

셰이더 프로그래밍

Lecture 4

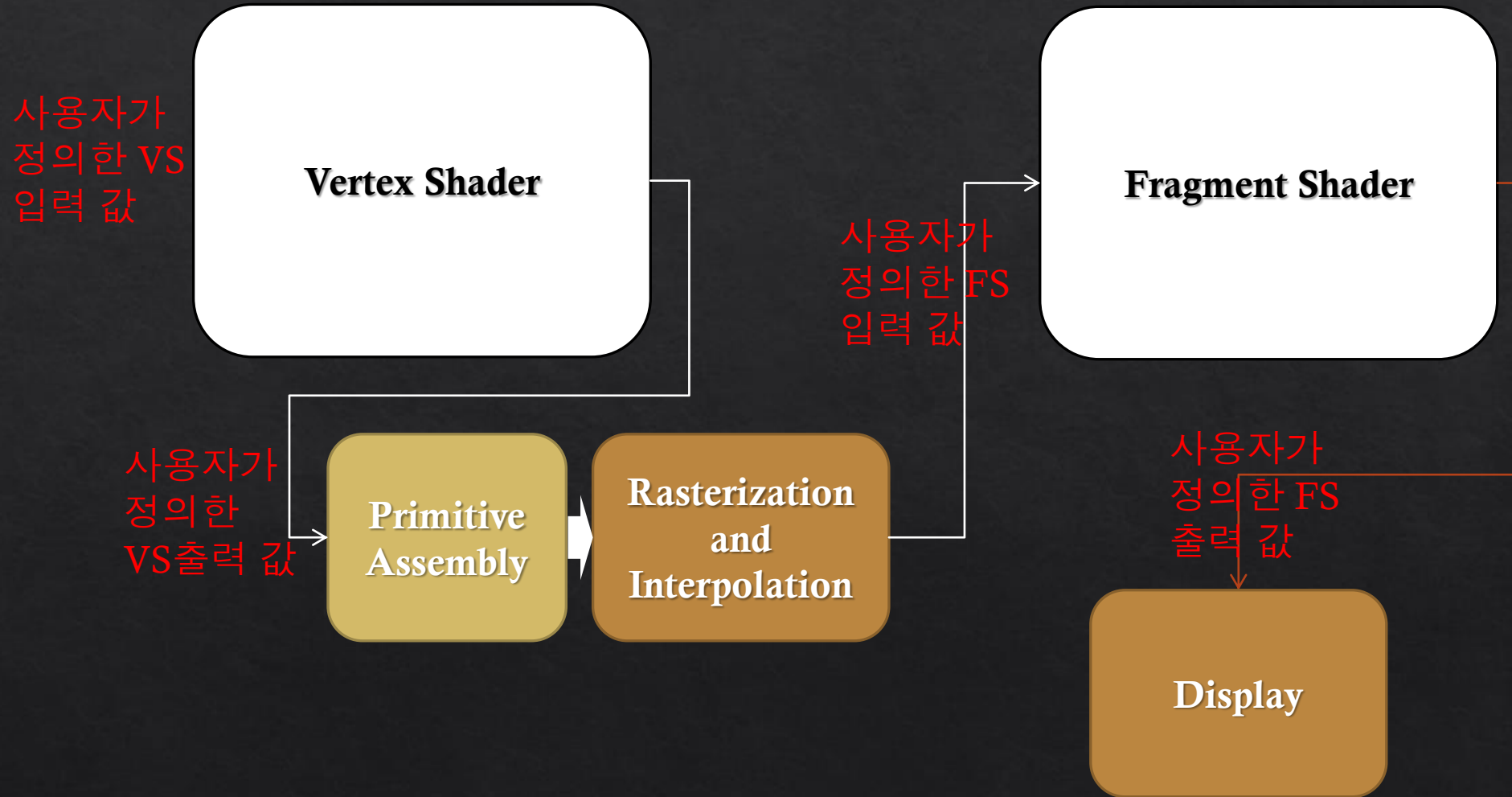
이택희

개요

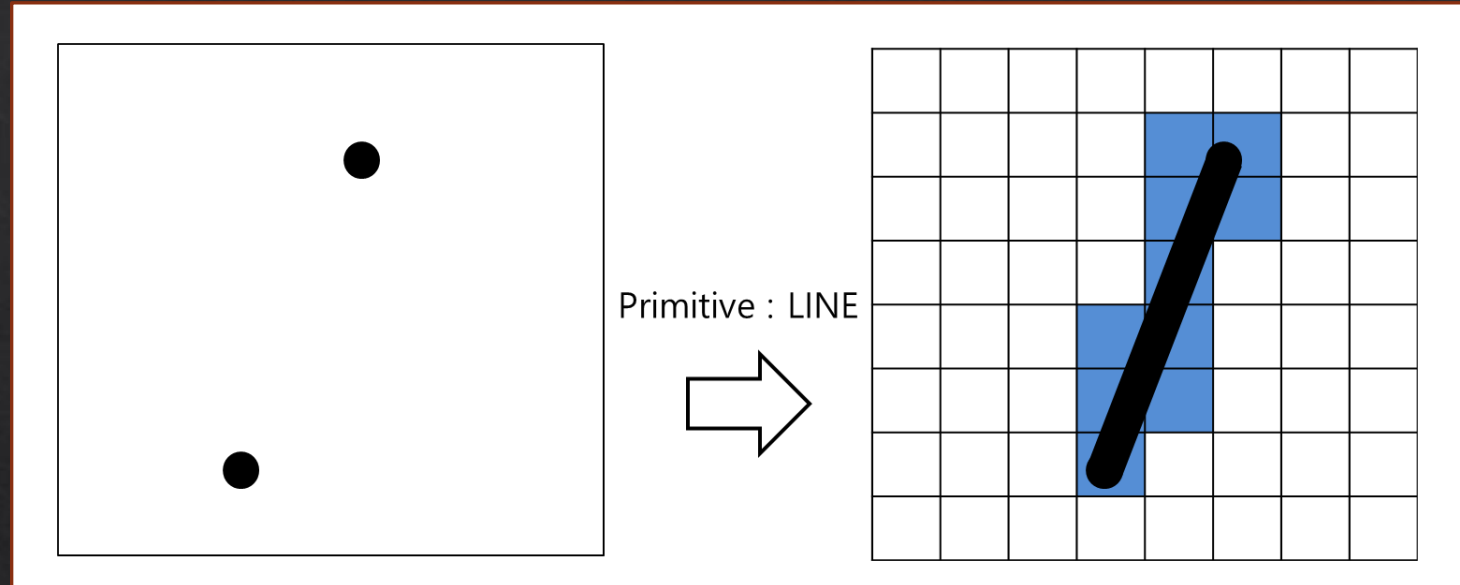
- ◇ 프래그먼트 셰이더
- ◇ Storage Qualifier
- ◇ 버텍스 셰이더 입력 데이터 패킹
- ◇ 실습

프래그먼트 셰이더

Shader 입출력



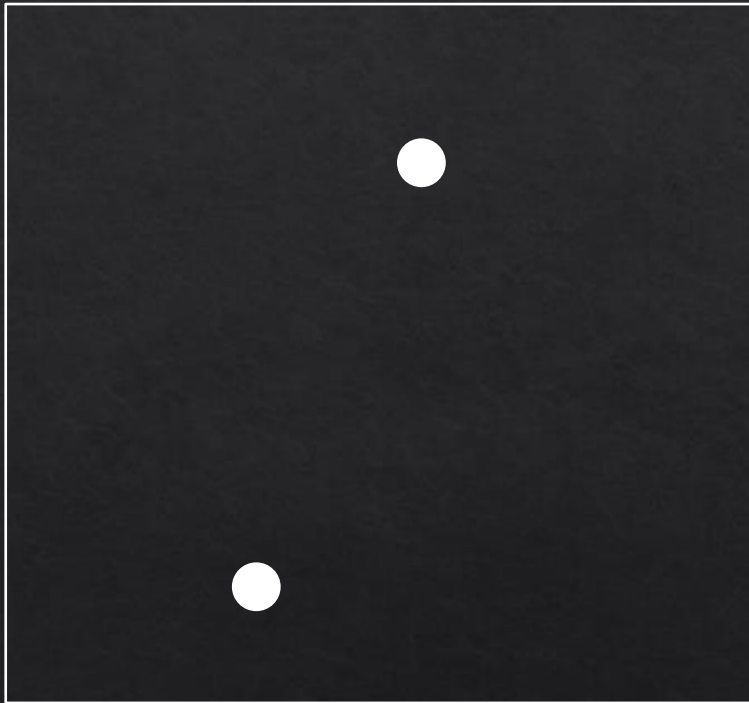
프래그먼트 셰이더



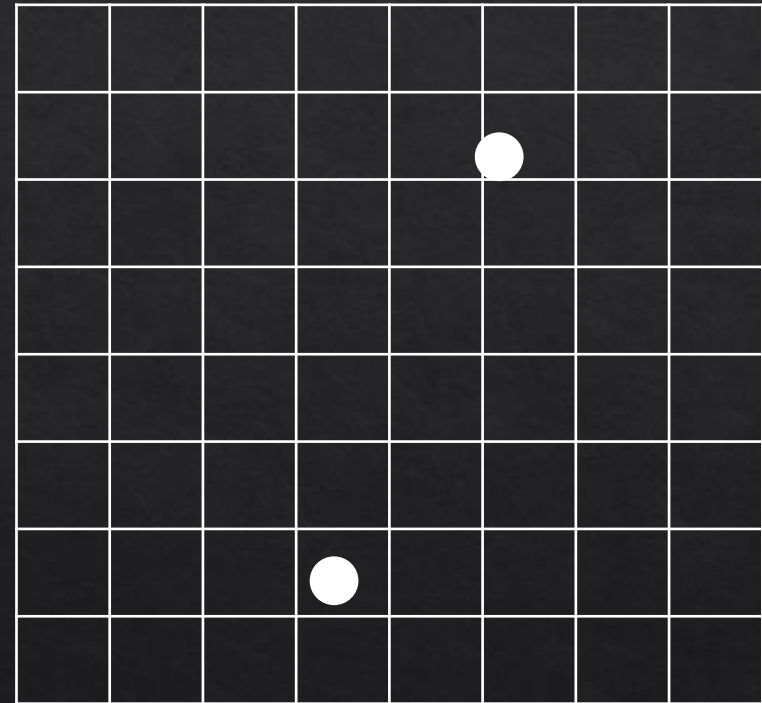
버텍스 셰이더의 출력과 프래그먼트 셰이더 입력의 타입은 서로 같아야 함

vertex → primitive → fragments 과정에서 따라서 **보간**이 일어남

프래그먼트 셰이더

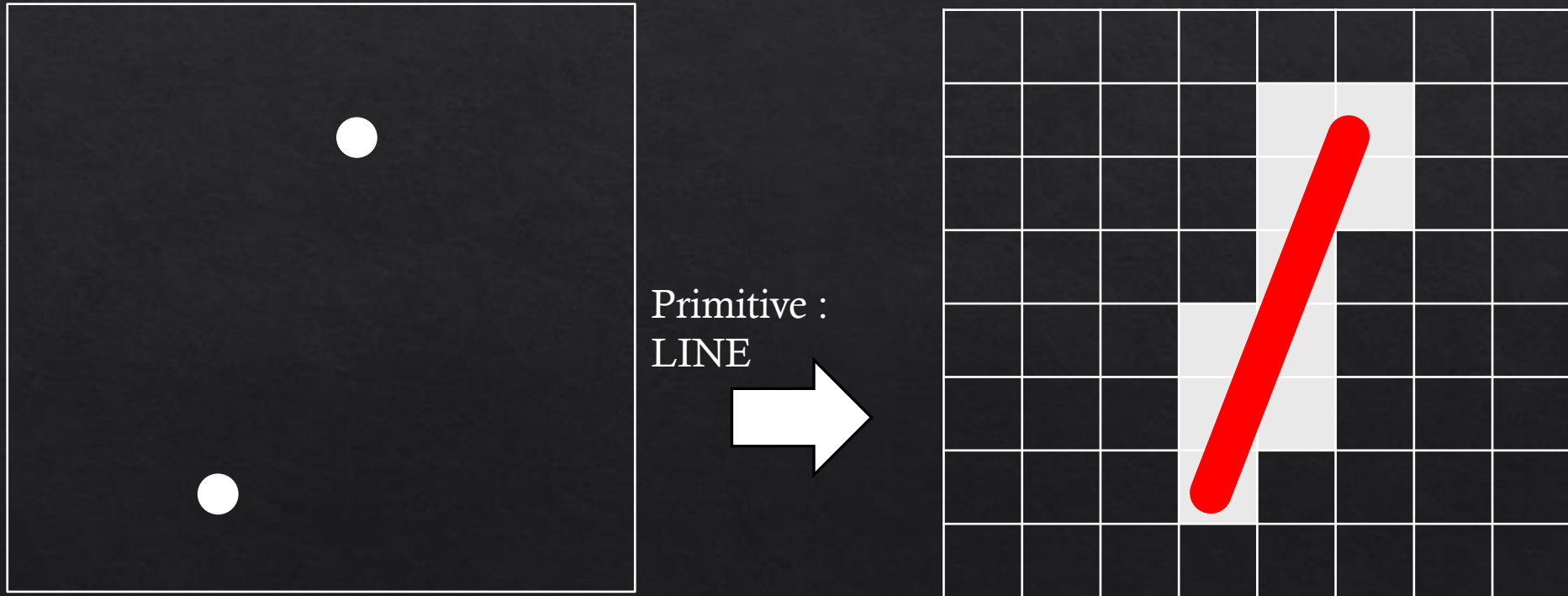


Primitive :
LINE



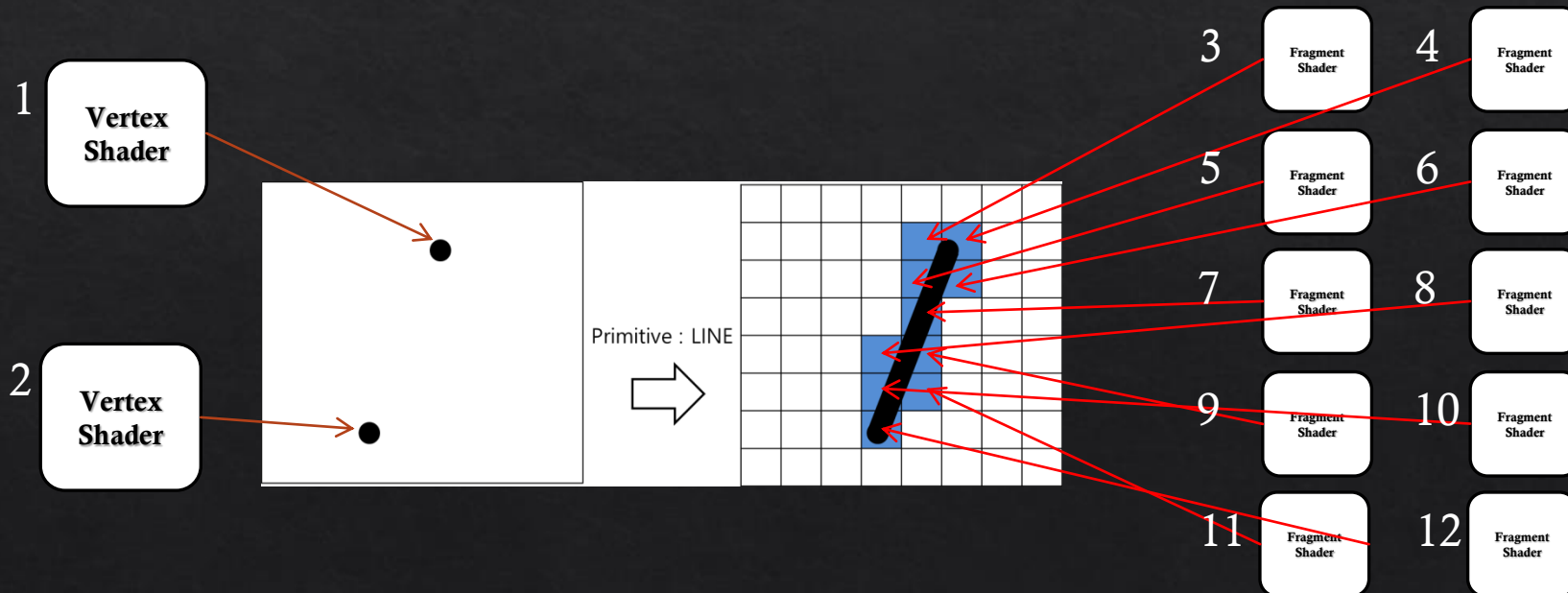
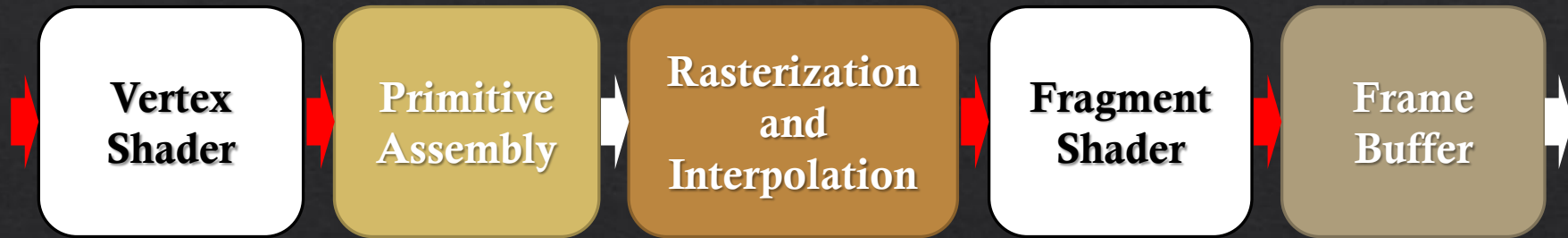
프래그먼트 셰이더

- ◇ 버텍스는 두 개이지만 보간된 프래그먼트는 10개

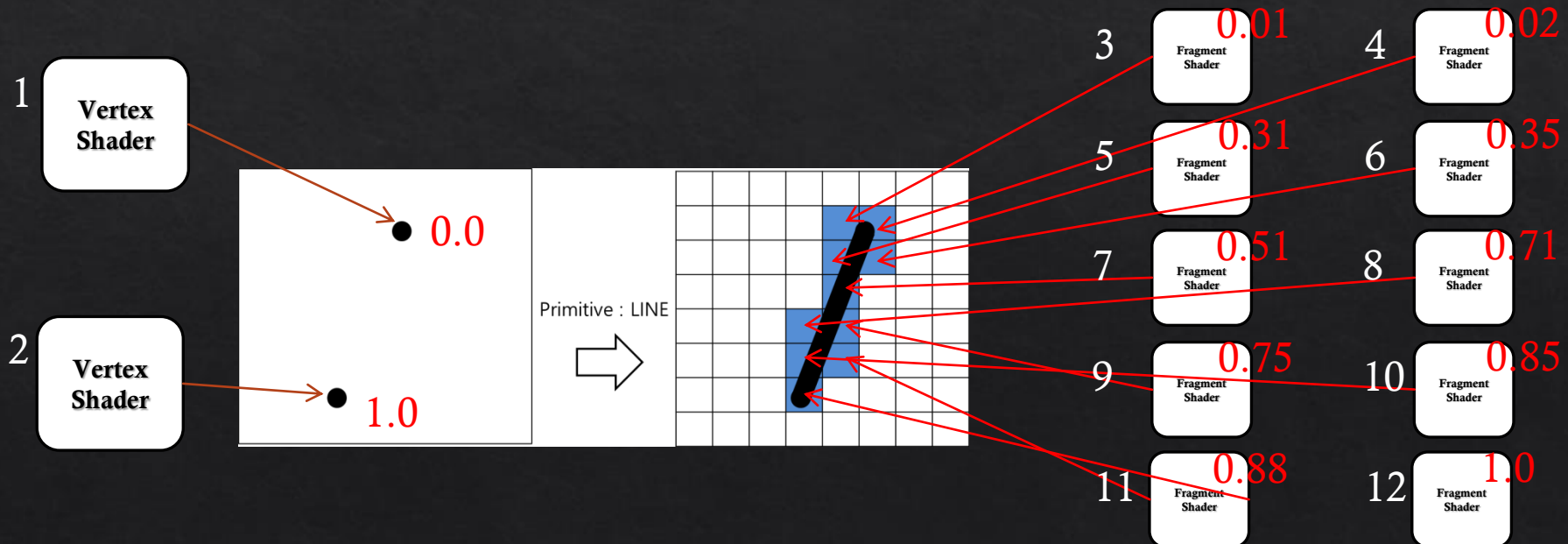
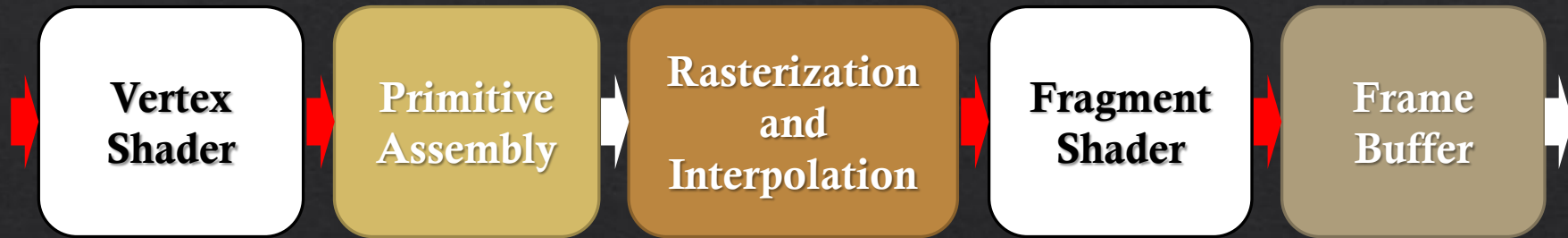


총 10번의 프래그먼트 셰이더가 동작함 !

프래그먼트 셰이더



프래그먼트 셰이더



Storage Qualifier

Storage Qualifier



각 Stage 의 입출력을 지정해야 함

Storage Qualifier

Stage 자체의 입출력
값에 대해 쓰이는
Qualifier



Storage Qualifier	Meaning
< none: default >	local read/write memory, or an input parameter to a function
const	a variable whose value cannot be changed
in	linkage into a shader from a previous stage, variable is copied in
out	linkage out of a shader to a subsequent stage, variable is copied out
attribute	compatibility profile only and vertex language only; same as in when in a vertex shader
uniform	value does not change across the primitive being processed, uniforms form the linkage between a shader, OpenGL, and the application
varying	compatibility profile only and vertex and fragment languages only; same as out when in a vertex shader and same as in when in a fragment shader

주의

Stage 내부에서
쓰이는 Modifier



Access Modifier	Description
in	Value copied into a function (default if not specified)
const in	Read-only value copied into a function
out	Value copied out of a function (undefined upon entrance into the function)
inout	Value copied into and out of a function

Storage Qualifier(in/out)

◇ 각 Stage 의 입출력을 지정

◇ 그래픽스 파이프라인 외부로의 입출력

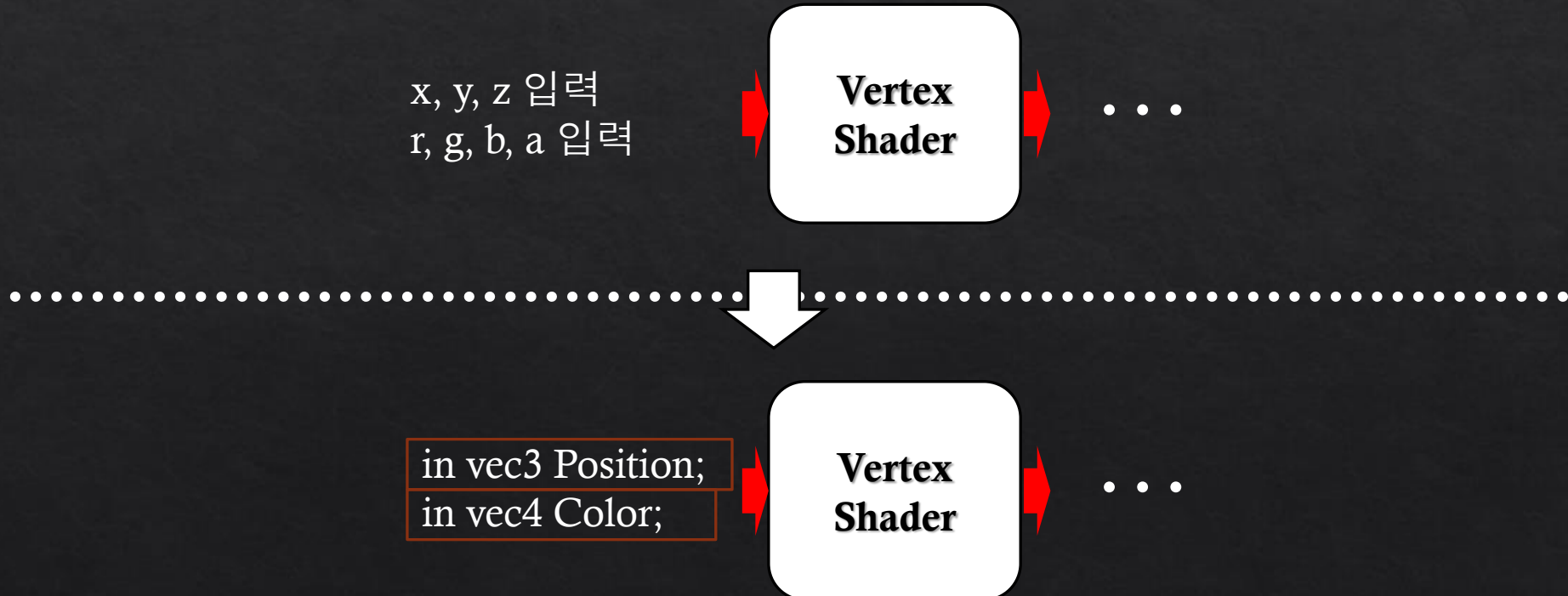


◇ 그래픽스 파이프라인 Stage 간의 입출력



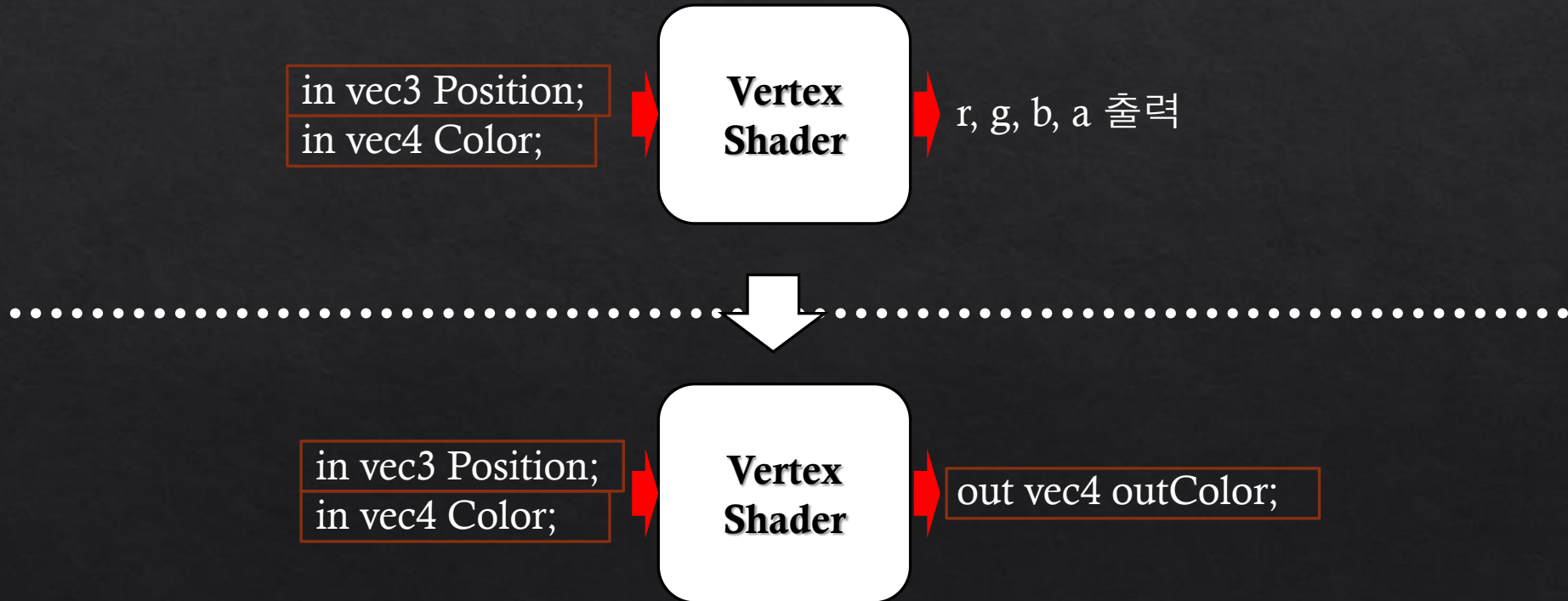
Storage Qualifier(in/out)

- 그래픽스 파이프라인 외부로부터의 입력
 - ◇ 버텍스 셰이더의 입력 값



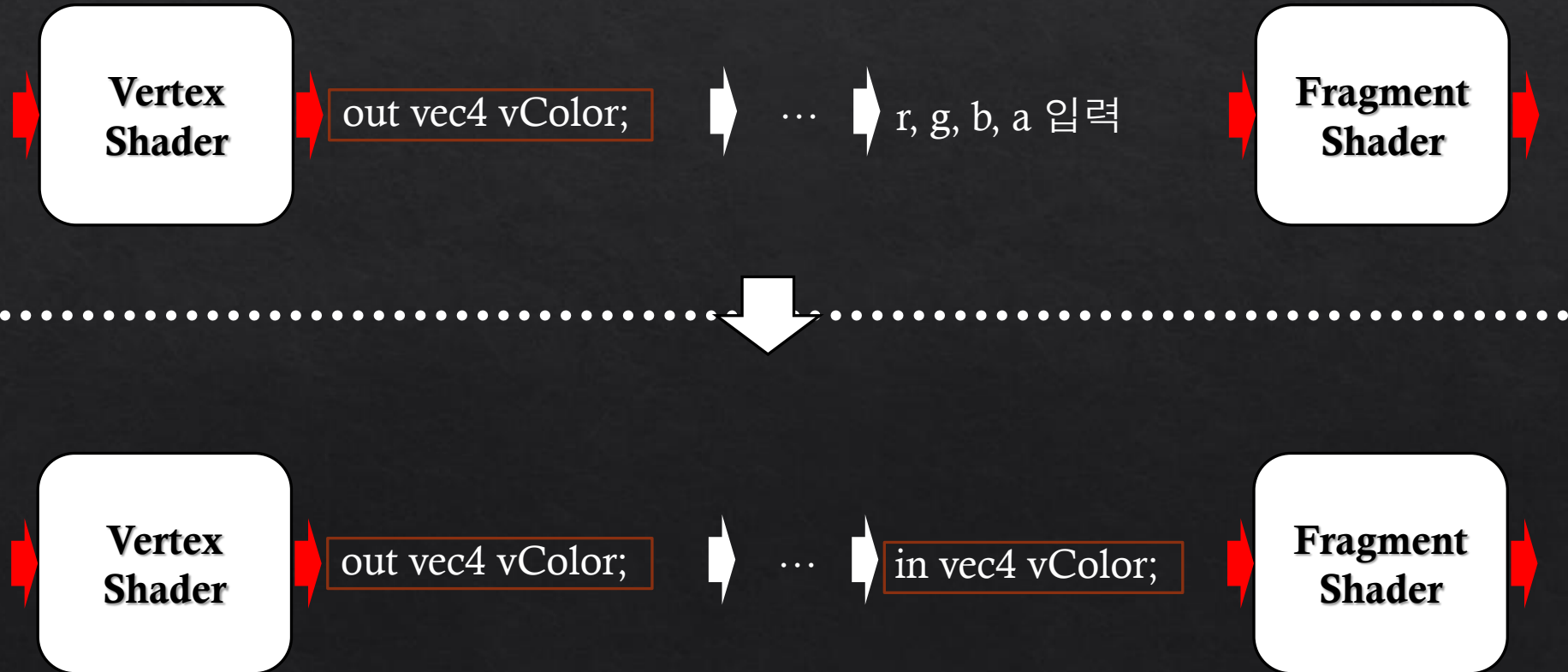
Storage Qualifier(in/out)

- 그래픽스 파이프라인 다음 Stage로 출력
 - ◇ 버텍스 셰이더의 출력 값



Storage Qualifier(in/out)

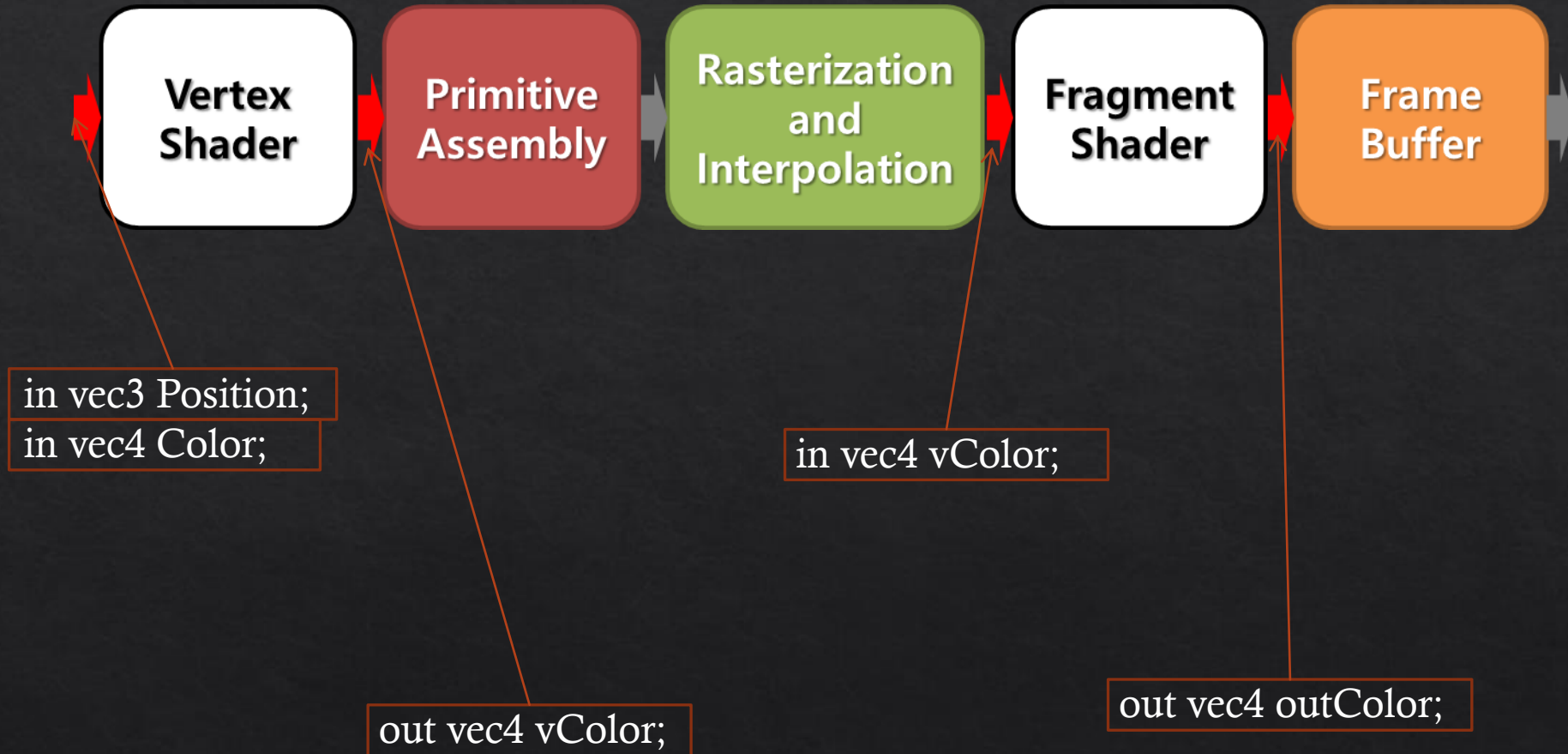
- 그래픽스 파이프라인 이전 Stage 로부터의 입력
 - ◇ 버텍스 셰이더의 출력값 == 프래그먼트 셰이더의 입력값



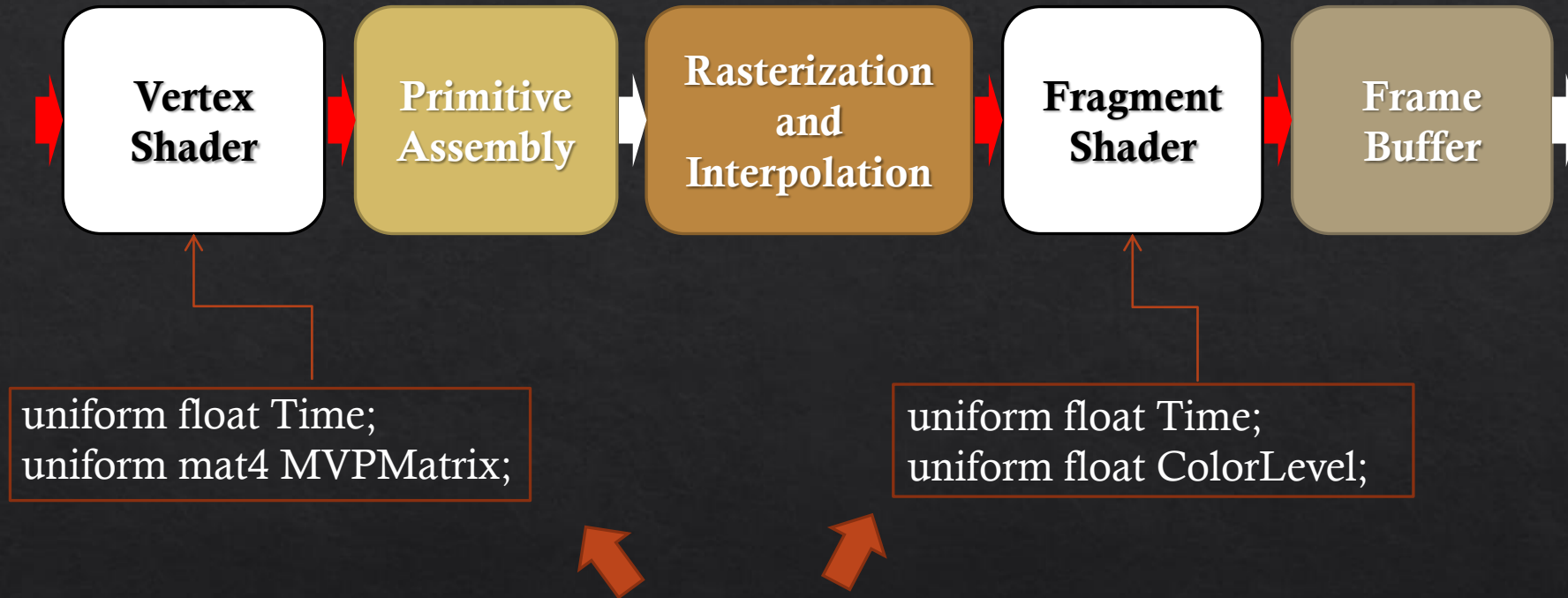
Storage Qualifier(in/out)



Storage Qualifier(in/out)



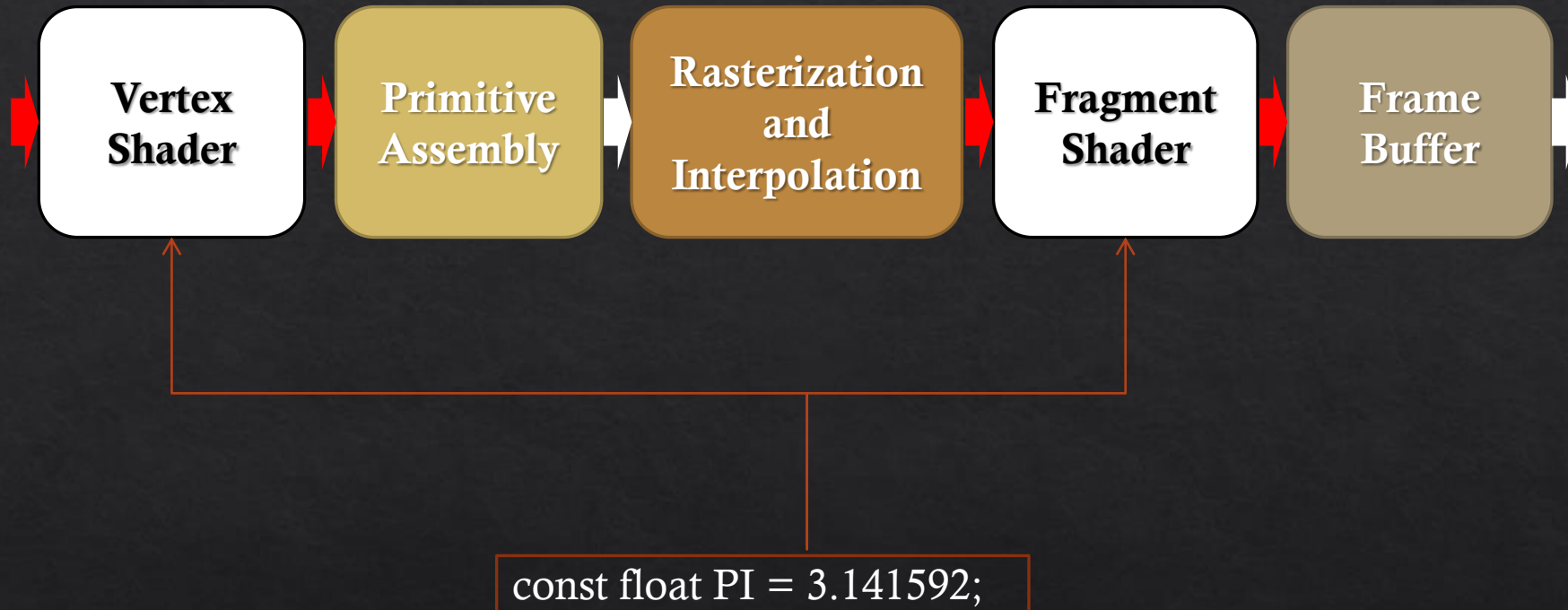
Storage Qualifier(uniform)



외부에서 지정한 값을 쓰고 싶을 경우 uniform 사용

`uniform float Time;` 과 같이 stage 간에 공유 가능함

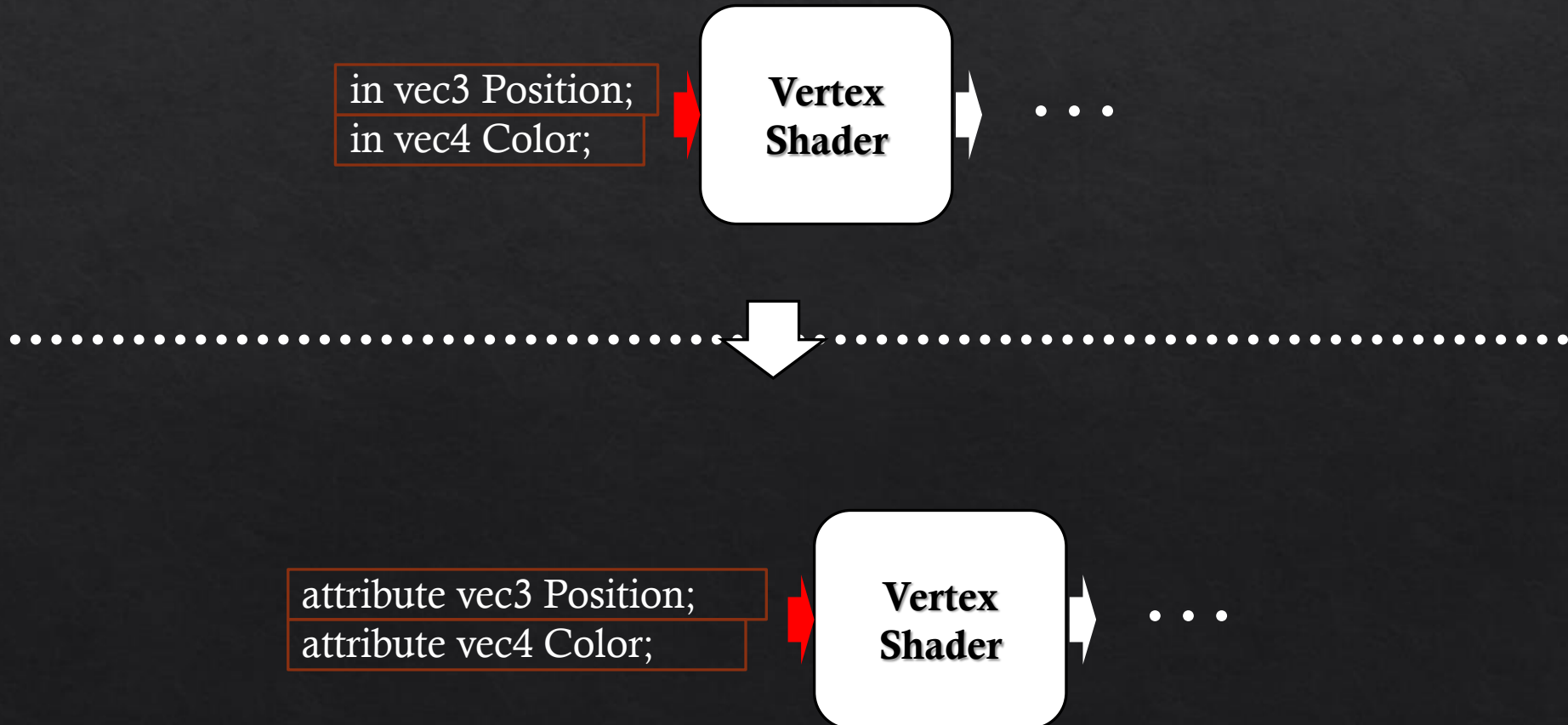
Storage Qualifier(const)



내부에서 지정하고 변하지 않을 경우 const 사용
각 셰이더 별로 선언 해야 함

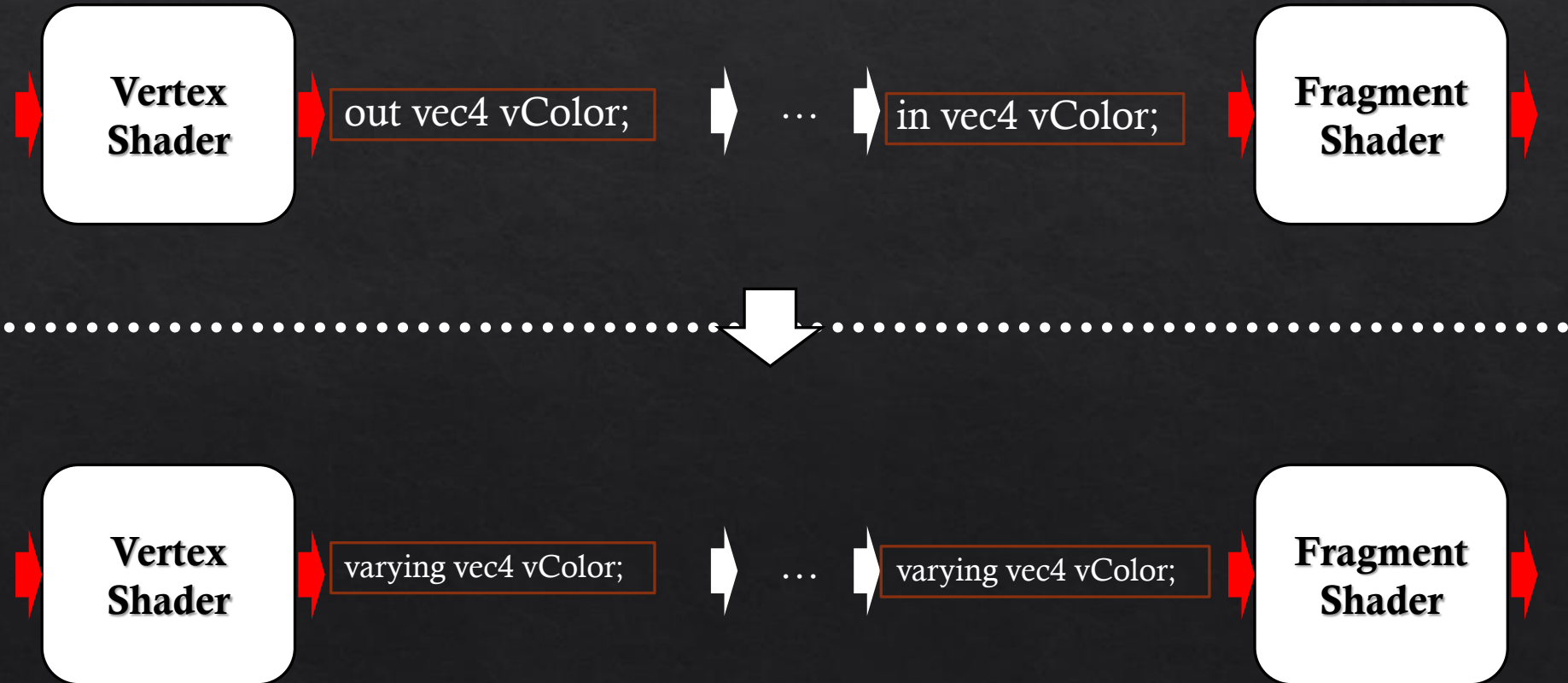
Storage Qualifier 사용

- 버텍스 셰이더의 경우 입력값 Qualifier 를 Attribute 바꾸어도 무방



Storage Qualifier 사용

- 버텍스 셰이더의 out 과 프래그먼트 셰이더의 in 의 경우 varying 으로 바꾸어도 무방



실습

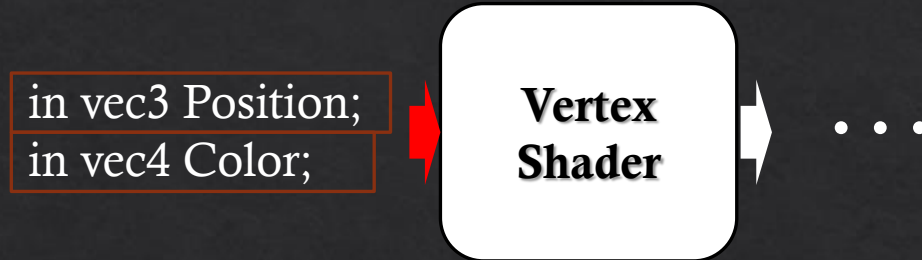
- ◆ 파티클 별 랜덤 컬러 부여
- ◆ 파티클이 서서히 사라지도록

실습

- ◇ 화면에 커다란 사각형 그리기 (색은 흰색으로)
 - ◇ $\text{Min}(-0.5, -0.5), \text{Max}(0.5, 0.5)$
- ◇ Varying 값을 사용하여 Interpolation 과정 이해
 - ◇ 버텍스 셰이더 출력 값 변화시켜 보기
 - ◇ 내부가 채워진 원 그려보기
 - ◇ 내부가 빈 원 그려보기
 - ◇ 여러 개의 동심원 그려보기
 - ◇ 특정 지점에 원 그려보기
 - ◇ 레이더 구현해 보기

버텍스 셰이더 입력 데이터 패킹

버텍스 셰이더 입력 데이터 패킹



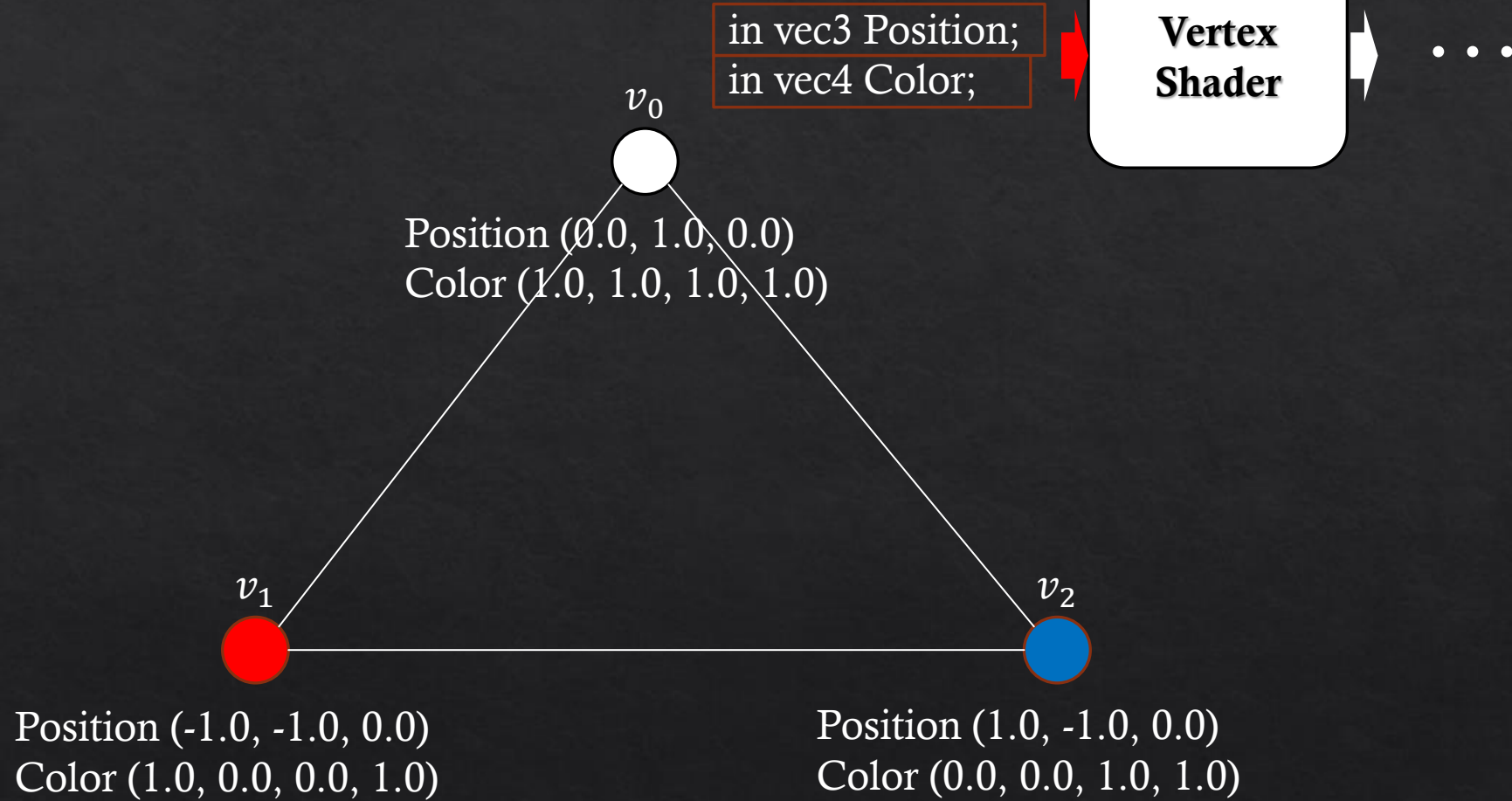
```
int positionAttribID glGetAttribLocation(gShaderProgram, "Position");  
int colorAttribID glGetAttribLocation(gShaderProgram, "Color");
```

```
glEnableVertexAttribArray(positionAttribID);  
glEnableVertexAttribArray(colorAttribID);
```

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

```
glVertexAttribPointer(positionAttribID, ?, GL_FLOAT, GL_FALSE, ?, 0);  
glVertexAttribPointer(colorAttribID, ?, GL_FLOAT, GL_FALSE, ?, 0);
```

버텍스 셰이더 입력 데이터 패킹



버텍스 셰이더 입력 데이터 패킹



1. 각각 Array 생성

```
float Position[9] = {0.0, 1.0, 0.0, -1.0, -1.0, 0.0, 1.0, -1.0, 0.0};
```

```
float Color[12] = {1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0};
```

2. 합친 Array 생성 - 1

```
float PositionColor[21] = {0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, -1.0, -1.0, 0.0, 1.0, 0.0, 0.0, 1.0,  
1.0, -1.0, 0.0, 0.0, 0.0, 1.0, 1.0};
```

3. 합친 Array 생성 - 2

```
float PositionColor[21] = {0.0, 1.0, 0.0, -1.0, -1.0, 0.0, 1.0, -1.0, 0.0, 1.0, 1.0, 1.0, 1.0,  
1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0};
```


버텍스 셰이더 입력 데이터 패킹

1. 각각 Array 생성

```
float Position[9] = {0.0, 1.0, 0.0, -1.0, -1.0, 0.0, 1.0, -1.0, 0.0};
```

```
float Color[12] = {1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0};
```

버텍스 셰이더 입력 데이터 패킹

```
//OpenGL 데이터 생성
```

```
GLuint VBO;
```

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

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

```
glBufferData(GL_ARRAY_BUFFER, sizeof(Position), Position, GL_STATIC_DRAW);
```

```
GLuint VBO1;
```

```
glGenBuffers(1, &VBO1);
```

```
glBindBuffer(GL_ARRAY_BUFFER, VBO1);
```

```
glBufferData(GL_ARRAY_BUFFER, sizeof(Color), Color, GL_STATIC_DRAW);
```

```
//사용시
```

```
int positionAttribID glGetAttribLocation(gShaderProgram, "Position");
```

```
int colorAttribID glGetAttribLocation(gShaderProgram, "Color");
```

```
glEnableVertexAttribArray(positionAttribID);
```

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

```
glVertexAttribPointer(positionAttribID, 3, GL_FLOAT, GL_FALSE, 0, 0);
```

```
glEnableVertexAttribArray(colorAttribID);
```

```
glBindBuffer(GL_ARRAY_BUFFER, VBO1);
```

```
glVertexAttribPointer(colorAttribID, 4, GL_FLOAT, GL_FALSE, 0, 0);
```

```
glDrawArrays(GL_TRIANGLES, 0, 3);
```

```
in vec3 Position;
```

```
in vec4 Color;
```

**Vertex
Shader**

...

버텍스 셰이더 입력 데이터 패킹

2. 합친 Array 생성 - 1

```
float PositionColor[21] = {0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, -1.0, -1.0, 0.0, 1.0,  
0.0, 0.0, 1.0, 1.0, -1.0, 0.0, 0.0, 0.0, 1.0, 1.0};
```

버텍스 셰이더 입력 데이터 패킹

```
//OpenGL 데이터 생성
GLuint VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(PositionColor), PositionColor, GL_STATIC_DRAW);

//사용시
int positionAttribID glGetAttribLocation(gShaderProgram, "Position");
int colorAttribID glGetAttribLocation(gShaderProgram, "Color");

glEnableVertexAttribArray(positionAttribID);
glEnableVertexAttribArray(colorAttribID);

glBindBuffer(GL_ARRAY_BUFFER, VBO);
glVertexAttribPointer(positionAttribID, 3, GL_FLOAT, GL_FALSE, 7*sizeof(float), 0);
glVertexAttribPointer(colorAttribID, 4, GL_FLOAT, GL_FALSE, 7*sizeof(float), (GLvoid*)(3*sizeof(float)));

glDrawArrays(GL_TRIANGLES, 0, 3);
```

in vec3 Position;
in vec4 Color;



**Vertex
Shader**



...

버텍스 셰이더 입력 데이터 패킹

2. 합친 Array 생성 - 2

```
float PositionColor[21] = {0.0, 1.0, 0.0, -1.0, -1.0, 0.0, 1.0, -1.0, 0.0, 1.0, 1.0,  
1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0};
```


버텍스 셰이더 입력 데이터 패킹

```
//OpenGL 데이터 생성
GLuint VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(PositionColor), PositionColor, GL_STATIC_DRAW);
```

```
//사용시
int positionAttribID glGetAttribLocation(gShaderProgram, "Position");
int colorAttribID glGetAttribLocation(gShaderProgram, "Color");
```

```
glEnableVertexAttribArray(positionAttribID);
glEnableVertexAttribArray(colorAttribID);
```

```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glVertexAttribPointer(positionAttribID, 3, GL_FLOAT, GL_FALSE, 0, 0);
glVertexAttribPointer(colorAttribID, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)(9*sizeof(float)));
```

```
glDrawArrays(GL_TRIANGLES, 0, 3);
```

in vec3 Position;
in vec4 Color;



**Vertex
Shader**



...

실습