셰이더프로그래밍

Lecture 3

이택희

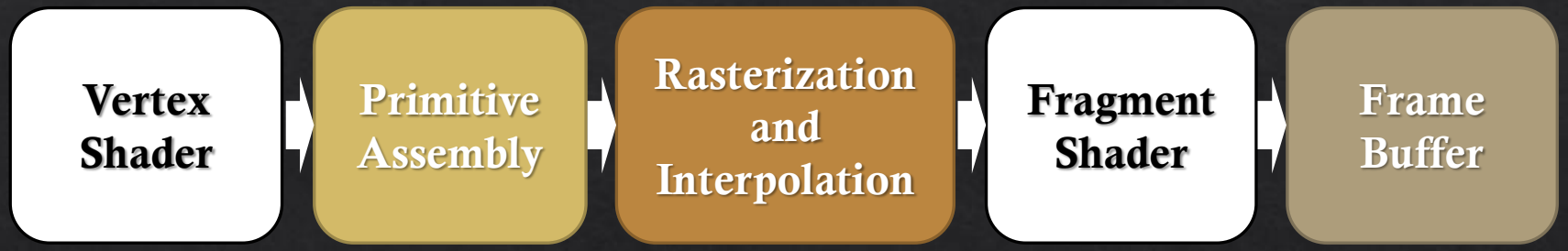
개요

- OpenGL Shading Language
- Shader 입출력
- Shader 컴파일
- Vertex Shader 입력
- Shader 외부 입력
- 실습

OpenGL Shading Language

OpenGL Shading Language

Fixed 파이프라인이었던 구조에서 프로그래밍 가능한 몇몇 단계로 대체되면서 바뀌게 되었다. 오픈지엘용이면 오픈지엘 쉐이딩 랭귀지 GLSL, 다이렉트는 HLSL 이라고 하이 레벨 셰이딩 랭귀지라고 한다.



프로그래밍 가능한 Shader!

OpenGL Shading Language

하나의 Vertex 단위로 처리되는 stage

OpenGL Shading Language

하나의 Fragment 단위로 처리되는 stage

OpenGL Shading Language

Tessellation Shader

석사 수준?

Geometry Shader

박사 수준?

본 강의에서는 다루지 않음

OpenGL Shading Language

- C 프로그래밍 형식

```
#version 450 core

void
main()
{
	// Your code goes here
}
```

c 프로그래밍에서 main 함수로 진입하는 것처럼 쉐이더에서도 main 함수로 진입을 한다.
쉐이더에서도 버텍스 정보가 들어오면 최초로 실행되는 것은 main()이다.

OpenGL Shading Language

- Basic Types

Type	Description
`float`	IEEE 32-bit floating-point value
`double`	IEEE 64-bit floating-point value
`int`	signed two's-complement 32-bit integer value
`uint`	unsigned 32-bit integer value
`bool`	Boolean value

OpenGL Shading Language

- 변수 범위 (C 와 동일)
  - 함수 외부에서 선언된 경우 전역 변수로 인식
  - For 나 If 문 이후 등장하는 { } 내부에서 선언된 경우 내부에서만 제어 가능
  - { } 외부의 Variables 는 { } 내부에서 제어 가능

for 문이나 if 문도 사용 가능하다.
함수의 범위도 같고
이니셜라이저 하는 방법도 뒷페이지와 같다.

```
for (int i = 0; i < 10; ++i) {
    // loop body
}
```

OpenGL Shading Language

- 변수 초기화

```
int       i, numParticles = 1500;
float     force, g = -9.8;
bool      falling = true;
double    pi = 3.14159265358979323846 26LF;
```

OpenGL Shading Language

- 변수 타입 캐스팅
  - 정보를 잃게 될 가능성이 있을 경우 명확하게 표시를 해 줘야 함
    - Ex : double → float, float → int, uint → int

Type Needed	Can Be Implicitly Converted From
uint	int
float	int, uint
double	int, uint, float

그냥 변경 가능

```
float f = 10.0;
int   ten = int(f);
```

타입 캐스팅 필요

OpenGL Shading Language

- 집합체 타입

Base Type	2D vec	3D vec	4D vec	Matrix Types		
float	vec2	vec3	vec4	mat2 mat2x2 mat3x2 mat4x2	mat3 mat2x3 mat3x3 mat4x3	mat4 mat2x4 mat3x4 mat4x4
double	dvec2	dvec3	dvec4	dmat2 dmat2x2 dmat3x2 dmat4x2	dmat3 dmat2x3 dmat3x3 dmat4x3	dmat4 dmat2x4 dmat3x4 dmat4x4
int	ivec2	ivec3	ivec4	—		
uint	uvec2	uvec3	uvec4	—		
bool	bvec2	bvec3	bvec4	—		

OpenGL Shading Language

- 집합체 타입 선언

```
vec3 velocity = vec3(0.0, 2.0, 3.0);
```

```
vec3 white = vec3(1.0);  // white = (1.0, 1.0, 1.0)
```

$$m = \mathrm{mat3}(4.0) = \begin{pmatrix} 4.0 & 0.0 & 0.0 \\ 0.0 & 4.0 & 0.0 \\ 0.0 & 0.0 & 4.0 \end{pmatrix}$$

OpenGL Shading Language

- 집합체 타입 선언

```
mat3 M = mat3(1.0, 2.0, 3.0,
              4.0, 5.0, 6.0,
              7.0, 8.0, 9.0);
```

```
vec3 column1 = vec3(1.0, 2.0, 3.0);
vec3 column2 = vec3(4.0, 5.0, 6.0);
vec3 column3 = vec3(7.0, 8.0, 9.0);

mat3 M = mat3(column1, column2, column3);
```

$$\begin{pmatrix} 1.0 & 4.0 & 7.0 \\ 2.0 & 5.0 & 8.0 \\ 3.0 & 6.0 & 9.0 \end{pmatrix}$$

```
vec2 column1 = vec2(1.0, 2.0);
vec2 column2 = vec2(4.0, 5.0);
vec2 column3 = vec2(7.0, 8.0);

mat3 M = mat3(column1, 3.0,

column2, 6.0,
column3, 9.0);
```

Column major matrices

OpenGL Shading Language

- 집합체 캐스팅

```
vec3 velocity = vec3(0.0, 2.0, 3.0);
ivec3 steps = ivec3(velocity);
```

```
vec4 color;
vec3 RGB = vec3(color);
```

```
vec3 white = vec3(1.0);  // white = (1.0, 1.0, 1.0)
vec4 translucent = vec4(white, 0.5);
```

OpenGL Shading Language

- 집합체 사용

Component Accessors	Description
(x, y, z, w)	components associated with positions
(r, g, b, a)	components associated with colors
(s, t, p, q)	components associated with texture coordinates

```
float red = color.r;
float v_y = velocity.y;
```

```
float red = color[0];
float v_y = velocity[1];
```

```
color = color.abgr;
```

```
vec4 color = otherColor.rgz;
```

vec4인데 r,g,z 3개만 초기화를 해서 오류 남.

```
vec2 pos;
float zPos = pos.z;
```

vec2에는 .z 가 없는데 사용하려고 해서
오류가 난다.

OpenGL Shading Language

- 집합체 사용 연습문제

시험에 따로 내는 것은 아닌데 필요한 것이라 넣어봄. 알아서 잘 풀어보기.

```
mat4 m = mat4(2.0);
vec4 zVec = m[2];
float yScale = m[1][1];
```

m?
zVec?
yScale?

OpenGL Shading Language

- 함수 선언

in, out 으로 선언을 할 수 있다?
포인터냐, 포인터 주소냐, 콘스탄트로 카피하냐, 그냥 카피해서 넘기냐 등등.
자세히 설명하진 않는다.

```
returnType functionName([accessModifier] type1 variable1,
                        [accessModifier] type2 varaible2,
                          ...)
{
    // function body
    return returnValue; // unless returnType is void
}
```

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

OpenGL Shading Language

- 함수 선언

쉐이더 코드 디버깅은 어떻게 하는가? 화면에 출력해서 눈으로 보고 확인하는 방법 많이 쓴다. 노가다긴 함.

```
float addSome(float a, out float b)
{
    a = a+1.0;
    b = a;
    Return a+b;
}
```

```
float addSome(const in float a, float b)
{
    a = a+1.0;
    b = a;
    Return a+b;
}
```

여기서 나온 값을 확인하고자 할 때는
칼라를 Green으로 하거나 Red로 하거나 등등
컬러값으로 출력해 디버깅을 한다.

```
float addSome(float a, inout float b)
{
    a = a+1.0;
    b = a;
    Return a+b;
}
```

```
float addSome(in float a, float b)
{
    a = a+1.0;
    b = a;
    Return a+b;
}
```

Shader 입출력

Shader 입출력

- 각 Shader 단위로 입력 값과 출력 값이 있으며 고유의 입출력 값과 사용자 정의 입출력 값이 있음

지난 시간에 한 것은 이거. 버텍스의 값을 직접 만들었다. 좌표를 만들어서 어레이로 넣음.
OpenGL 에서 자동으로 만든 것이 아니라 사용자가 정의한 입력 값이라 한다.
오늘 이것에 대해 더 알아본다.

- 사용자가 정의한 입력 값

- OpenGL 고유의 입력 값
  - gl_VertexID
  - gl_InstanceID
  - …

위의 명령은
이 수업에서 이것을 쓸 일은 없겠지만
알아두면 나중에 도움이 될 것이다.

Vertex Shader

- 사용자가 정의한 VS 출력 값

- OpenGL 고유의 출력 값
  - gl_Position
  - gl_PointSize
  - …

OpenGL에서 무조건 출력이 되어야 하는 것?
- 포지션
어디에 그릴 것인지를 판단해야 그릴지 말지를 결정함.
이걸 넣지 않으면 오류를 낸다.

Shader 입출력

- 각 Shader 단위로 입력 값과 출력 값이 있으며 고유의 입출력 값과 사용자 정의 입출력 값이 있음

결정되는 형태에 따라 FS가 몇개가 될지는 계속 달라짐.
그것을 주의해서 생각해야 한다.

프래그먼트 쉐이더

- 사용자가 정의한 FS 입력 값
- OpenGL 고유의 입력 값
  - gl_FragCoord
  - gl_FrontFacing
  - gl_PointCoord
  - …

이때도 고유의 값을 받는다. 이 수업에서는 일일이 설명할 필요는 없다.

Fragment Shader

입력된 개수만큼 출력되는데 도저히 필요없는 값이라면 건너뛰기도 함.
사용자가 정의한 FS 출력 값을 쓰는 이유?
- 기본적으로는 OpenGL 고유의 출력값은 프레임버퍼에 나옴. 여기에 다양한 버퍼들을 추가할 수가 있다. 사용자가 정의한 출력 값을 저장하도록 정리할 수가 있다는 것.

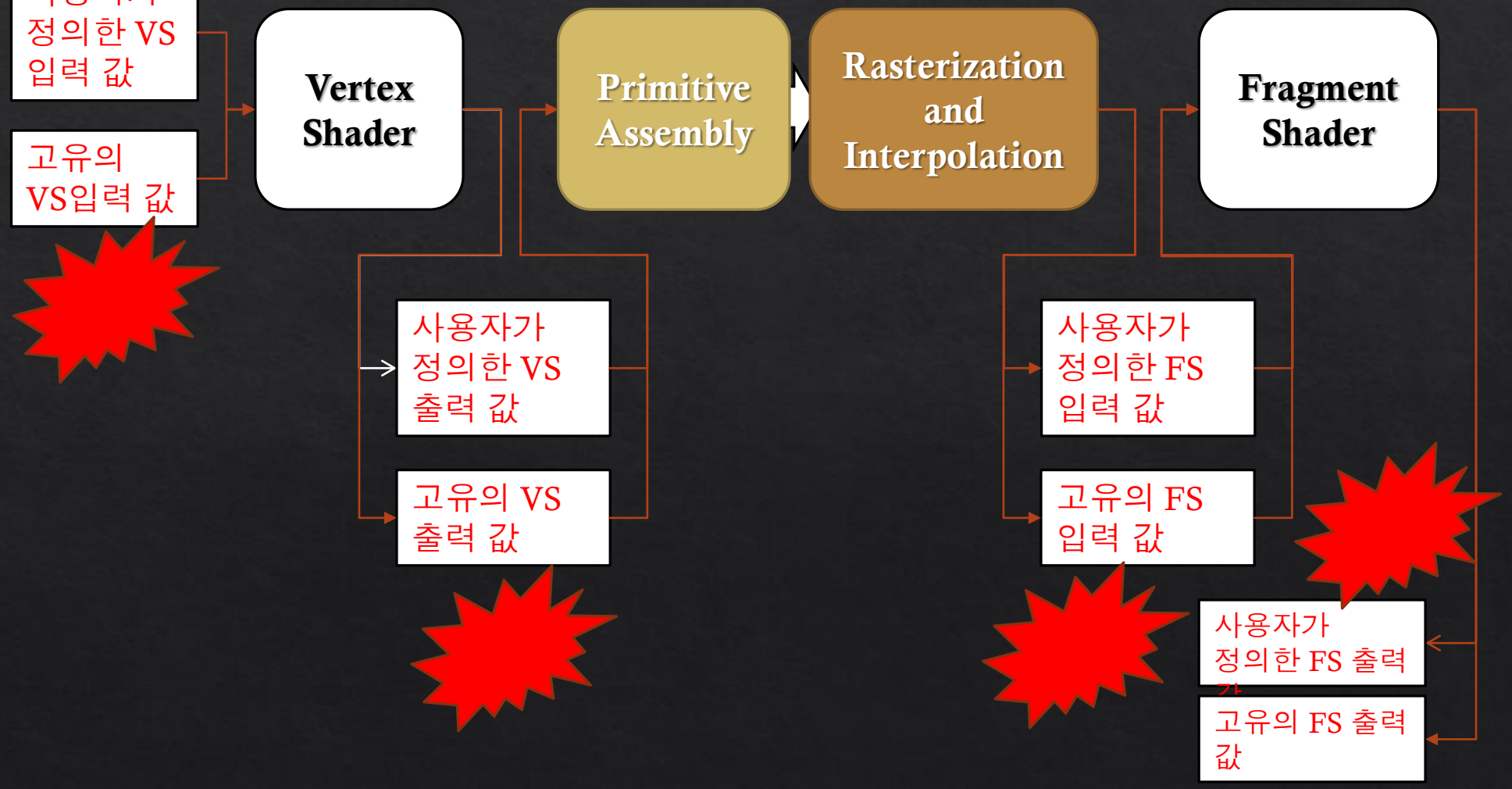
- 사용자가 정의한 FS 출력 값
- OpenGL 고유의 출력 값
  - gl_FragDepth
  - …

프래그먼트 쉐이더는 맨 마지막 단계로, 색 출력하는 거다. 그래서 제일 중요한 출력값은 '컬러'다. 일부러 물어보려고 PPT에 안 쓴 것이다.

이 네 단계 중에서 가장 신경써야 하는 스테이지는 Vertex Shader다. 유일하게 외부의 입력을 받는 스테이지기 때문에. 그래서 오늘 정리해본다.

Shader 입출력

이 중간 단계에서는 개수가 크게 바뀔 수가 있다는 것을
주의해서 봐야 한다.
레스터라이제이션 단계의 역할을 잘 알고 있어야 한다.

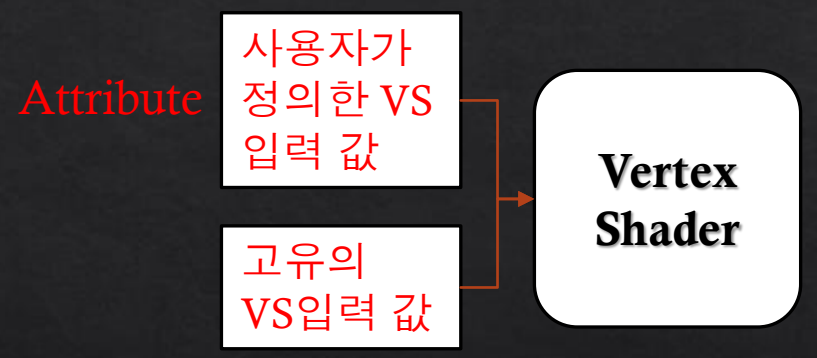


Vertex shader 입력

Vertex Shader 입력

- 사용자가 정의한 VS 입력 값
  - Attribute 라 칭함

버텍스셰이더의 입력값은 어트리뷰트라 한다. 그럼 함수들의 이름이 이해가 됨.
어트리뷰트는 하나가 아니라 굉장히 많아질 수가 있고, 선언된 어트리뷰트마다 제어를 할 수 있어야 한다.
그렇기 때문에 인덱스 형태로 관리가 된다.




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

0이라는 게 어트리뷰트를 제어하기 위한 인덱스를 말한다. 인덱스가 0이라는 뜻이었음.

Vertex Shader 입력

사용자가 정의한 입력 값

→ layout (location = 0) in vec3 Position;

버텍스 셰이더 코드의 layout 부분과 연동됨

Vertex Shader

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

Vertex Shader 입력

이런 모양의 삼각형 두개
출력하는 것 해 봤다.

사용자가 원하는 입력 값이 두 개라면?

사용자가 정의한 입력 값

→ layout (location = 0) in vec3 Position;
→ layout (location = 1) in vec3 Color;

Vertex Shader

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

glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, VBO1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);
```

Vertex Shader 입력

- layout 없이도 사용 가능
  - GLint glGetAttribLocation(GLuint program, const char* name);

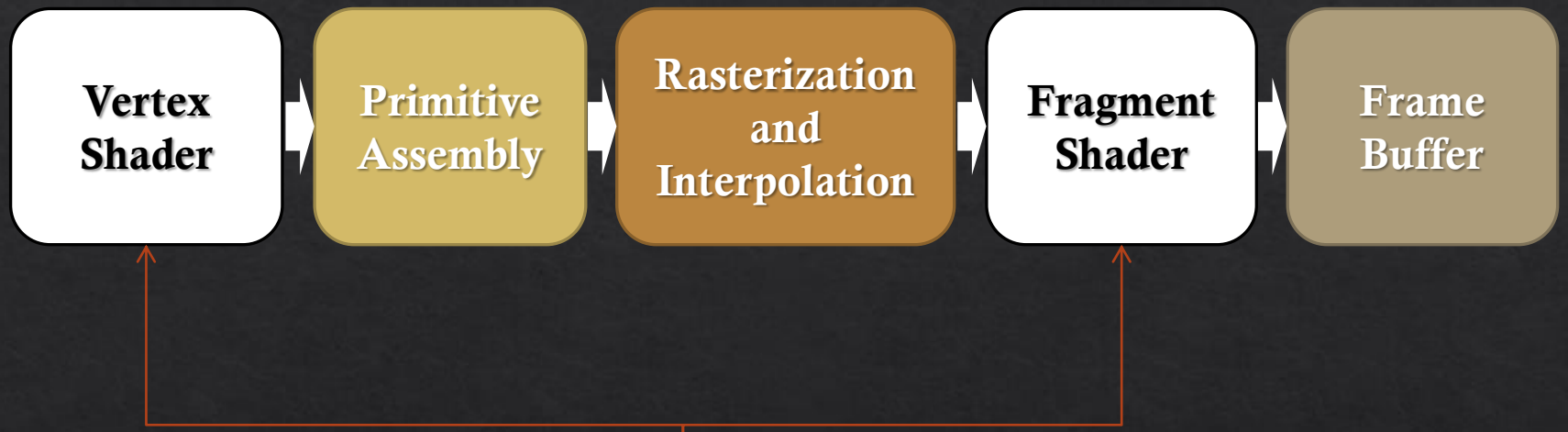
사용자가 정의한 입력 값

→ in vec3 Position;
→ in vec3 Color;

```
GLint id0 =  glGetAttribLocation(program, “Position”);
GLint id1 =  glGetAttribLocation(program, “Color”);

glEnableVertexAttribArray(id0);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glVertexAttribPointer(id0, 3, GL_FLOAT, GL_FALSE, 0, 0);

glEnableVertexAttribArray(id1);
glBindBuffer(GL_ARRAY_BUFFER, VBO1);
glVertexAttribPointer(id1, 3, GL_FLOAT, GL_FALSE, 0, 0);
```

shader 외부 입력

Shader 외부 입력



모든 버텍스에 동일한 값을 적용한다면?
변환행렬이 대표적인 예다. 프로젝트 매트릭스 같은 것. 모든 버텍스에 동일한 것이 곱해짐.
시간도 대표적인 예다. 시간에 따라 움직여보고 싶을 때 입력되는 메쉬를 시간에 따라 움직인다면
프레임 하나가 정해져 있어야 한다. 공통적으로 적용되는 외부입력 변수다.
관절마다 가중치에 대한 것도? 그렇다고 함. ㅇㅇ...

외부 입력 변수
- 시간에 관계된 변수
- 가중치에 관계된 변수
- 기타..

Shader 외부 입력

- uniform 선언자 사용

in이라 선언된 것은 attribute고, 유니폼은 전반적인 파이프라인에 공통되게 적용할 수 있는 값이다. 드로우 어레이스?가 호출되기 전까지는 바꿀 수 있지만 호출된 후에는 바꿀 수 없다. 버텍스마다 있는 게 아니라 전체적 파이프라인에 하나가 있는 값이다. 하나만 있는 건 아니고 다양한 자료형으로 여러 개 존재 가능. static 변수와 비슷하게 작동한다고 생각하면 됨. 매 프레임마다 새롭게 작용.

  - uniform 으로 선언된 변수는 외부에서 입력된 값을 가지고 있으며 셰이더 내부에선 읽기만 가능하고 쓰기는 불가능 함
  - 셰이더 전반에 걸쳐 읽기가 가능함

Shader 외부 입력

```
layout (location = 0) in vec3 Position;

uniform float gScale; //외부에서 값을 넘겨받음

void main()
{
    gl_Position = vec4(gScale * Position.x, gScale * Position.y, Position.z, 1.0);
}
```

Shader 외부 입력

- GLint glGetUniformLocation(GLuint program, const char* name);

```
layout (location = 0) in vec3 Position;

 글로벌 변수 같은 것인데 유니폼이 붙는다. 값을 주기 위해서는 로케이션을 받아야 한다. 로케이션값을 이용해서 값을 set 해준다.
uniform float gScale; //외부에서 값을 넘겨받음

void main()
{
    gl_Position = vec4(gScale * Position.x, gScale * Position.y, Position.z, 1.0);
}
```

- GLint id = glGetUniformLocation(program, “gScale”);

Shader 외부 입력

set 하는 방법은 유니폼이 붙여 쓴다.

- void glUniform{1234}{fdi ui}(GLint location, TYPE value);
- void glUniform{1234}{fdi ui}v(GLint location, GLsizei count, const TYPE * values);
- void glUniformMatrix{234}{fd}v(GLint location, GLsizei count, GLboolean transpose, const GLfloat * values);
- void glUniformMatrix{2x3,2x4,3x2,3x4,4x2,4x3}{fd}v(GLint location, GLsizei count, GLboolean transpose, const GLfloat * values);

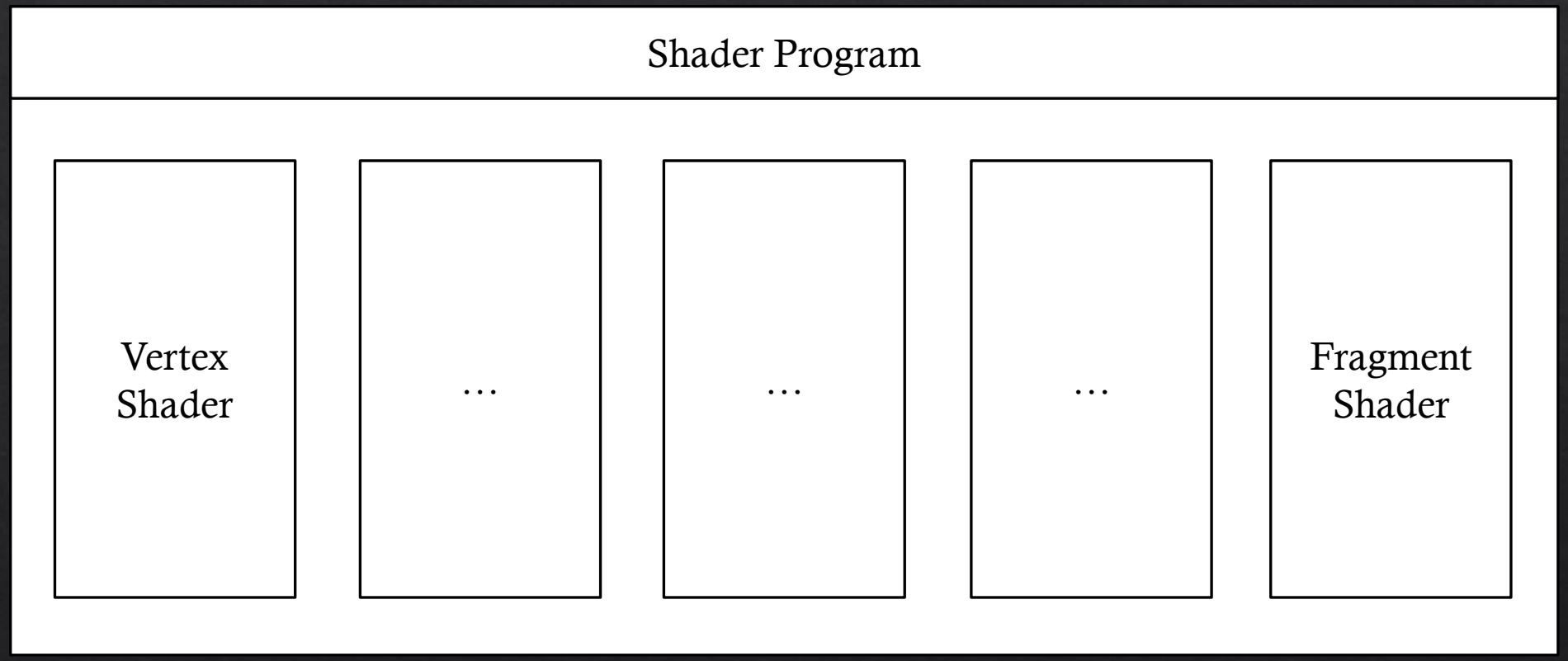
Shader 외부 입력

```
layout (location = 0) in vec3 Position;

uniform float gScale; //외부에서 값을 넘겨받음

void main()
{
    gl_Position = vec4(gScale * Position.x, gScale * Position.y, Position.z, 1.0);
}
```

- GLint id = glGetUniformLocation(program, “gScale”);
- glUniform1f(id, 0.5f);

Shader 컴파일

Shader 컴파일



Shader 컴파일

#define GL_FRAGMENT_SHADER
#define GL_VERTEX_SHADER

0. Shader Program 생성

```
GLuint ShaderProgram = glCreateProgram();
```

1. Shader Object 생성

```
GLuint ShaderObj = glCreateShader(ShaderType);
```

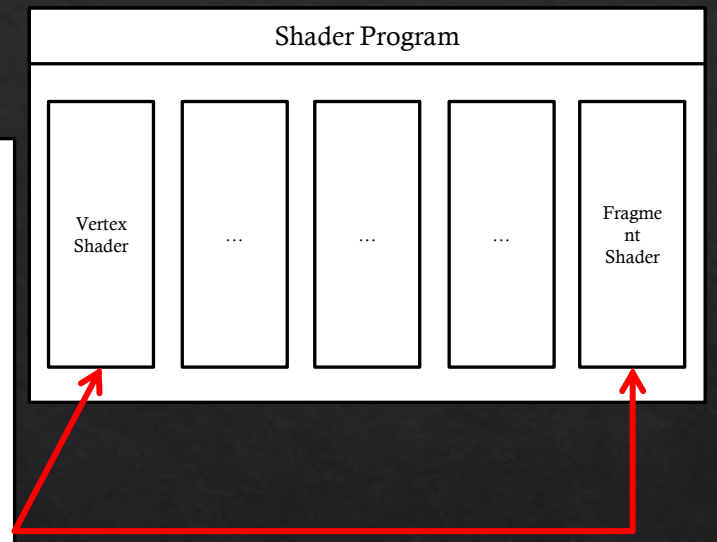
2. Object 에 Source 할당 및 컴파일

```
glShaderSource(ShaderObj, 1, p, Lengths);
glCompileShader(ShaderObj);
```

3. 컴파일 완료 확인

3. Shader Program 에 Attach

```
glAttachShader(ShaderProgram, ShaderObj);
```



Vertex Shader 및 Fragment Shader Object 를 각각 생성 후 Attach 함

Shader 컴파일

- Attach 완료 후 링크 수행
  - glLinkProgram(ShaderProgram);
- 링크 완료 확인
- 성공적으로 링크가 끝났으면 사용 준비 끝
- 사용시 아래 아래 함수를 사전에 불러줘야 해당 shader 프로그램이 동작함
  - glUseProgram(GLuint program);

실습 : 외부 입력 추가

실습 : 외부 입력 추가

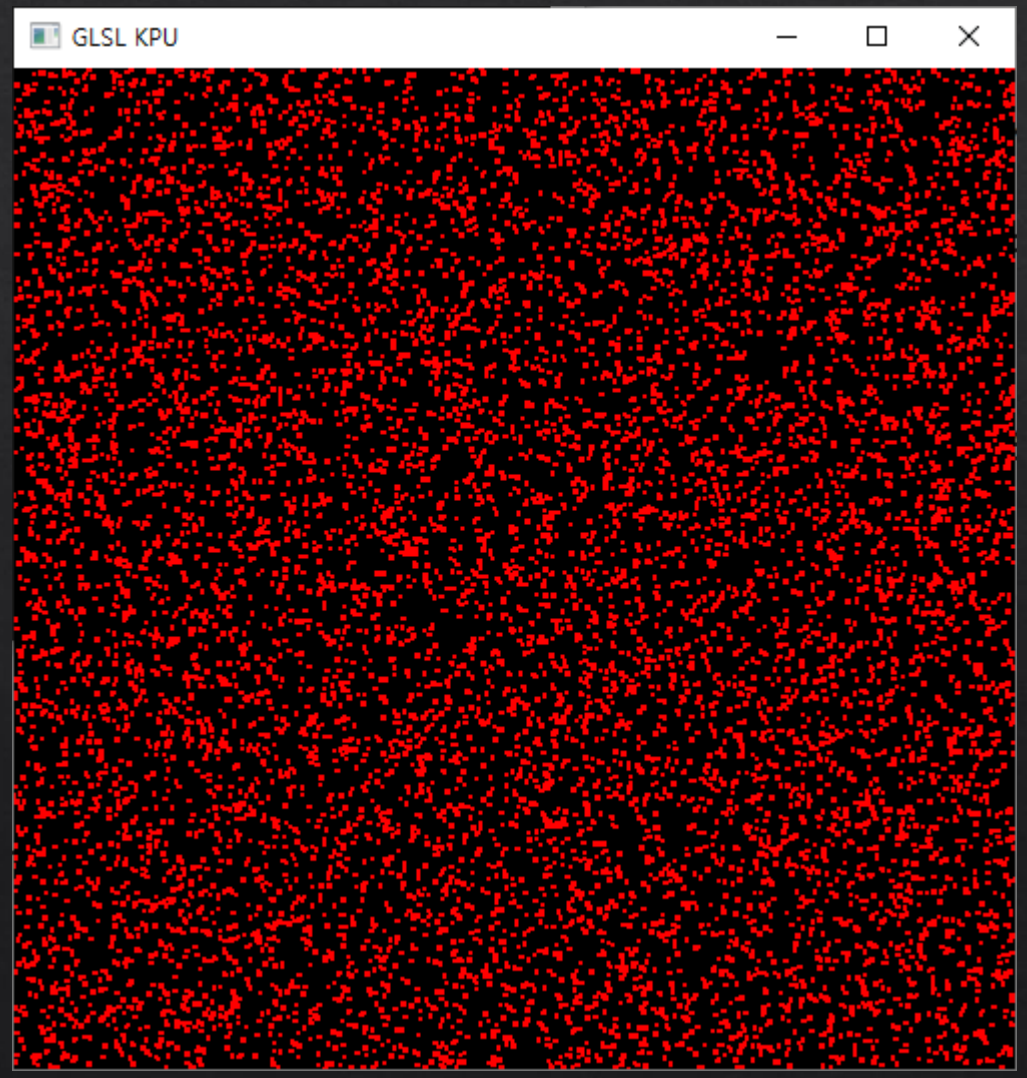
Color 입력

Position 입력

변화가 있는 입력

실습 : 외부 입력 추가

- 사각형 파티클 생성 후 화면에 그려보기
- 랜덤 위치에 그려보기
- 파티클 움직임 적용 해 보기
  - 속도 부여
  - 더 필요한 사항은?



실습 : 간단한 파티클 애니메이션1

- 버텍스 Attribute
  - 위치
    - {(Random Position), …..}

실습 : 간단한 파티클 애니메이션2

- 버텍스 Attribute
  - 위치, 속도
    - {(Random Position), (Random Velocity), …..}
- 셰이더
  - 속도 계산 추가

실습 : 간단한 파티클 애니메이션3

- 버텍스 Attribute
  - 위치, 속도
    - {(Random Position), (Random Velocity), …..}
- 셰이더
  - 가속도 추가

실습 : 간단한 파티클 애니메이션4

- 버텍스 Attribute
  - 위치, 속도, 시작 시간, 생명
  - {(Random Position), (Random Velocity), (Emit Time, Life Time)…..}
    - Emit Time
      - 파티클이 생성되는 시작 시간
        - EX: 파티클 배치 후 몇 초 후에 생성될 것인가?
    - Life Time
      - 파티클이 유지되는 시간
        - EX: 파티클 생성 후 몇 초간 유지될 것인가?

실습 : 간단한 파티클 애니메이션5

- 버텍스 Attribute
  - 위치, 속도, 시작 시간, 생명, 주기, 폭
  - {(Random Position), (Random Velocity), (Emit Time, Life Time), (ratio, amplitude)…..}
    - ratio, amplitude
      - Sin 함수 주기, 폭 결정 (랜덤 값)

실습 : 간단한 파티클 애니메이션5

윈도우 중앙에서 오른쪽으로 파티클 발사

실습 : 간단한 파티클 애니메이션5

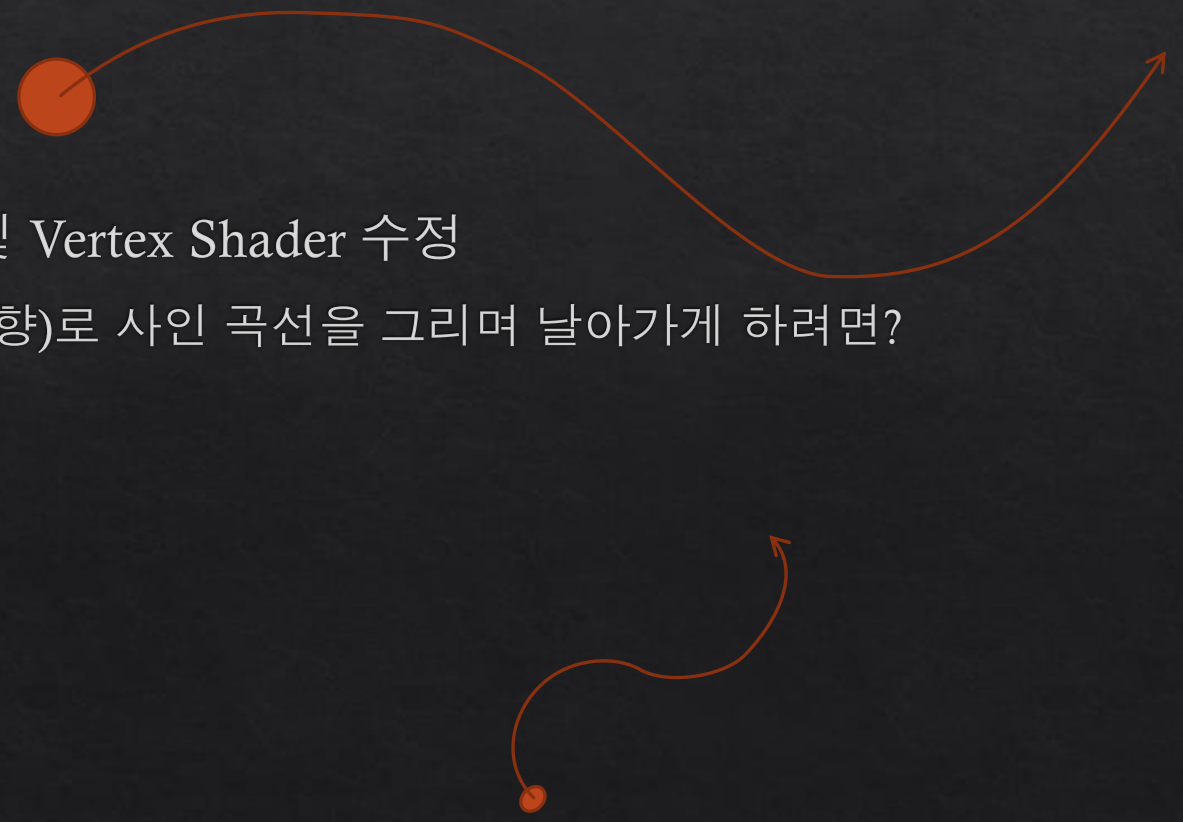
파티클 움직임이 사인 곡선을 따라가도록 함

실습 : 간단한 파티클 애니메이션5

사인곡선의 주기, 폭을 랜덤하게 부여

실습 : 간단한 파티클 애니메이션5

- Vertex Shader 만 수정
  - Emit 후 점차적으로 퍼지게 하려면?

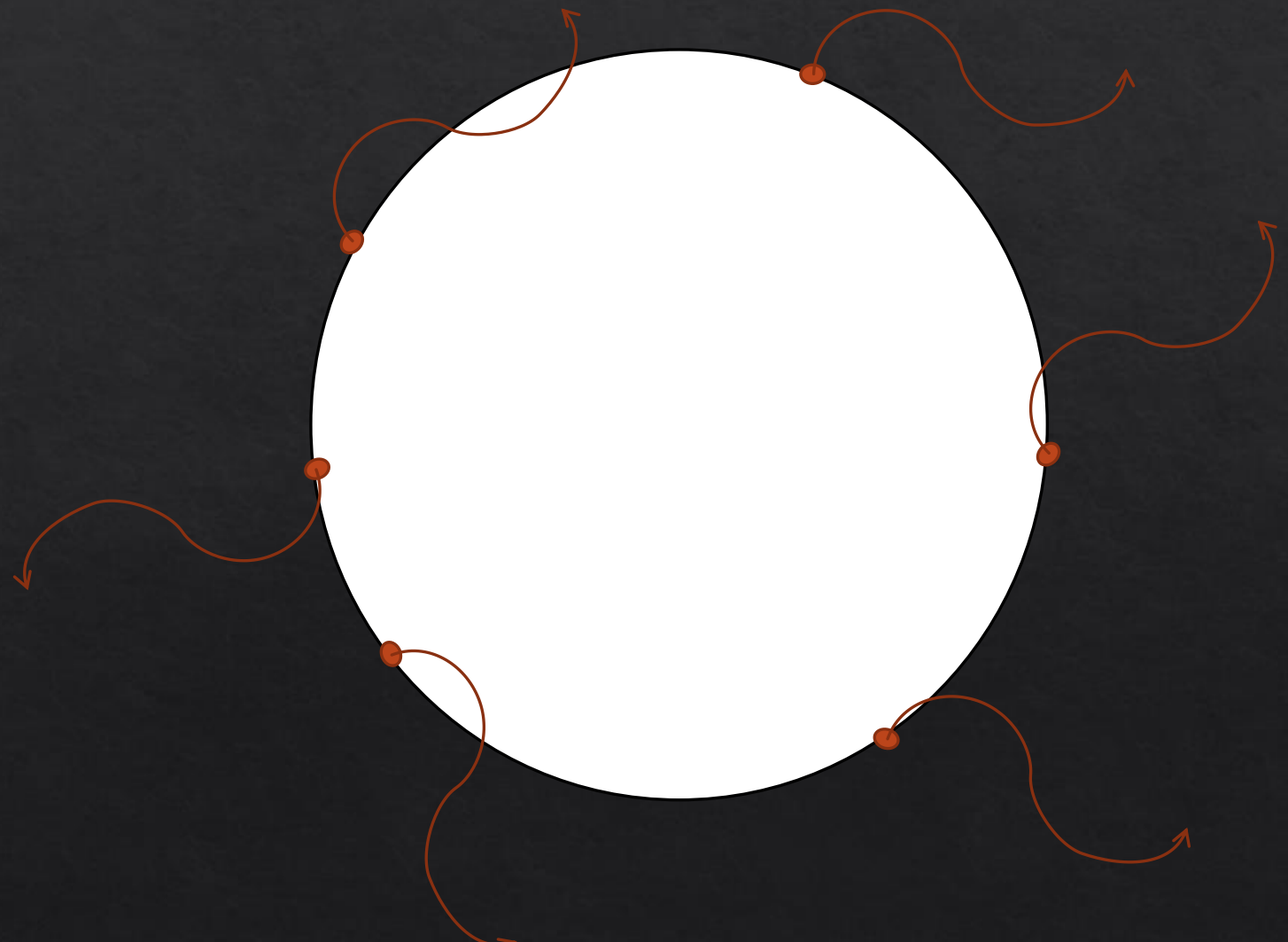


- Rendering 코드 및 Vertex Shader 수정
  - 주어진 속도(방향)로 사인 곡선을 그리며 날아가게 하려면?

실습 : 간단한 파티클 애니메이션6

- 버텍스 Attribute
  - 위치, 속도, 시작 시간, 생명, 주기, 폭, 고유 값
  - {(Random Position), (Random Velocity), (Emit Time, Life Time), (ratio, amplitude), (value)…..}
    - Value
      - 다양한 용도로 사용 가능한 값
        - EX: 시작 위치

실습 : 간단한 파티클 애니메이션6



실습 : 간단한 파티클 애니메이션6

- 중력 가속도 추가
- 외부 입력 가속도 추가