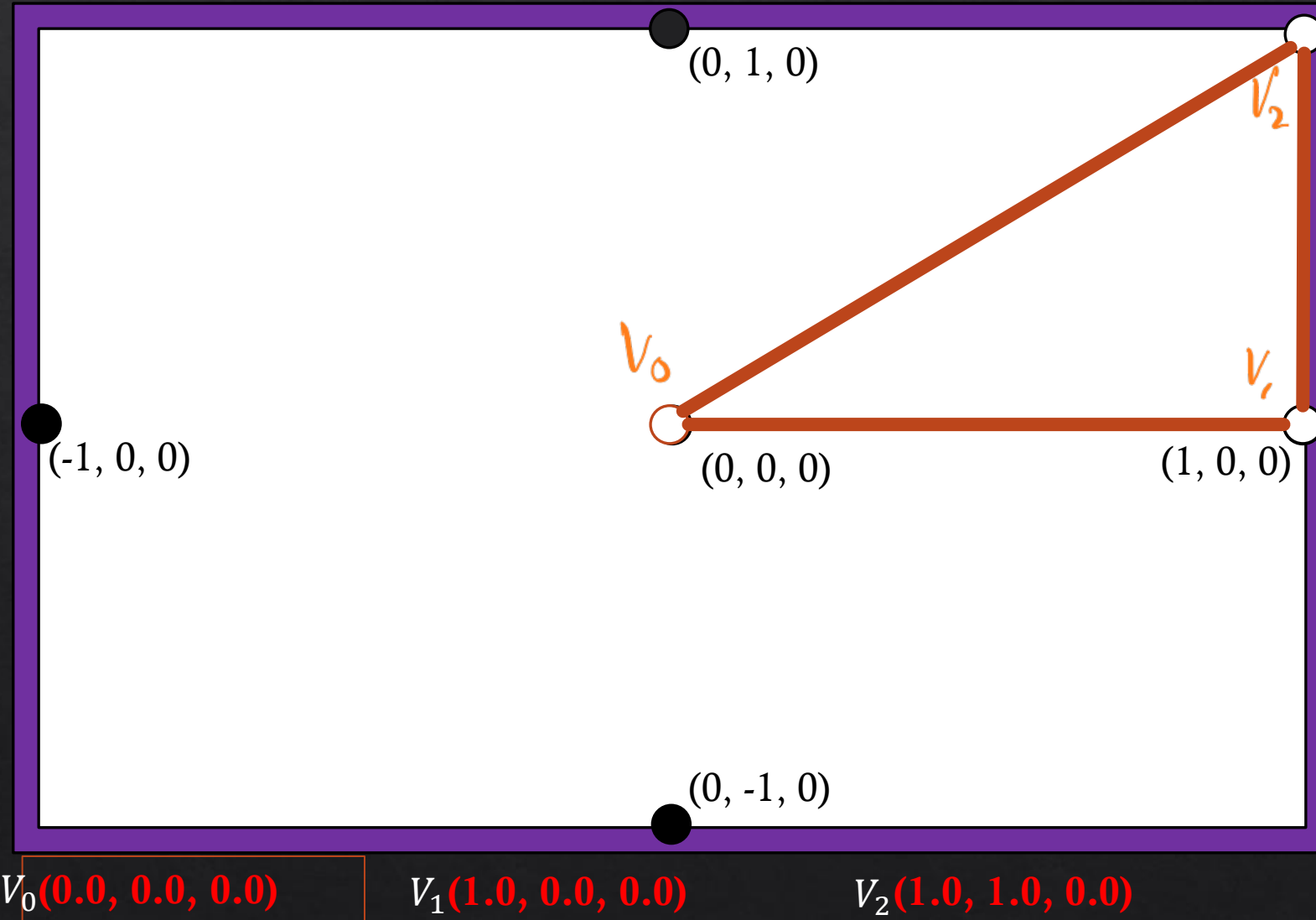
투영 매트릭스에 따라 달라질 수 있으나 일단 기본 좌표계 기반으로 진행

OpenGL 데이터 준비

Vertices !



OpenGL 데이터 준비



오른손 좌표계
↓
 V_0, V_1, V_2
↓
CCW
↓
앞면

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};
```

CPU 메모리에 저장된다.

OpenGL 데이터 준비

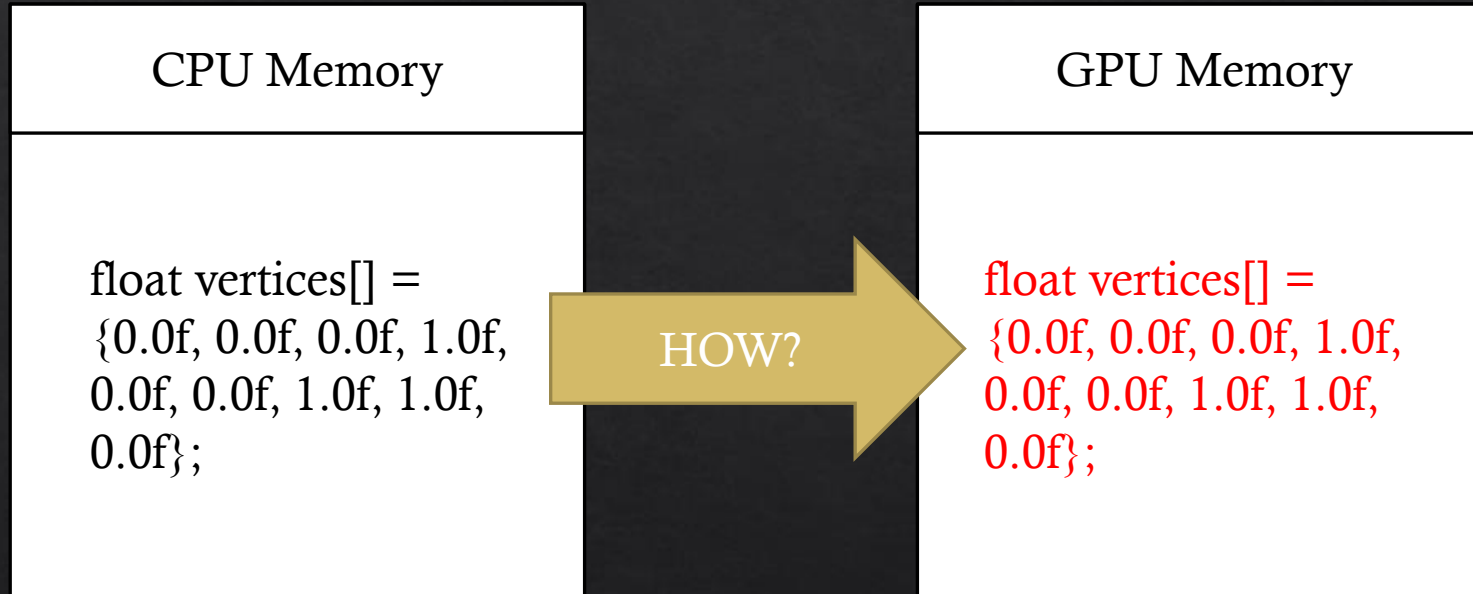
```
float vertices[] = {0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f};
```

위 array 는 어디에 저장이 되어 있을까?

위 array 를 OpenGL 에서 바로 사용 가능할까?

No, 가늠 하더라도 매우 비효율적인 것이다.

OpenGL 데이터 준비



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

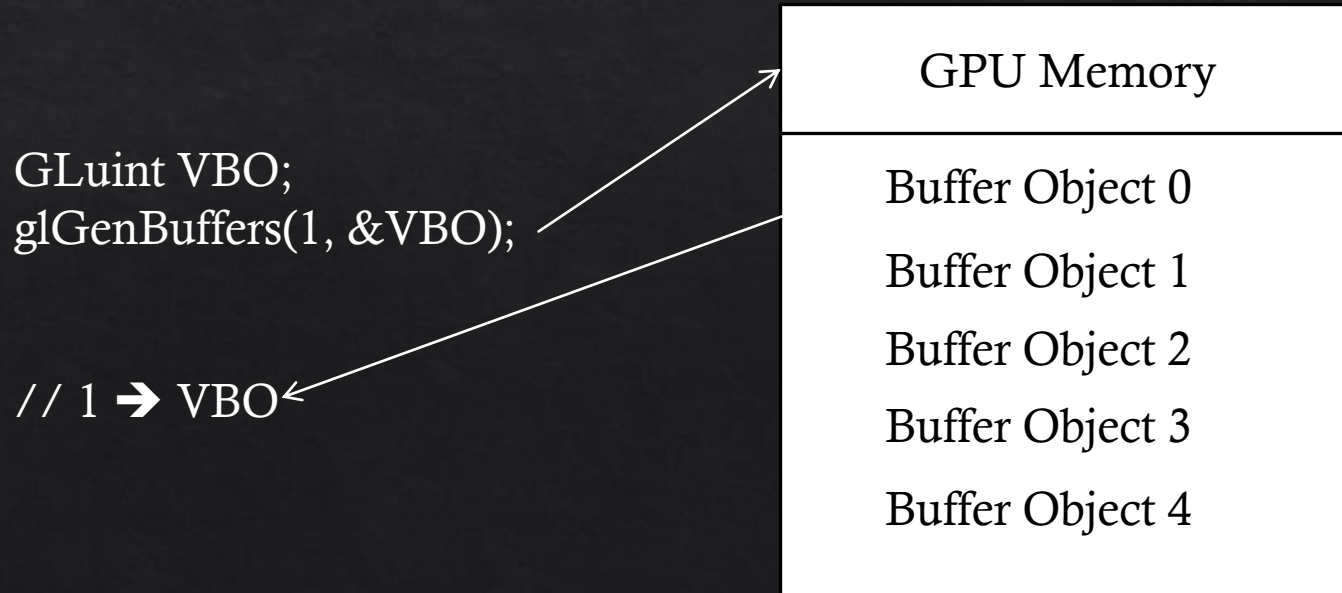
GPU Memory
<u>Buffer Object 1</u>
Buffer Object 2
Buffer Object 3
Buffer Object 4
Buffer Object 5

vertex를 담을 용도라면! Vertex Buffer Object 라 함

외부에선 buffer Object의 ID를 가지고
접근 하게 된다 (GPU의 포인터(주소값)를 못쓰기 때문)

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



OpenGL 데이터 준비

◆ glBindBuffer(GLenum target, GLuint id);

◆ 생성된 VBO 를 ID 를 사용하여 Bind 함

◆ Bind 란?

◆ 실제 OpenGL에서 작업할 대상의 형태와 용도를 구체화해 주는 것

◆ 데이터를 올리려고 하는데 그 데이터가 array 형식의 buffer를 가진다면 GL_ARRAY_BUFFER를 사용

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

```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

GL_ARRAY_BUFFER

VBO

OpenGL 데이터 준비

◆ `glBufferData(GLenum target, GLsizeiptr size, const GLvoid *data, GLenum usage);`

◆ Bind된 VBO 에 데이터를 할당

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

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

↳ 한 번 초기화 되면 2회 이상 데이터를 바꿀 수 없음

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

GL_ARRAY_BUFFER

VBO

GPU Memory

Buffer Object 0

Buffer Object 1

Buffer Object 2

Buffer Object 3

...

CPU에서 GPU 메모리로 데이터를 올려주게 된다. (시각이 오래 걸림)

여유 보일 때 조금씩 로딩하는 기법들을 사용한다.

dynamic 의 경우 레이어를 변경할 수 있다.

OpenGL 데이터 준비



한번 데이터를 불러오려면 ID만 가지고 있으면 계속 사용할 수 있다.

OpenGL 데이터 사용

OpenGL 데이터 사용

◇ 일단 BIND

- ◇ OpenGL 데이터 생성 시 이미 Bind 했으나
 - ◇ 중간에 다른 오브젝트가 BIND 되었을 가능성
- ◇ OpenGL 은 종류 당 (GL_ARRAY_BUFFER 같은) 하나의 오브젝트 만 Bind 허용

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

glEnableVertexAttribArray → 이 함수에 대한 설명은 다음 시간에..

GL_ARRAY_BUFFER

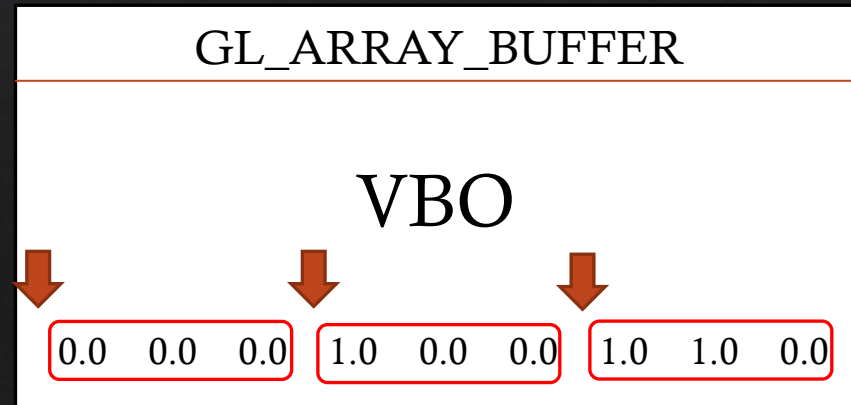
VBO

0.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0

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

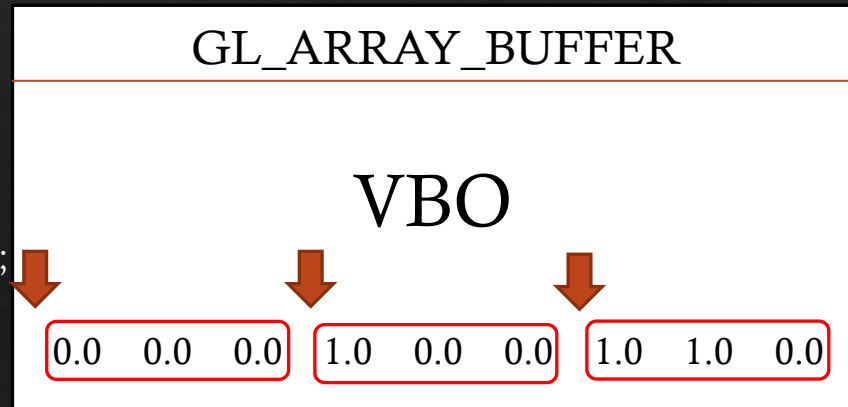
```
glEnableVertexAttribArray(0);  
glBindBuffer(GL_ARRAY_BUFFER, VBO);  
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
```



OpenGL 데이터 사용

- ◇ `glDrawArrays(GLenum mode, GLint first, GLsizei count);`
 - ◇ 어떠한 Primitive로 구성할 것인지 선택
 - ◇ Vertex 몇 개를 그릴 것인지 입력
 - ◇ 이 함수 호출 즉시 GPU가 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

