

Metal / Vulkan 을 이용한 지도 렌더링

김봉주 / 조영욱 NAVER Glace

CONTENTS

1. 프로젝트 소개
2. Metal 정점 바인딩
3. Metal 유니폼 바인딩
4. Vulkan 정점 바인딩
5. Vulkan 유니폼 바인딩
6. 결론 및 향후 계획

1. 프로젝트 소개

1.1 지도는 3D

멀티 플랫폼 풀 벡터 3D 렌더링

- 호환성이 뛰어난 OpenGL ES 2.0 렌더러
- iOS / Android / Linux 앱에서 공통으로 사용



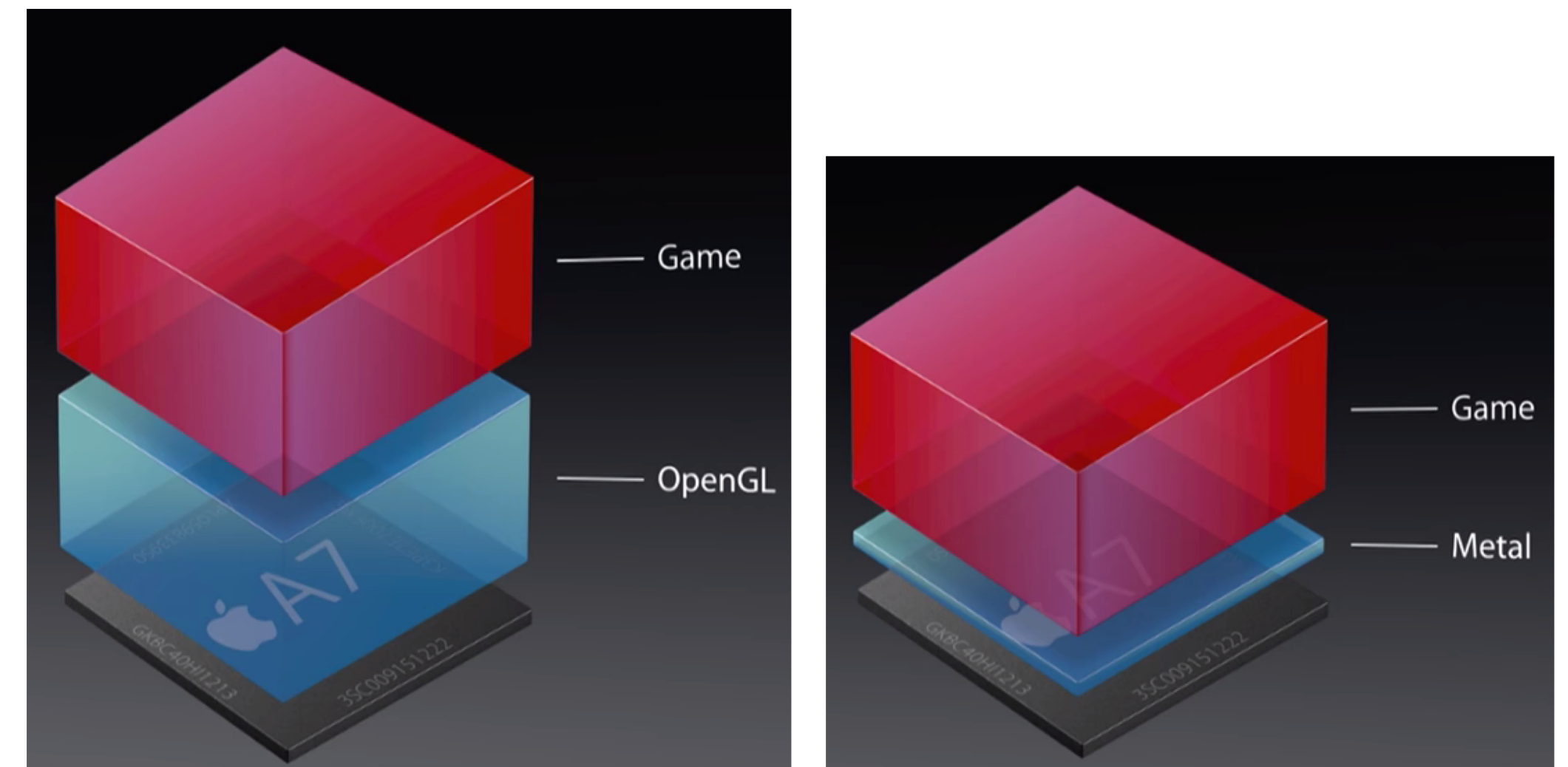
1.2 OpenGL 의 대안, Metal 과 Vulkan

OpenGL 의 문제

- 기능적 제약
- 안정성 부족
- 퍼포먼스 한계

Metal 과 Vulkan

- 강력하고 쉬운 low-level API
- 뛰어난 성능
- 개발 편의성 (Metal)



<https://www.youtube.com/watch?v=w87fOAG8fjk>

1.3 마이그레이션 (OpenGL to Metal)

세 가지 포팅

1. CPU 코드(API) 포팅

```
glBindBuffer(GL_ARRAY_BUFFER, positionData);
glBindBuffer(GL_UNIFORM_BUFFER, matrixData);
```



```
[encoder setVertexBuffer:positionData offset:0 atIndex:0];
[encoder setVertexBuffer:matrixData offset:0 atIndex:1];
```

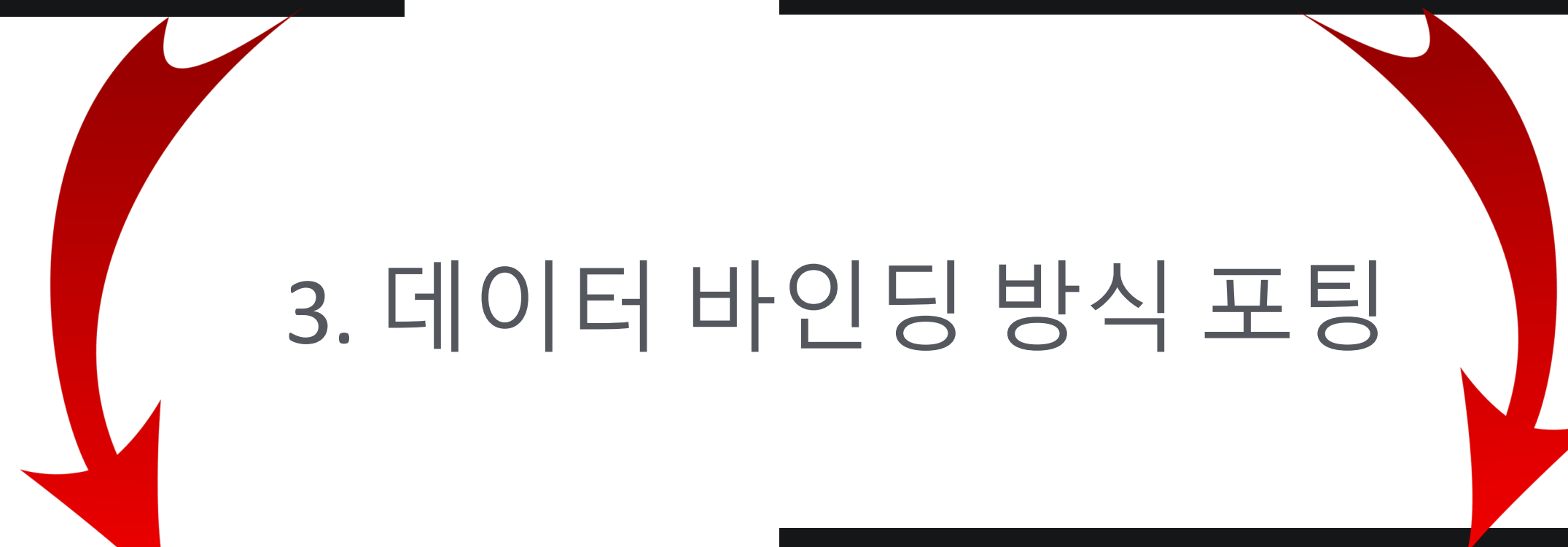
2. GPU 코드(셰이더) 포팅

```
attribute vec2 a_position;
uniform mat4 u_matrix;
```



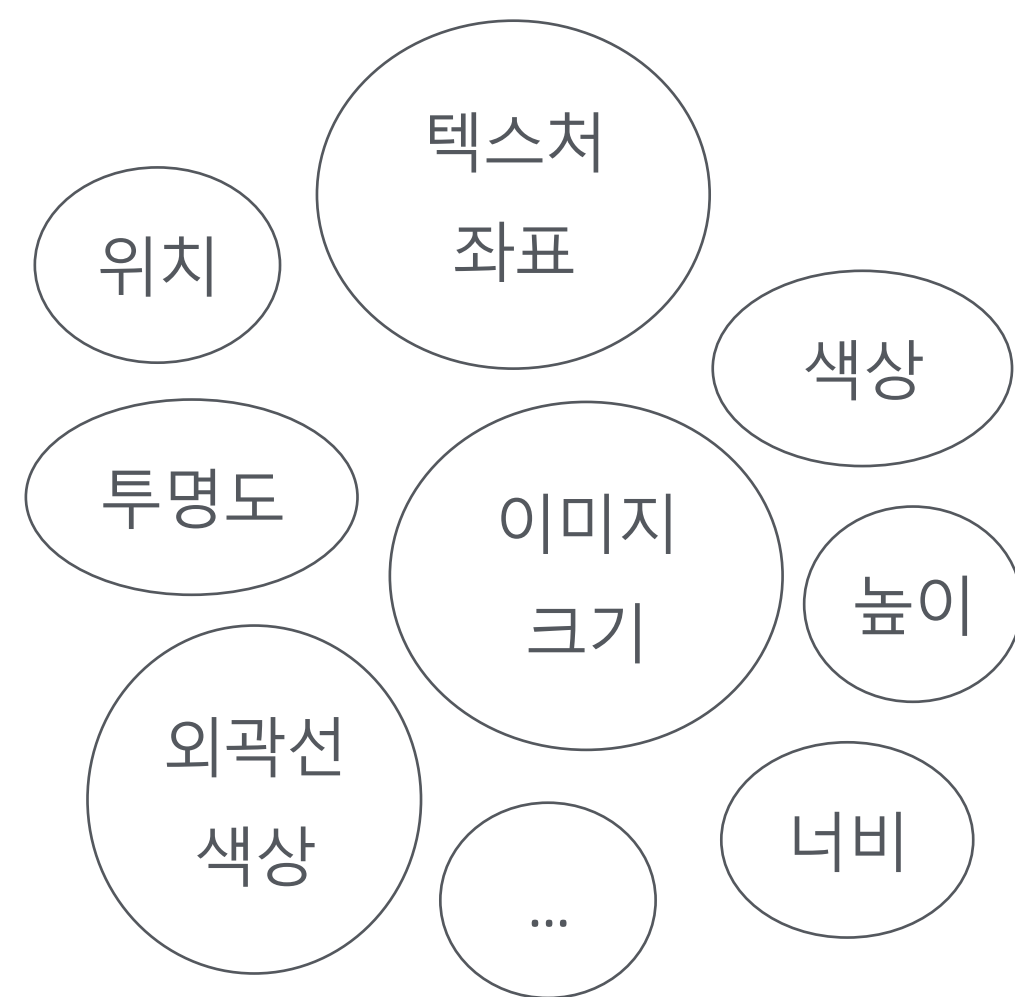
```
vertexShader(constant short2* positions [[buffer(0)]],
             uint vid [[vertex_id]],
             constant float4x4& matrix [[buffer(1)]] { ... }
```

3. 데이터 바인딩 방식 포팅

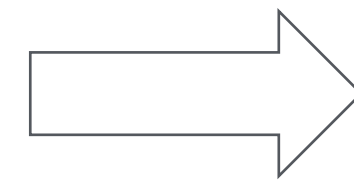


1.4 데이터 바인딩

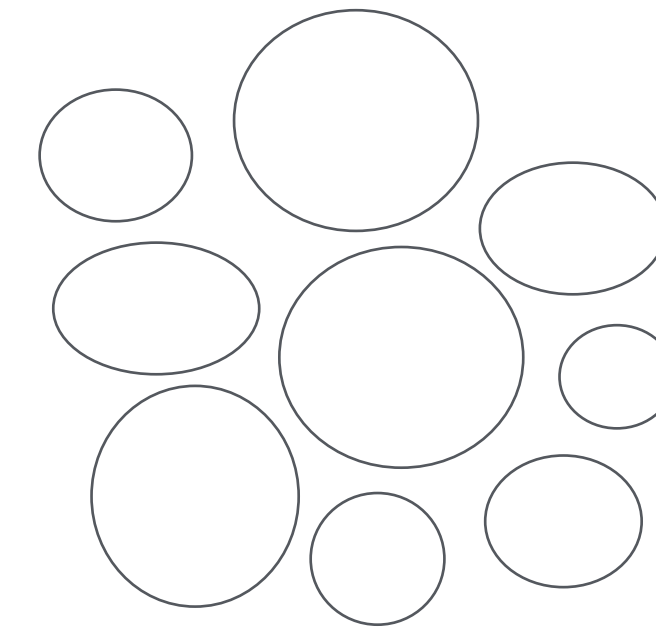
데이터(메모리)에 맞게 셰이더의 변수(크기, 정렬) 맞춤



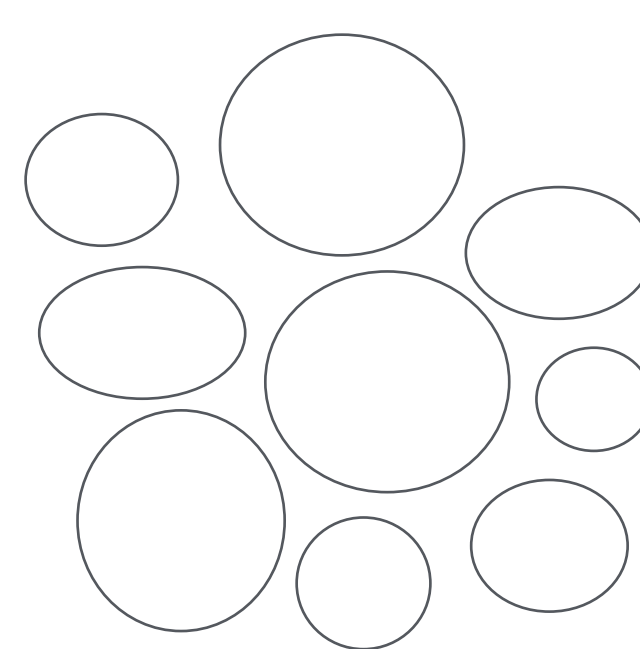
C++ 에서 생성한
iOS / Android / Linux
공통 데이터



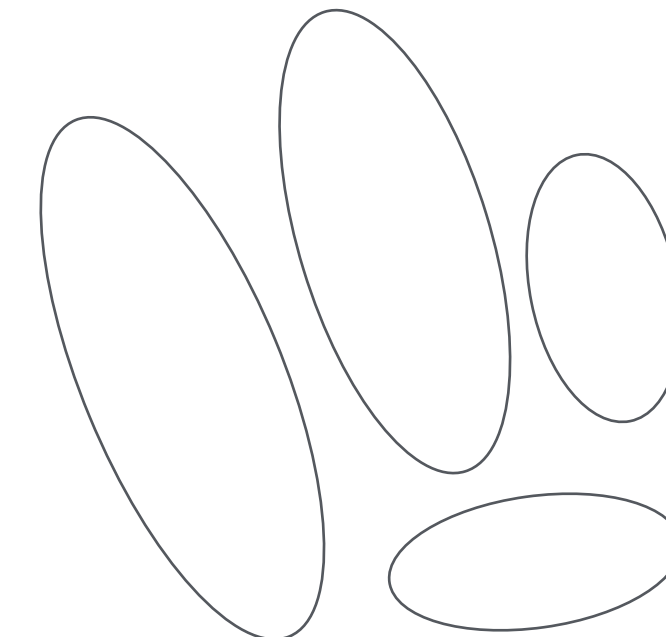
OpenGL



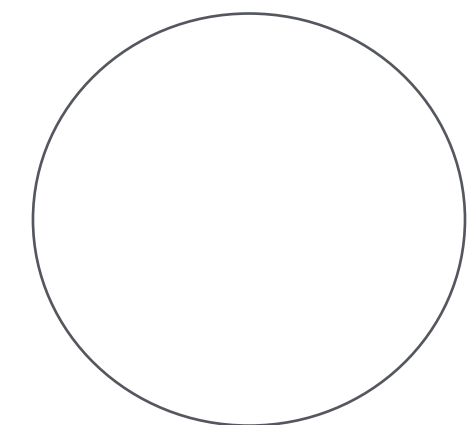
Metal
Vulkan



or



or



2. Metal 정점 바인딩



2. Metal 정점 바인딩

2.1 태초에 위치만 있었다



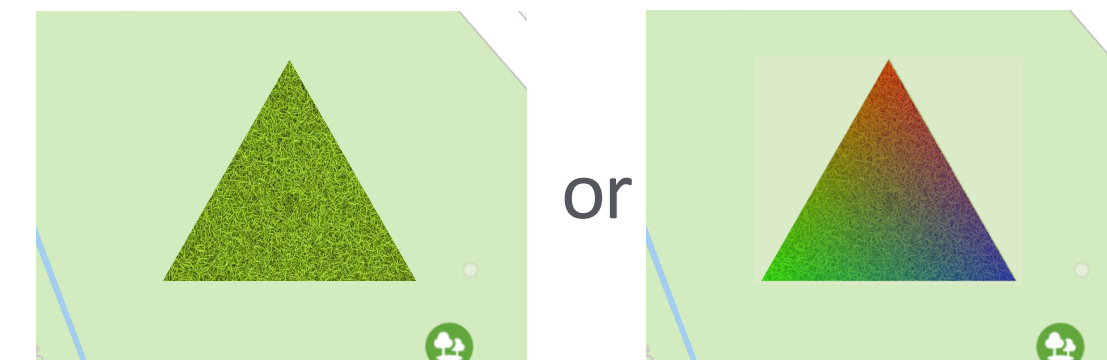
2.2 풀 텍스처 좌표 추가



2.3 색상 추가



2.4 색상이 있거나 혹은 없거나



2.1 태초에 위치만 있었다

위치 타입 그대로 매개 변수에 사용

CPU

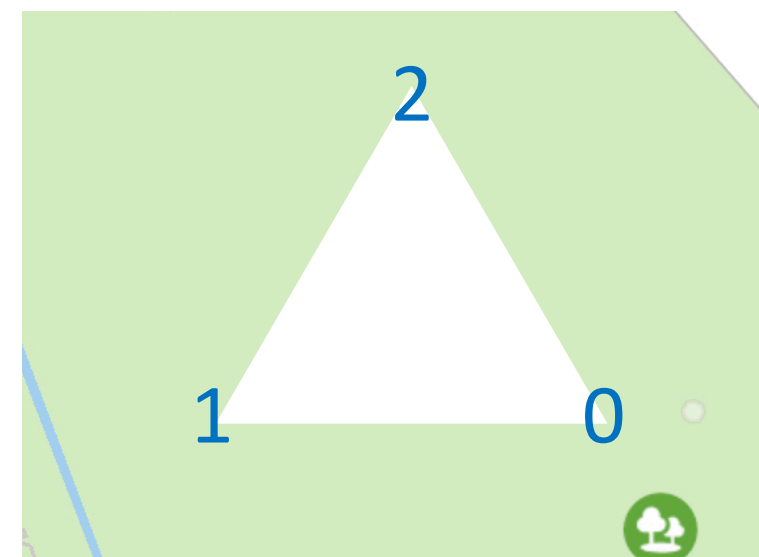
0	1	2
(250, -250)	(-250, -250)	(0, 250)

(made in C++)

`encoder.setVertexBuffer(positionData, 0, 0);`

(Objective C, Swift)

0	(250, -250)	(-250, -250)	(0, 250)
1			



GPU

```
vertex VertexOut
vertexShader(constant short2* positions [[buffer(0)]],
              uint vid [[vertex_id]]) {
    short2 pos = positions[vid];
    ...
}
```

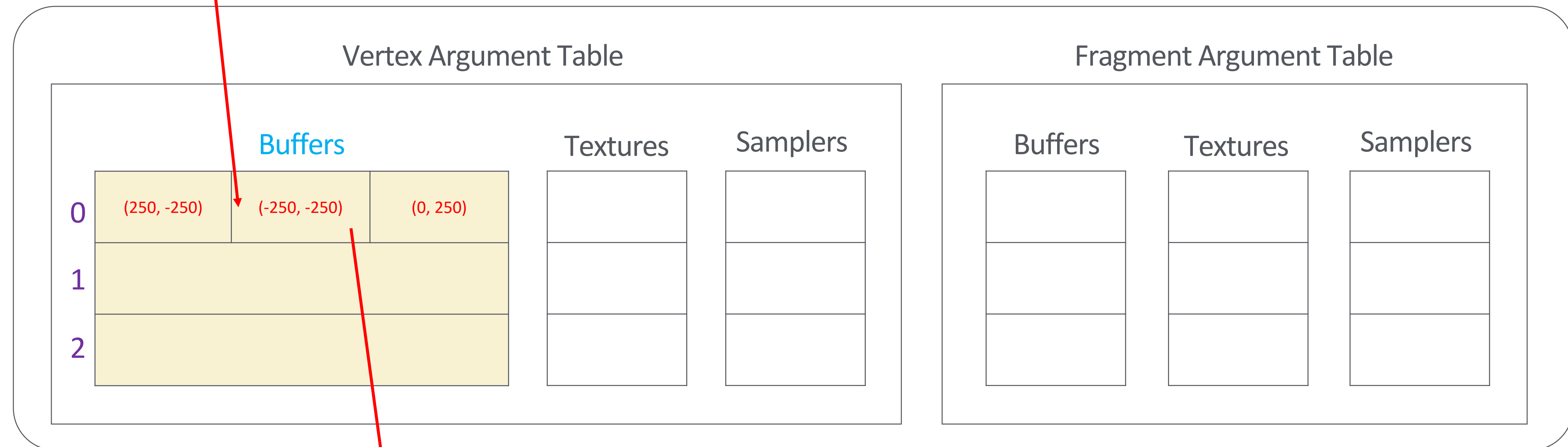
(Metal Shading Language)

Argument Table

buffer, texture, sampler 가 보관되는 곳

```
encoder.setVertexBuffer(positonData, 0, 0);
```

Render Command Encoder



```
vertexShader(constant short2* positions [[buffer(0)]],  
             uint vid [[vertex_id]])
```

Constant 주소 공간

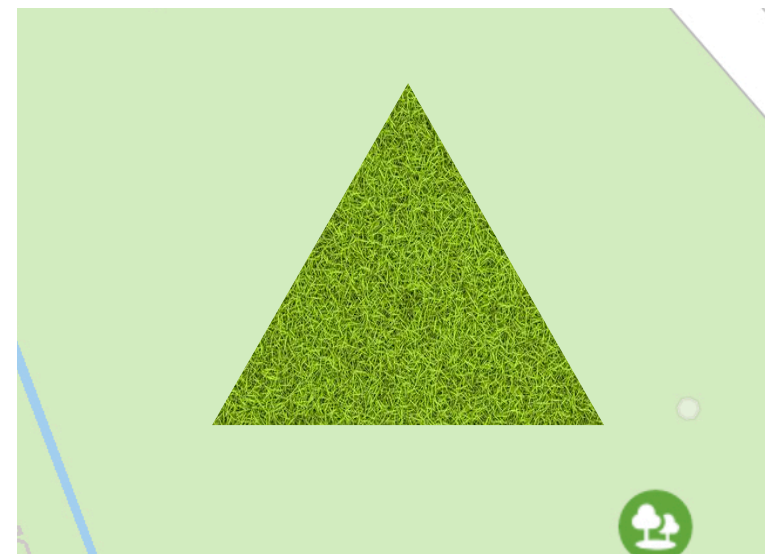
포인터나 참조 타입 변수 사용 시 반드시 주소 공간 명시

- device: readable and writeable
- constant: read-only

```
vertexShader(constant short2* positions [[buffer(0)]],  
             uint vid [[vertex_id]])
```

2.2 풀 텍스처 좌표 추가

정점 정보가 늘어나서 하나의 구조체로 관리



0

(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
-------------	--------	--------------	--------	----------	----------



```
struct VertexIn {
    short2 pos;
    float2 tex;
} __attribute__((packed aligned(4)));
```

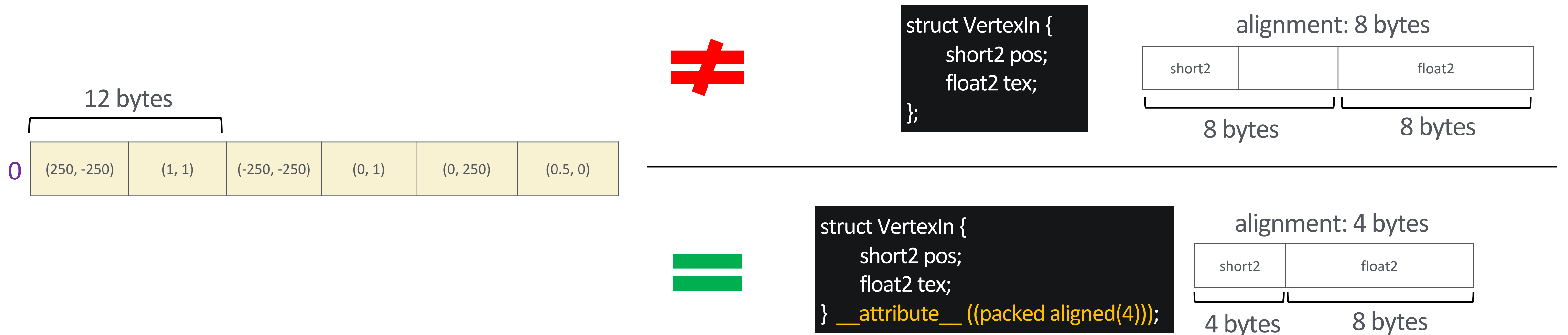
```
vertexShader(constant VertexIn* vertices [[buffer(0)]],
             uint vid [[vertex_id]])
```


alignment

구조체 alignment = 멤버 중 최대 alignment

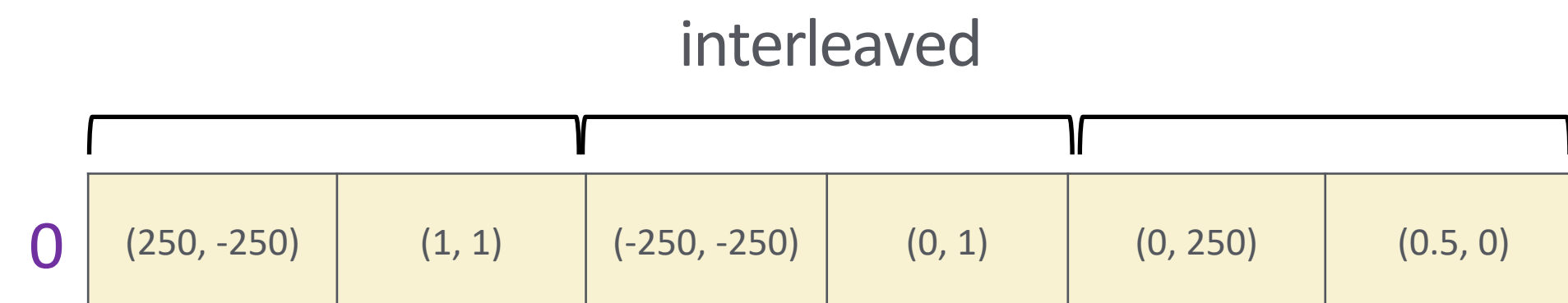
- 단, 최대 16 => iOS 메모리 alignment

구조체 크기 = 구조체 alignment 의 배수



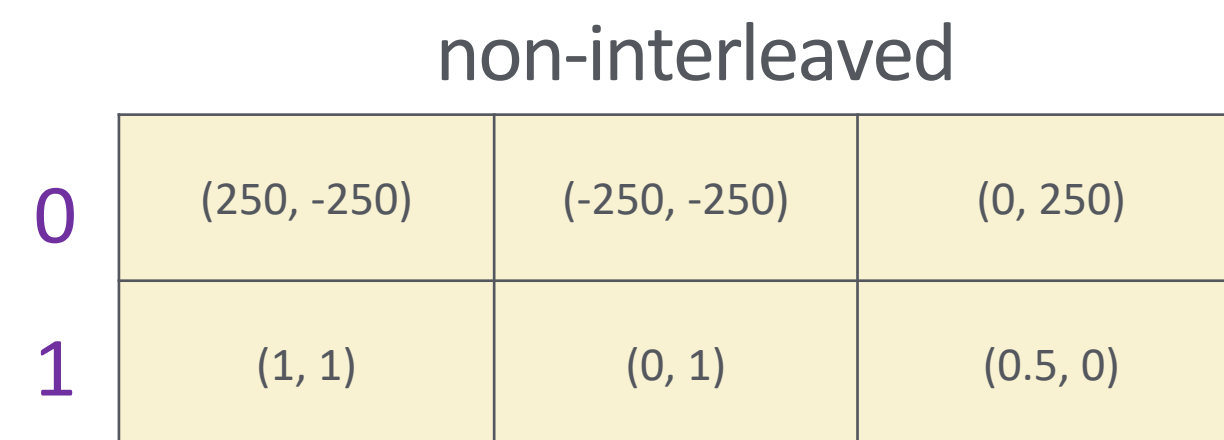
Interleaved 데이터

Interleaved 한 데이터만 하나의 버퍼로 받을 수 있음



```
struct VertexIn {
    short2 pos;
    float2 tex;
} __attribute__((packed aligned(4)));

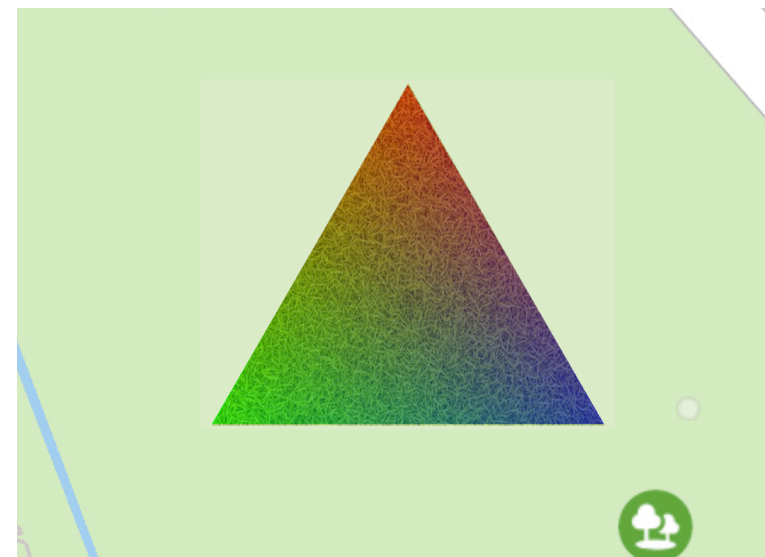
vertexShader(constant VertexIn* vertices [[buffer(0)]],
             uint vid [[vertex_id]])
```



```
vertexShader(constant short2* positions [[buffer(0)]],
             constant float2* tex [[buffer(1)]]
             uint vid [[vertex_id]])
```

2.3 색상 추가

정점 정보가 많아져서 바인딩이 부담될 땐 stage_in 사용



0	(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
1	(1, 0, 0, 1)		(0, 1, 0, 1)		(0, 0, 1, 0)	



```
struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
    float4 color [[attribute(2)]];
} __attribute__((packed, aligned(4)));

vertexShader(VertexIn in [[stage_in]])
```

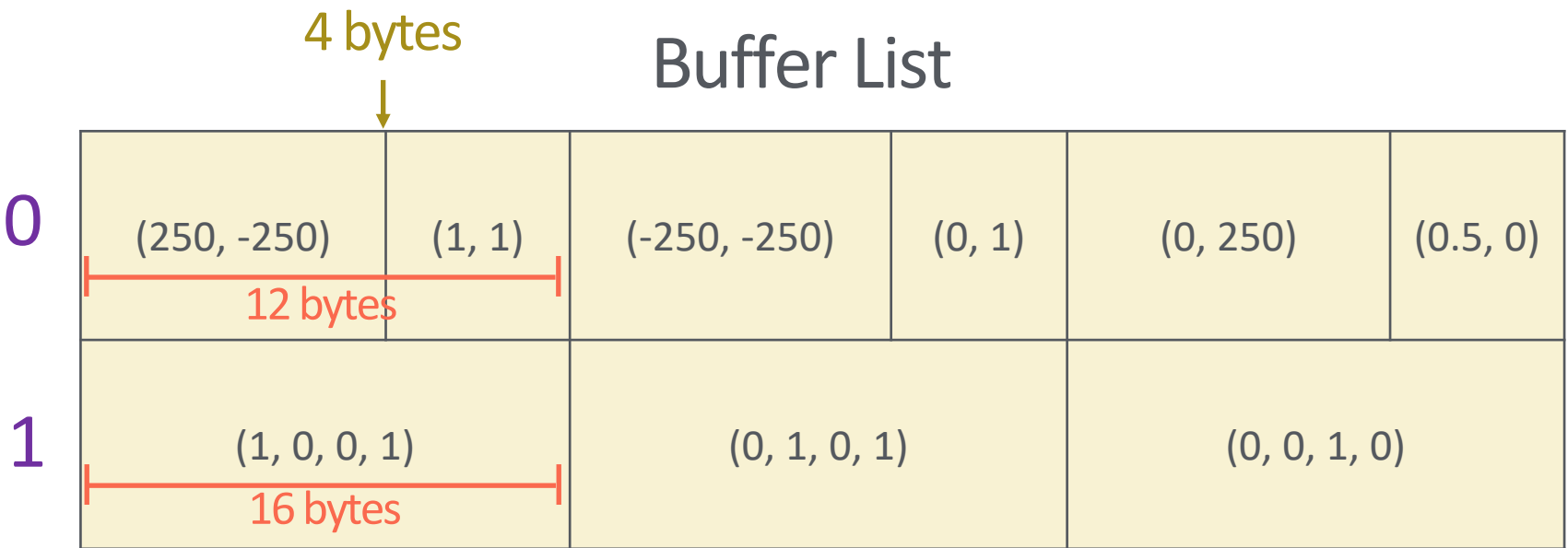

Vertex Descriptor

Vertex Descriptor 에 정점 정보(타입, 위치, 크기) 입력

MTLVertexDescriptor

Attributes			Layouts		
0	format	short2		0	stride
	buffer	0			
	offset	0			
1	format	float2		1	stride
	buffer	0			
	offset	4			
2	format	float4		2	stride
	buffer	1			
	offset	0			

```
struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
    float4 color [[attribute(2)]];
};
```



stage_in

정점 배열 + 인덱스 => 정점 하나

```
vertexShader(const VertexIn* vertices [[buffer(0)]],
             uint vid [[vertex_id]]) {
    VertexIn in = vertices[vid];
}
```

=

vertices + vid

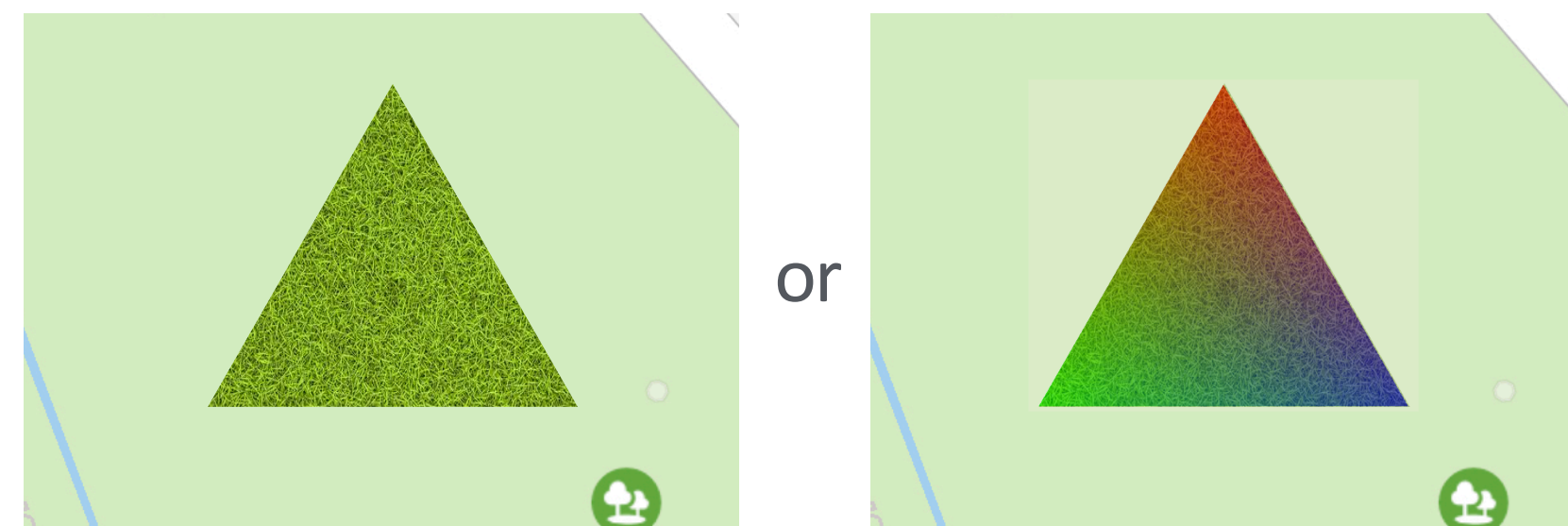


```
vertexShader(VertexIn in [[stage_in]]) {
}
```

2.4 색상이 있거나 혹은 없거나

옵셔널한 멤버에는 Function Constant 사용

- Function Constant 멤버 있을 땐 정점 배열로 받을 수 없음 -> stage_in 사용 필수



0	(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
1	(1, 0, 0, 1)		(0, 1, 0, 1)		(0, 0, 1, 0)	



```
constant bool attribute_color_defined [[function_constant(0)]];

struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
    float4 color [[attribute(2), function_constant(attribute_color_defined)]];
};

vertexShader(VertexIn in [[stage_in]])
```

Function Constant

셰이더 함수 컴파일 시 입력할 수 있는 특별한 상수

- 전처리기 명령어 (#ifdef ~ #else ~ #endif) 대체

Objective C

```
MTLFunctionConstantValues* constantValues = [MTLFunctionConstantValues new];

bool attributeColorDefined = true;
[constantValues setConstantValue:&attributeColorDefined type:MTLDataTypeBool atIndex:0];
bool attributeOpacityDefined = false;
[constantValues setConstantValue:&attributeOpacityDefined type:MTLDataTypeBool atIndex:1];

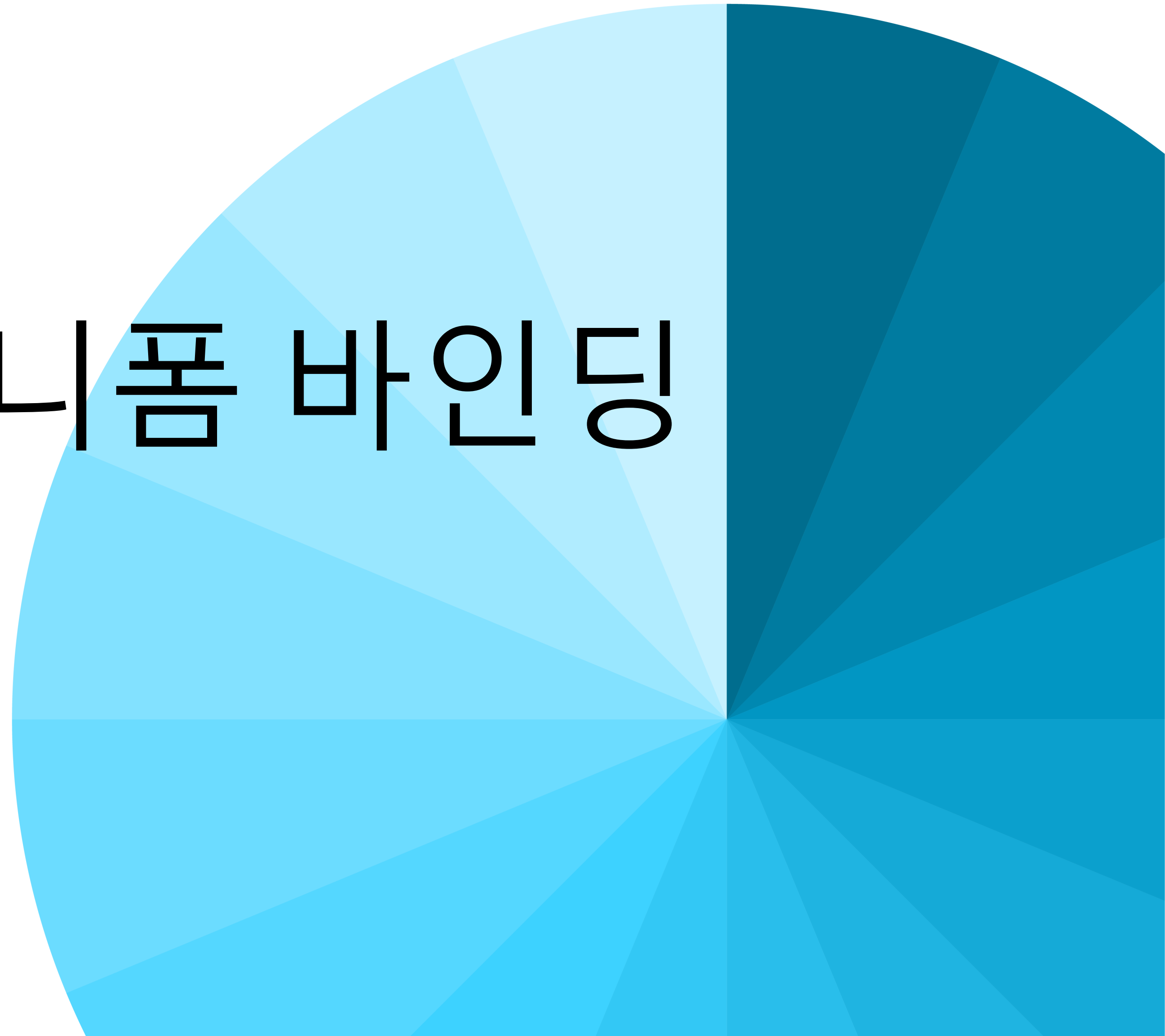
[library newFunctionWithName:@"vertexShader" constantValues:constantValues error:&error];
```

```
constant bool attribute_color_defined [[function_constant(0)]];
constant bool attribute_opacity_defined [[function_constant(1)]];

struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
     float4 color [[attribute(2), function_constant(attribute_color_defined)]];
     float opacity [[attribute(3), function_constant(attribute_opacity_defined)]];
};

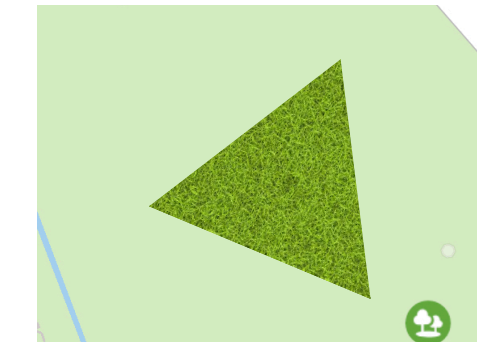
vertexShader(VertexIn in [[stage_in]])
```


3. Metal 유니폼 바인딩



3. Metal Uniform Binding

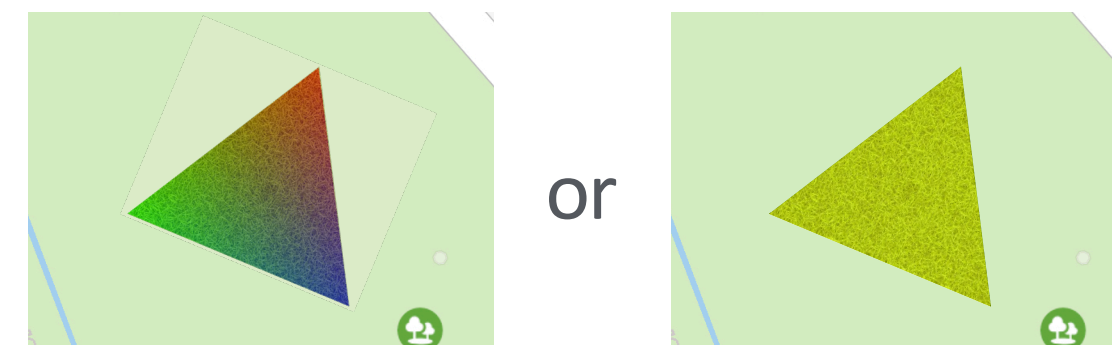
3.1 숨겨왔던 비밀, 행렬



3.2 단일 색상 추가

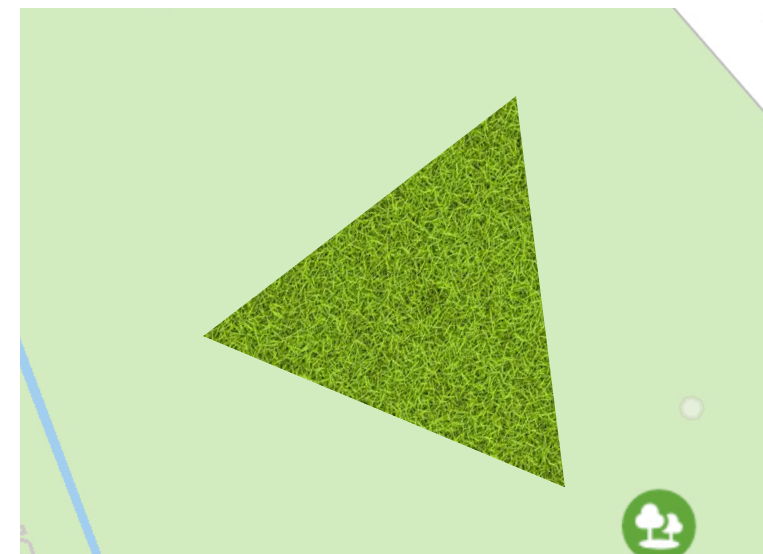


3.3 정점 당 색상 혹은 단일 색상



3.1 숨겨왔던 비밀, 행렬

행렬만 있으니 타입 그대로 매개 변수에 사용



0	(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
1	(0.5, 0.87, 0, 1, -.087, 0.5, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1)					



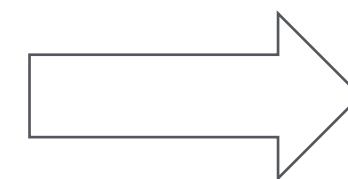
```
struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
};

vertexShader(VertexIn in [[stage_in],
    constant float4x4& matrix [[buffer(1)]]])
```


버퍼 인덱스

정점 매개 변수에 stage_in 을 사용했다면 정점 버퍼는 한 개 이상

0	(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
1	0.5		0.3		0.7	
2	(0.5, 0.87, 0, 1, -.087, 0.5, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1)					

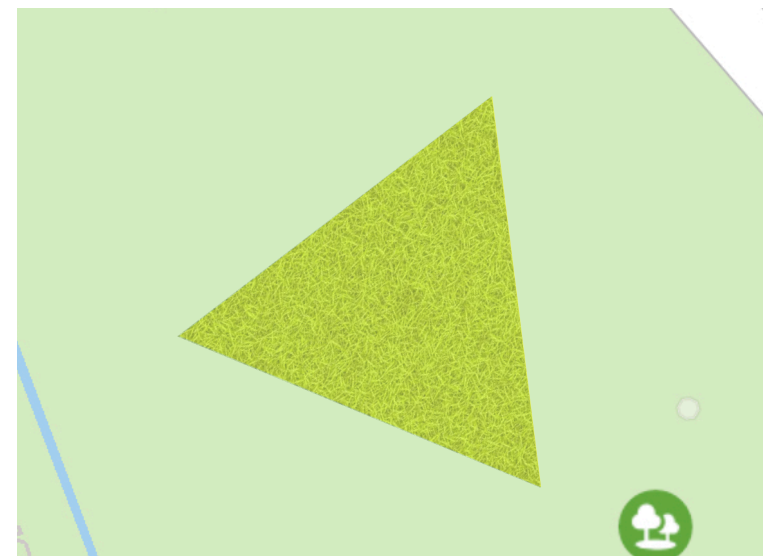


```
struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
    float opacity [[attribute(2)]];
};

0-1
vertexShader(VertexIn in [[stage_in],
    constant float4x4& matrix [[buffer(2)]]])
```

3.2 단일 색상 추가

유니폼 정보가 늘어나서 구조체로 관리



0	(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
1	(0.5, 0.87, 0, 1, -.087, 0.5, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1), (1, 1, 0, 0.5)					



```
struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
};

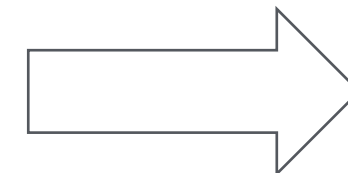
struct Uniform {
    float4x4 matrix;
    float4 color;
};

vertexShader(VertexIn in [[stage_in],
               constant Uniform& unf [[buffer(1)]]])
```


유니폼 변수 당 하나의 버퍼를 쓸 경우

잡은 setVertexBuffer 호출은 성능 저하

0	(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
1	(0.5, 0.87, 0, 1, -.087, 0.5, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1)					
2	(1, 0, 0, 1)					
3	0.5					



```
struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
};

vertexShader(VertexIn in [[stage_in]],
    constant matrix4x4& mat [[buffer(1)]],
    constant float4& color [[buffer(2)]],
    constant float& opacity [[buffer(3)])])
```

4회|의 setVertexBuffer 호출

3.3 정점 당 색상 혹은 단일 색상

셰이더 함수 안에서 Function Constant 변수를 조건으로 사용



0	(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
1	(1, 0, 0, 1)		(0, 1, 0, 1)		(0, 0, 1, 0)	
2	(0.5, 0.87, 0, 1, -.087, 0.5, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1), (1, 1, 0, 0.5)					



```
constant bool attribute_color_defined [[function_constant(0)]];
constant int vertex_uniform_index [[function_constant(1)]];

struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
    float4 color [[attribute(2), function_constant(attribute_color_defined)]];
};

struct Uniform {
    float4x4 matrix;
    float4 color;
};

vertexShader(VertexIn in [[stage_in],
               constant Uniform& unf [[buffer(vertex_uniform_index)]]]) {
    float4 color = attribute_color_defined ? in.color : unf.color;
}
```

버퍼 인덱스 with Function Constant

Function Constant 사용하는 경우 유니폼 버퍼 시작 인덱스 필요

0	(250, -250)	(1, 1)	(-250, -250)	(0, 1)	(0, 250)	(0.5, 0)
1	(1, 0, 0, 1)		(0, 1, 0, 1)		(0, 0, 1, 0)	
2	0.5		0.3		0.7	
3	(0.5, 0.87, 0, 1, -.087, 0.5, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1)), (1, 1, 0, 0.5)					



```
constant bool attribute_color_defined [[function_constant(0)]];
constant bool attribute_opacity_defined [[function_constant(1)]];
constant int vertex_uniform_index [[function_constant(2)]];

struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
    float4 color [[attribute(2), function_constant(attribute_color_defined)]];
    float opacity [[attribute(3), function_constant(attribute_opacity_defined)]];
};

vertexShader(VertexIn in [[stage_in],
                                0 or 0-1 or 0-2
                                constant Uniform& unf [[buffer(vertex_uniform_index)]]],
                                1 or 2 or 3)
```


매개 변수에 Function Constant 적용

유니폼 멤버는 낭비하지 않지만, 다른 낭비

- 매개 변수 속성으로 Function Constant 사용 시 !(not) 사용 불가

~~!function_constant(...)~~

0	...	...	...	...	...	...
1	(1, 0, 0, 1)	(0, 1, 0, 1)	(0, 0, 1, 0)			
2	0.5	0.3	0.7			
3	(0.5, 0.87, 0, 1, -0.87, 0.5, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1)					
4	(1, 0, 0, 1)					
5	0.5					



```
constant bool attribute_color_defined [[function_constant(0)]];
constant bool uniform_color_defined [[function_constant(1)]];
constant bool attribute_opacity_defined [[function_constant(2)]];
constant bool uniform_opacity_defined [[function_constant(3)]];
constant int vertex_uniform_index [[function_constant(4)]];

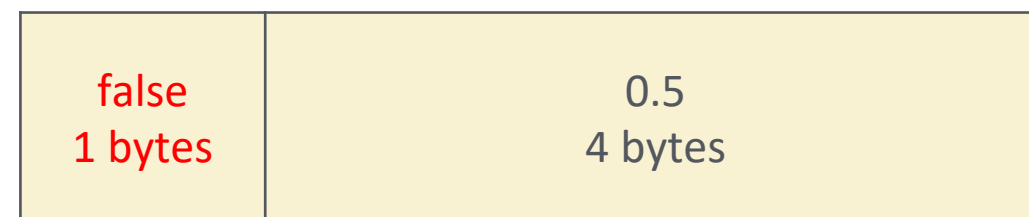
struct VertexIn {
    short2 pos [[attribute(0)]];
    float2 tex [[attribute(1)]];
    float4 color [[attribute(2), function_constant(attribute_color_defined)]];
    float opacity [[attribute(3), function_constant(attribute_opacity_defined)]];
};

vertexShader(VertexIn in [[stage_in]],
    constant matrix4x4& mat [[buffer(vertex_uniform_index)],
    constant float4& color [[buffer(vertex_uniform_index + 1), function_constant(uniform_color_defined)],
    constant float& opacity [[buffer(vertex_uniform_index + 2), function_constant(uniform_opacity_defined)]]])
```

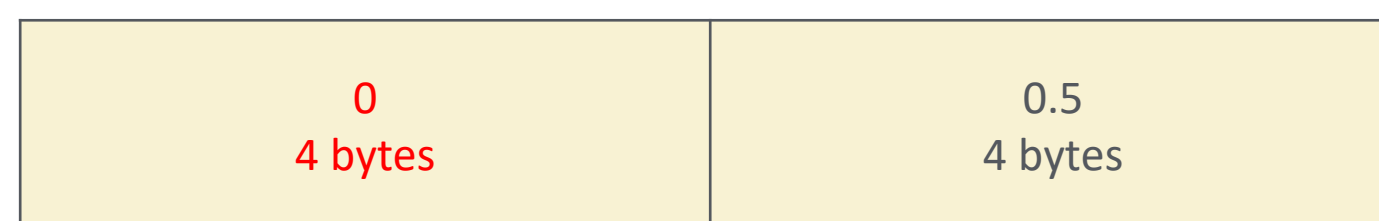
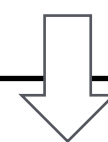
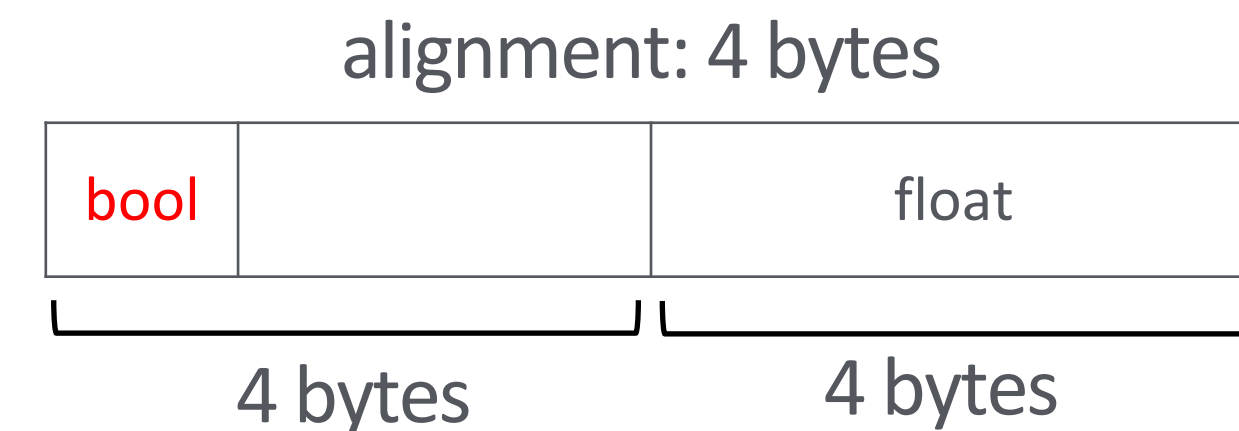
최소 4의 배수 bytes 맞추기

setVertexBuffer 사용 시 constant 주소 공간의 최소 alignment 는 4
- 멤버에 bool, short 등의 변수가 있는 경우 int 로 변경

```
vertexShader(VertexIn in [[stage_in],  
              constant Uniform& unf [[buffer(1)]])
```



```
struct Uniform {  
    bool flag;  
    float opacity;  
};
```



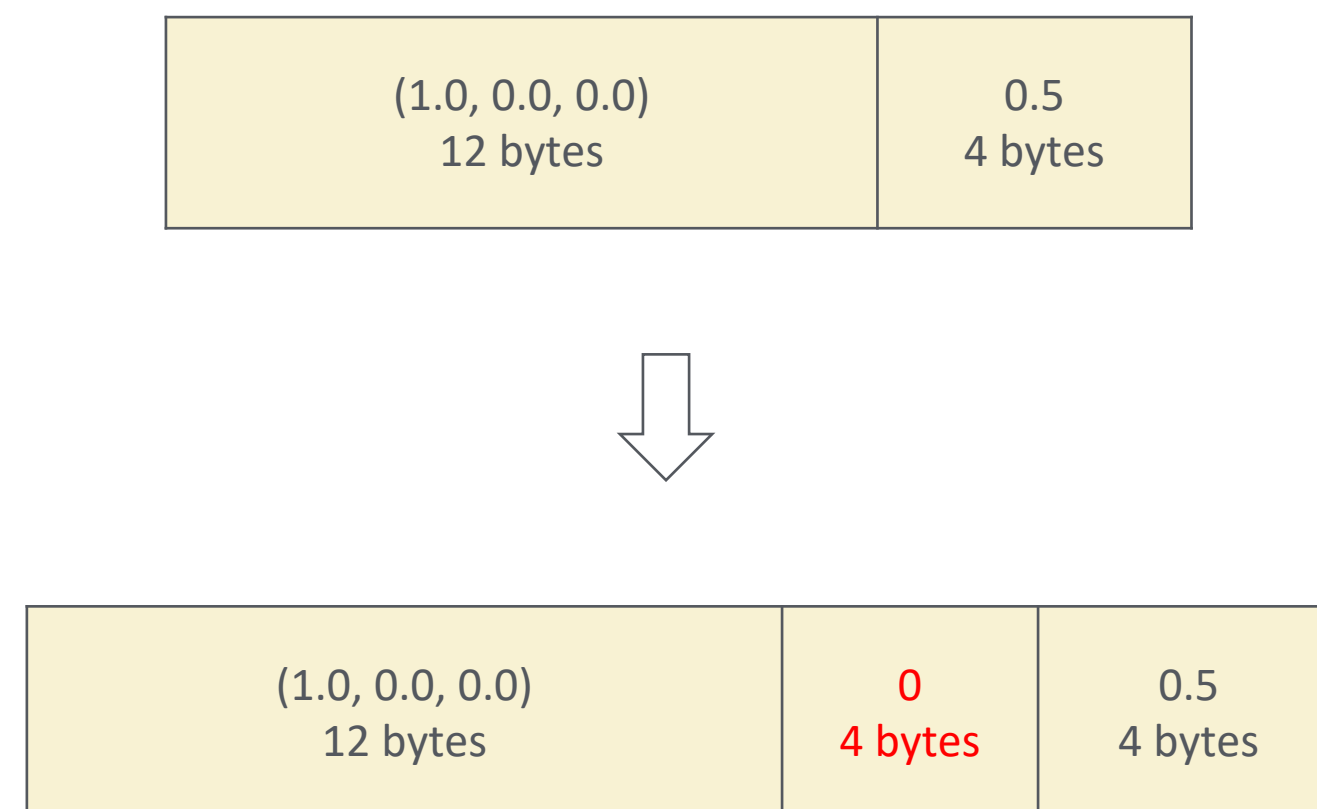
```
struct Uniform {  
    int flag;  
    float opacity;  
};
```



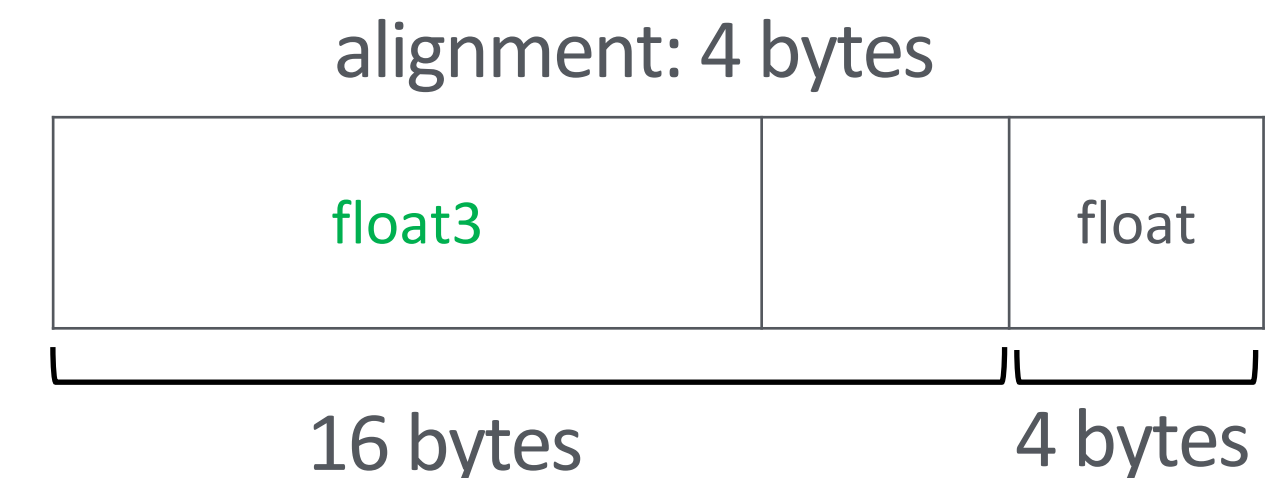
float3 타입을 써야 하는 경우 - 방법 1

float3 의 크기는 12 bytes, alignment 는 16 bytes

- 4 bytes 더미 메모리로 채움



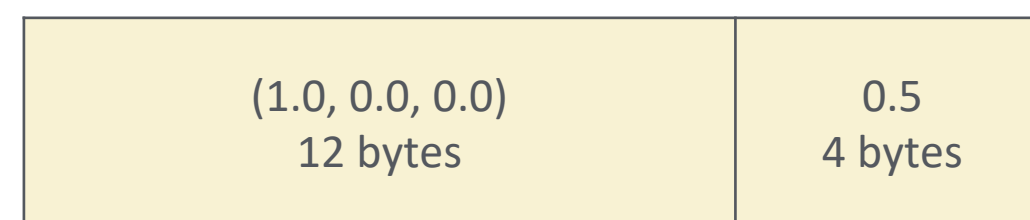
```
struct Uniform {
    float3 color;
    float opacity;
} __attribute__((packed aligned(4)));
```



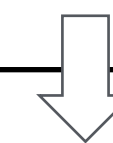
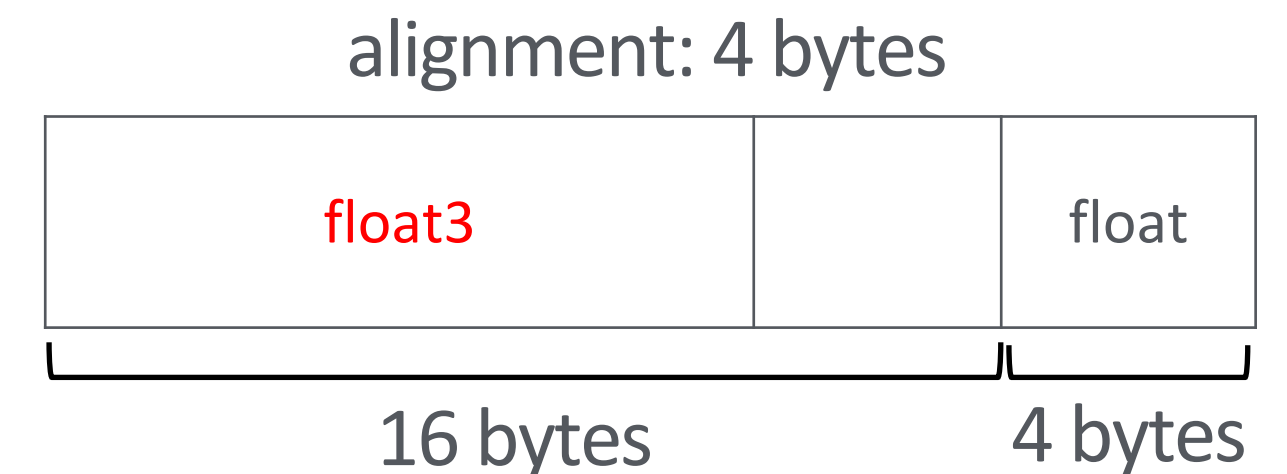
float3 타입을 써야 하는 경우 - 방법 2

float3 의 크기는 12 bytes, alignment 는 16 bytes

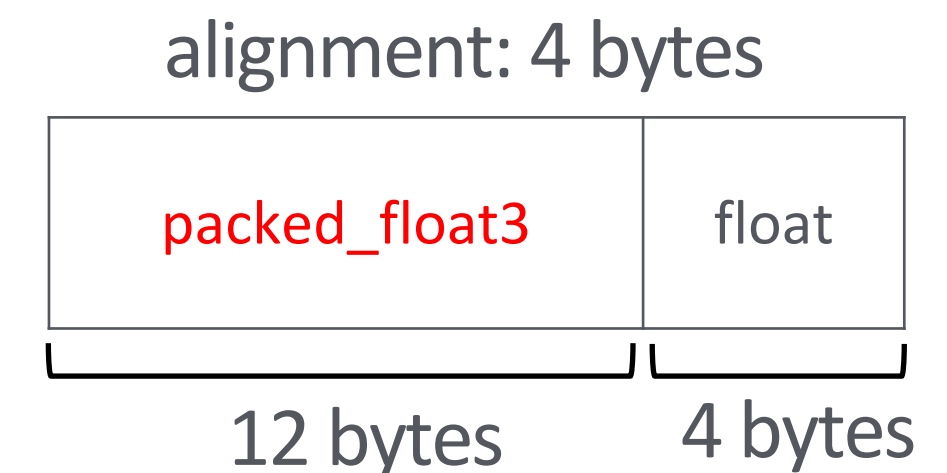
- packed_float3 로 대체 -> 크기, alignment 모두 12 bytes



```
struct Uniform {
    float3 color;
    float opacity;
} __attribute__((packed aligned(4)));
```



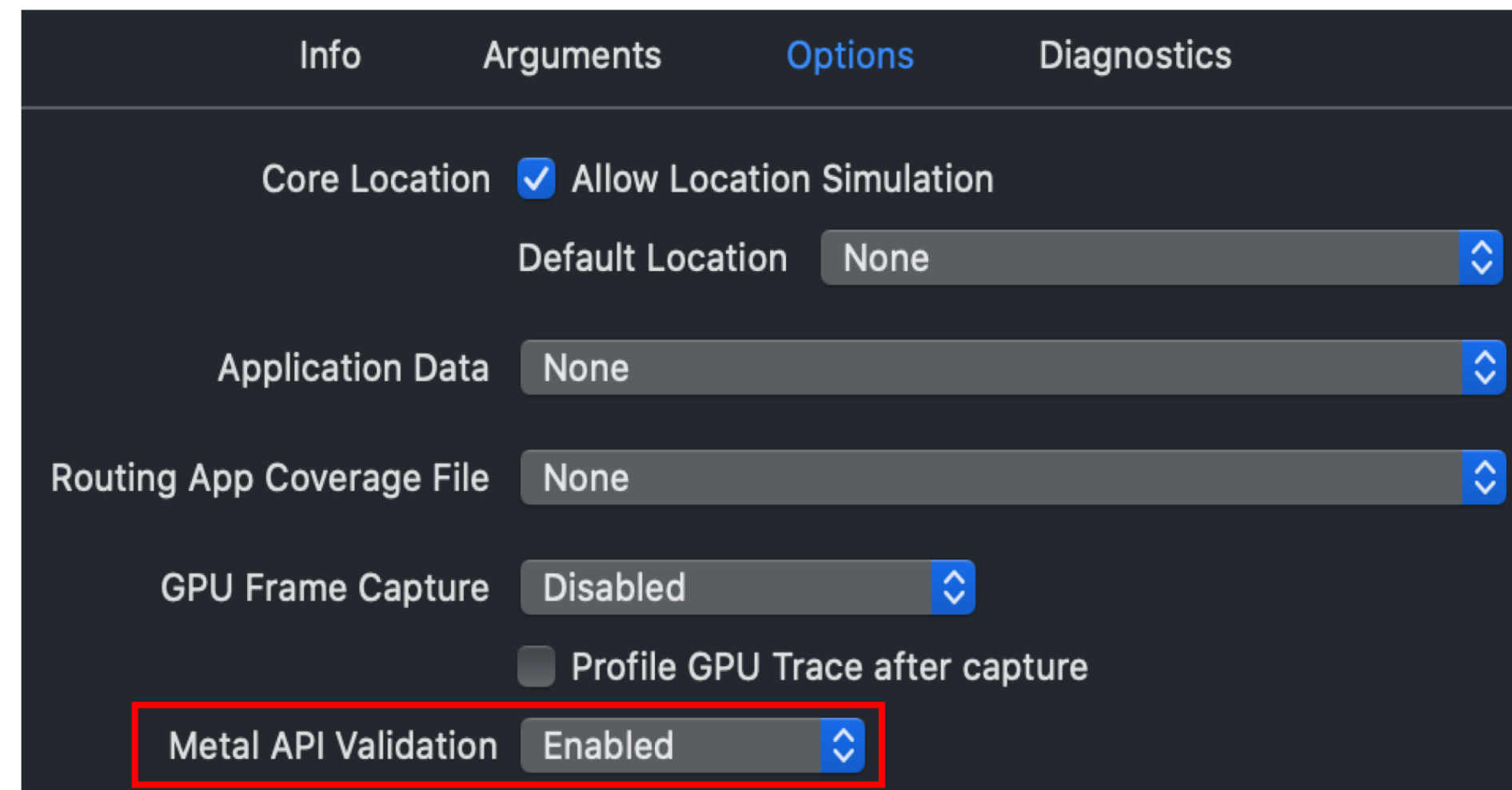
```
struct Uniform {
    packed_float3 color;
    float opacity;
} __attribute__((packed aligned(4)));
```



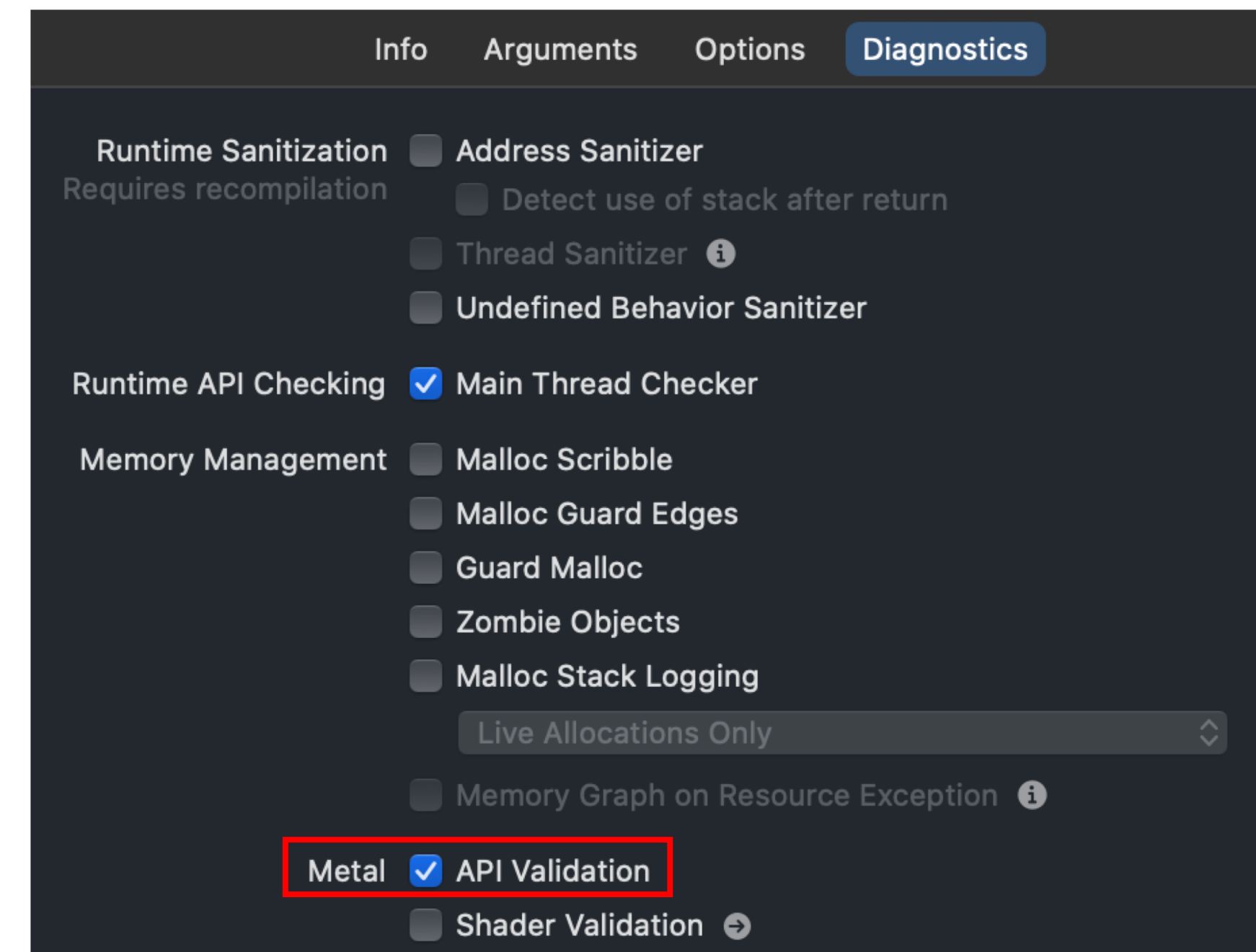
Metal API Validation

데이터 바인딩 작업 시 켜두면 유익

Xcode 11



Xcode 12



Metal API Validation 메시지

buffer 인덱스를 잘 못 지정한 경우

Disabled: 'Execution of the command buffer was aborted due to an error during execution. Caused GPU Hang Error (IOAF code 3)'

Enabled: 'Vertex Function(vertexShader): missing buffer binding at index 1 for unf[0]'

데이터의 크기와 유니폼 구조체의 크기가 다른 경우

Disabled: 이상하게 그려지거나 디버그 모드에서 문제 없이 잘 돌아가기도 함

Enabled: 'Vertex Function(vertexShader): argument unf[0] from buffer(1) with
offset(0) and length(72) has space for 72 bytes, but argument has a length(80).'

4. Vulkan 정점 데이터 바인딩

4.1 마이그레이션 (Metal to Vulkan)

세 가지 포팅

1. CPU 코드(API) 포팅

```
[encoder setVertexBuffer:positionData offset:0 atIndex:0];
[encoder setVertexBuffer:matrixData offset:0 atIndex:1];
```

```
vkCmdBindVertexBuffers(drawCmdBuffer, 0,
    positionData.size(), positionData.data(),
    offsets.data());
vkCmdBindDescriptorSets(drawCmdBuffer,
    VK_PIPELINE_BIND_POINT_GRAPHICS,
    pipelineLayout, 0, 1, &matrixDataDescriptorSet,
    dynamicOffsets.size(), dynamicOffsets.data());
```

3. 데이터 바인딩 방식 포팅

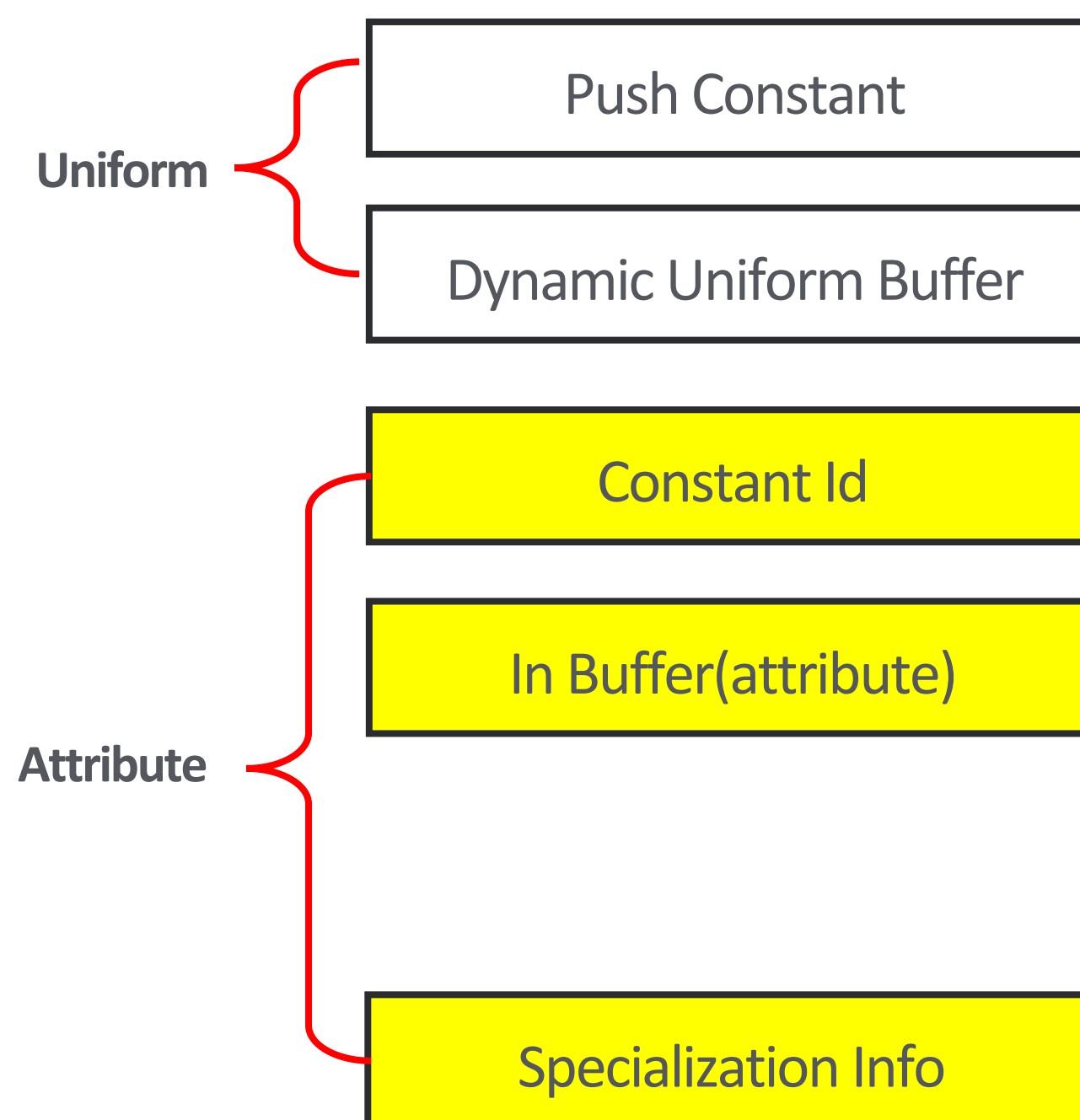
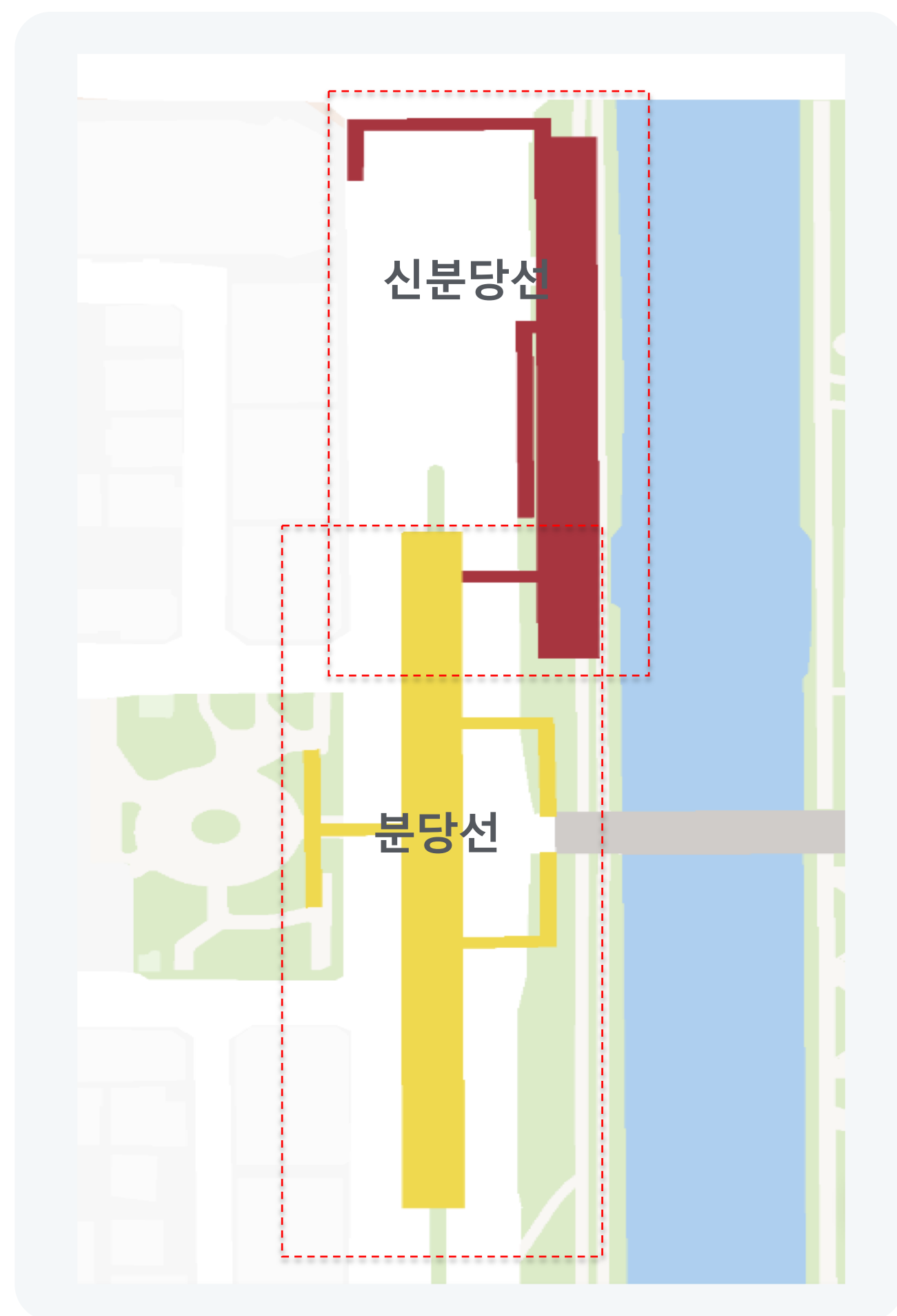
2. GPU 코드(셰이더) 포팅

```
attribute vec2 a_position;
uniform mat4 u_matrix;
```

```
layout (location = 0) in ivec2 a_pos;
layout (binding = 0) uniform UniformBufferObject {
    mat4 matrix;
} ubo;
layout (constant_id = 0) const int attribute_opacity_defined = 0;
```


4.2 데이터 바인딩 사례

바닥 면형 (Vertex Shader)



vertex shader

```

layout (push_constant) uniform PushConsts {
    mat4 matrix;
} pushConsts;

layout (binding = 0) uniform UniformBufferObject {
    vec4 color;
    float opacity;
} ubo;

layout (constant_id = 0) const int attribute_color_defined = 0;
layout (constant_id = 1) const int attribute_opacity_defined = 0;

layout (location = 0) in ivec2 a_pos;
layout (location = 1) in vec4 a_color;
layout (location = 2) in float a_opacity;

layout (location = 0) out vec4 v_color;
layout (location = 1) out float v_opacity;

out gl_PerVertex {
    vec4 gl_Position;
};

void main() {
    v_color = (attribute_color_defined == 1) ? a_color : ubo.color;
    v_opacity = (attribute_opacity_defined == 1) ? a_opacity : ubo.opacity;

    gl_Position = pushConsts.matrix * vec4(a_pos, 0, 1);
}
    
```

4.3 Vertex Input Description

정점 정보(위치, 색상, 투명도) 업로드

VkVertexInputBindingDescription

Attributes

0 (a_pos)	format	short2
	buffer	0
	offset	0
1 (a_color)	format	float4
	buffer	1
2 (a_opacity)	format	float
	buffer	2
	offset	0

Layouts

0	stride	4
1	stride	16
2	stride	4

Buffer List

(250, -250) 4 bytes	(-250, -250) 4 bytes	...
(1.0, 1.0, 0.0, 1.0) 16 bytes	(1.0, 1.0, 0.0, 1.0) 16 bytes	...
(1.0) 4bytes	(1.0) 4bytes	...

vertex shader

```

layout (push_constant) uniform PushConsts {
    mat4 matrix;
} pushConsts;

layout (binding = 0) uniform UniformBufferObject {
    vec4 color;
    float opacity;
} ubo;

layout (constant_id = 0) const int attribute_color_defined = 0;
layout (constant_id = 1) const int attribute_opacity_defined = 0;

layout (location = 0) in ivec2 a_pos;
layout (location = 1) in vec4 a_color;
layout (location = 2) in float a_opacity;

layout (location = 0) out vec4 v_color;
layout (location = 1) out float v_opacity;

out gl_PerVertex {
    vec4 gl_Position;
};

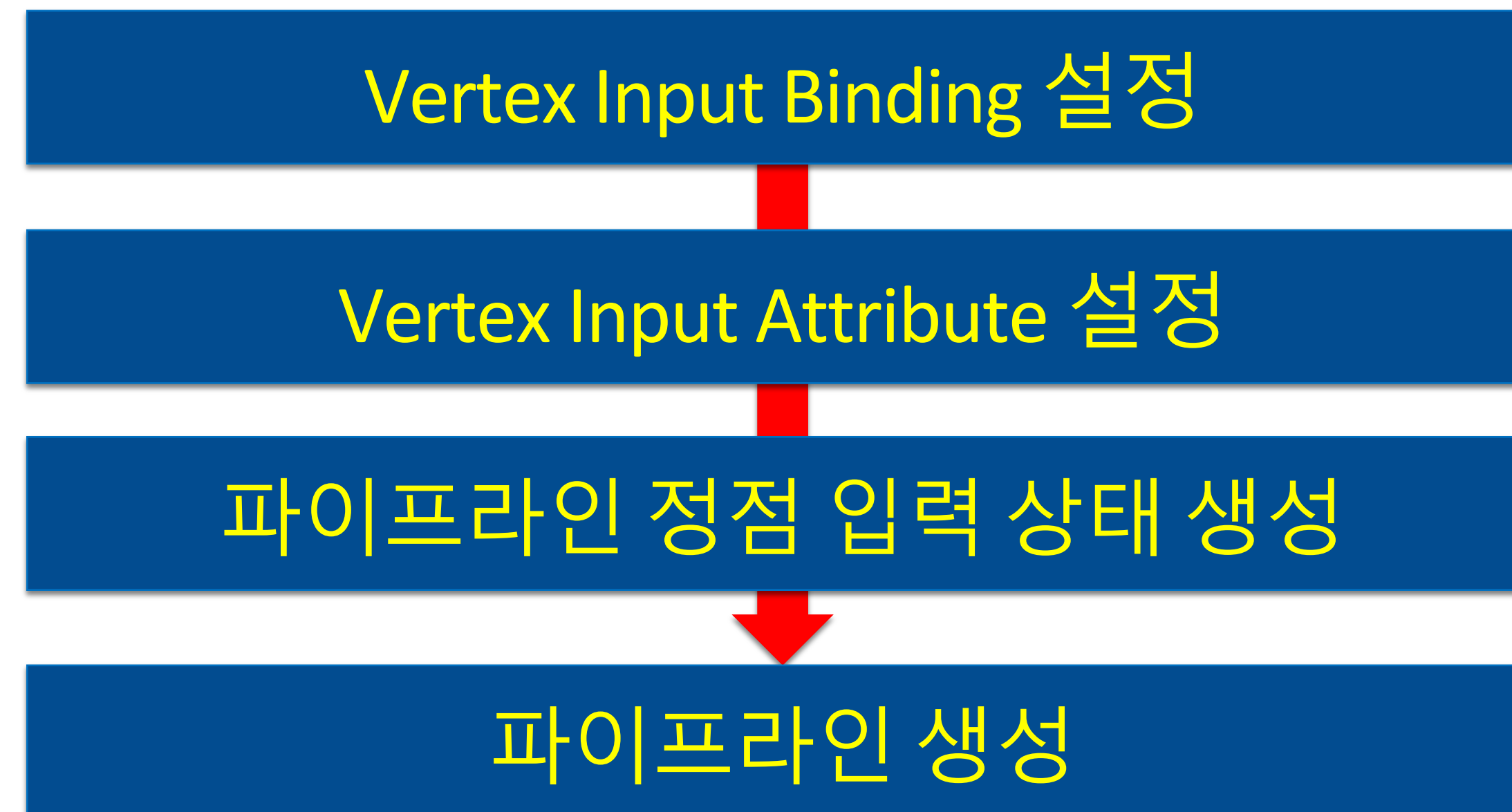
void main() {
    v_color = (attribute_color_defined == 1) ? a_color : ubo.color;
    v_opacity = (attribute_opacity_defined == 1) ? a_opacity : ubo.opacity;

    gl_Position = pushConsts.matrix * vec4(a_pos, 0, 1);
}

```

4.3 Vertex Input Description

Attribute 상태는 어떻게 바인딩하지?



4.3 Vertex Input Description

Vertex Input Binding 설정

- VkVertexInputBindingDescription

- [0] = {VkVertexInputBindingDescription}
binding = {uint32_t} 0
stride = {uint32_t} 4
inputRate = {VkVertexInputRate} VK_VERTEX_INPUT_RATE_VERTEX
- [1] = {VkVertexInputBindingDescription}
binding = {uint32_t} 1
stride = {uint32_t} 16
inputRate = {VkVertexInputRate} VK_VERTEX_INPUT_RATE_VERTEX
- [2] = {VkVertexInputBindingDescription}
binding = {uint32_t} 2
stride = {uint32_t} 4
inputRate = {VkVertexInputRate} VK_VERTEX_INPUT_RATE_VERTEX

Create Vertex Input Description

```
inline VkVertexInputBindingDescription vertexInputBindingDescription(  
    uint32_t binding,  
    uint32_t stride,  
    VkVertexInputRate inputRate) {  
    VkVertexInputBindingDescription inputBindingDescription{};  
    inputBindingDescription.binding = binding;  
    inputBindingDescription.stride = stride;  
    inputBindingDescription.inputRate = inputRate;  
    return inputBindingDescription;  
}
```

Buffer List

0	(250, -250) 4 bytes	(-250, -250) 4 bytes	...
1	(1.0, 1.0, 0.0, 1.0) 16 bytes	(1.0, 1.0, 0.0, 1.0) 16 bytes	...
2	(1.0) 4bytes	(1.0) 4bytes	...

4.3 Vertex Input Description

Vertex Input Attribute 설정

- VkVertexInputAttributeDescription

[0] = {VkVertexInputAttributeDescription}
location = {uint32_t} 0
binding = {uint32_t} 0
format = {VkFormat} VK_FORMAT_R16G16_SINT
offset = {uint32_t} 0

[1] = {VkVertexInputAttributeDescription}
location = {uint32_t} 1
binding = {uint32_t} 1
format = {VkFormat} VK_FORMAT_R32G32_SFLOAT
offset = {uint32_t} 0

[2] = {VkVertexInputAttributeDescription}
location = {uint32_t} 2
binding = {uint32_t} 2
format = {VkFormat} VK_FORMAT_R32G32_SFLOAT
offset = {uint32_t} 0

Create Vertex Input Description

```
inline VkVertexInputAttributeDescription vertexInputAttributeDescription(  
    uint32_t binding,  
    uint32_t location,  
    VkFormat format,  
    uint32_t offset) {  
    VkVertexInputAttributeDescription inputAttributeDescription;  
    inputAttributeDescription.binding = binding;  
    inputAttributeDescription.location = location;  
    inputAttributeDescription.format = format;  
    inputAttributeDescription.offset = offset;  
    return inputAttributeDescription;  
}
```

Attributes

0	format	short2
	buffer	0
	offset	0
1	format	float4
	buffer	1
	offset	0
2	format	float
	buffer	2
	offset	0

4.3 Vertex Input Description

파이프라인 정점 입력 상태 생성

- VkPipelineVertexInputStateCreateInfo

```
VkPipelineVertexInputStateCreateInfo vertexInputStateCreateInfo{  
    .sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO,  
    .vertexBindingDescriptionCount = vertexInputBindingDescriptions.size(),  
    .pVertexBindingDescriptions = vertexInputBindingDescriptions.data(),  
    .vertexAttributeDescriptionCount = vertexInputAttributeDescriptions.size(),  
    .pVertexAttributeDescriptions = vertexInputAttributeDescriptions.data(),  
}
```

```
VkGraphicsPipelineCreateInfo pipelineCreateInfo{  
    .sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO,  
    .pVertexInputState = &vertexInputStateCreateInfo,  
};
```

```
vkCreateGraphicsPipelines(device, pipelineCache, 1, &pipelineCreateInfo, nullptr, &pipeline);
```

4.4 Specialization Constant

OpenGL 코드 이식 시 문제점

- 전처리기 명령어 (#ifdef ~ #else ~ #endif) 대체

```
#if defined(HAS_UNIFORM_u_color)
uniform highp vec4 u_color;
#else
attribute highp vec4 a_color;
varying highp vec4 color;
#endif
```

vertex shader

```
layout (push_constant) uniform PushConsts {
    mat4 matrix;
} pushConsts;

layout (binding = 0) uniform UniformBufferObject {
    vec4 color;
    float opacity;
} ubo;

layout (constant_id = 0) const int attribute_color_defined = 0;
layout (constant_id = 1) const int attribute_opacity_defined = 0;

layout (location = 0) in ivec2 a_pos;
layout (location = 1) in vec4 a_color;
layout (location = 2) in float a_opacity;

layout (location = 0) out vec4 v_color;
layout (location = 1) out float v_opacity;

out gl_PerVertex {
    vec4 gl_Position;
};

void main() {
    v_color = (attribute_color_defined == 1) ? a_color : ubo.color;
    v_opacity = (attribute_opacity_defined == 1) ? a_opacity : ubo.opacity;

    gl_Position = pushConsts.matrix * vec4(a_pos, 0, 1);
}
```

constant_id

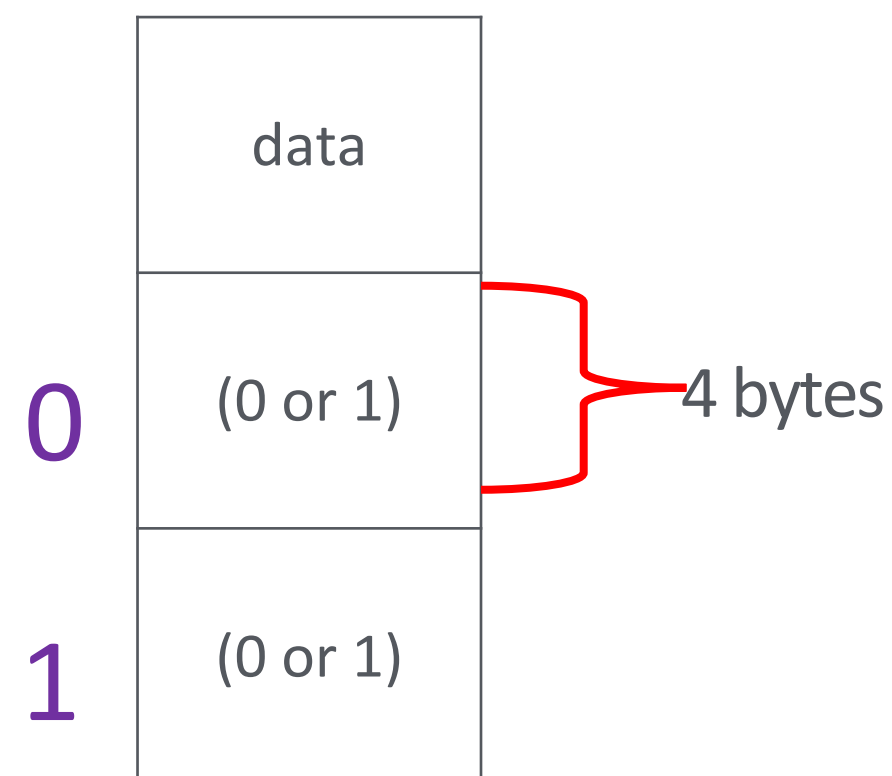
attribute or uniform data

4.4 Specialization Constant

Shader 함수 컴파일시 입력할 수 있는 특별한 상수 생성

- Specialization Data

```
std::vector<uint32_t> specializationData;  
specializationData.push_back(colorEnabled ? 1 : 0);  
specializationData.push_back(opacityEnabled ? 1 : 0);
```



```
vertex shader

layout (push_constant) uniform PushConsts {
    mat4 matrix;
} pushConsts;

layout (binding = 0) uniform UniformBufferObject {
    vec4 color;
    float opacity;
} ubo;

layout (constant_id = 0) const int attribute_color_defined = 0;
layout (constant_id = 1) const int attribute_opacity_defined = 0;

layout (location = 0) in ivec2 a_pos;
layout (location = 1) in vec4 a_color;
layout (location = 2) in float a_opacity;

layout (location = 0) out vec4 v_color;
layout (location = 1) out float v_opacity;

out gl_PerVertex {
    vec4 gl_Position;
};

void main() {
    v_color = (attribute_color_defined == 1) ? a_color : ubo.color;
    v_opacity = (attribute_opacity_defined == 1) ? a_opacity : ubo.opacity;

    gl_Position = pushConsts.matrix * vec4(a_pos, 0, 1);
}
```

constant_id

attribute or uniform data

4.4 Specialization Constant

Map Entry 생성

- VkSpecializationMapEntry

```
std::vector<VkSpecializationMapEntry> specializationMapEntries;
specializationMapEntries.push_back(
    initializers::specializationMapEntry(0, 0, sizeof(uint32_t)));
specializationMapEntries.push_back(
    initializers::specializationMapEntry(1, 4, sizeof(uint32_t)));
```

constant id	offset	size
0	0	4 (sizeof(uint32_t))
1	4	4 (sizeof(uint32_t))

Specialization constant

```
VkSpecializationMapEntry specializationMapEntry(uint32_t constantID, uint32_t offset, size_t size) {
    VkSpecializationMapEntry specializationMapEntry{};
    specializationMapEntry.constantID = constantID;
    specializationMapEntry.offset = offset;
    specializationMapEntry.size = size;
    return specializationMapEntry;
}

VkSpecializationInfo
specializationInfo(uint32_t mapEntryCount, const VkSpecializationMapEntry* mapEntries,
    size_t dataSize, const void* data) {
    VkSpecializationInfo specializationInfo{};
    specializationInfo.mapEntryCount = mapEntryCount;
    specializationInfo.pMapEntries = mapEntries;
    specializationInfo.dataSize = dataSize;
    specializationInfo.pData = data;
    return specializationInfo;
}
```

4.4 Specialization Constant

Specialization 정보 생성

- VkSpecializationInfo

```
VkSpecializationInfo specializationInfo = initializers::specializationInfo(
    static_cast<uint32_t>(specializationMapEntries.size()),
    specializationMapEntries.data(),
    sizeof(specializationData),
    specializationData.data());
```

Specialization constant

```
VkSpecializationMapEntry specializationMapEntry(uint32_t constantID, uint32_t offset, size_t size) {
    VkSpecializationMapEntry specializationMapEntry{};
    specializationMapEntry.constantID = constantID;
    specializationMapEntry.offset = offset;
    specializationMapEntry.size = size;
    return specializationMapEntry;
}
```

```
VkSpecializationInfo
specializationInfo(uint32_t mapEntryCount, const VkSpecializationMapEntry* mapEntries,
    size_t dataSize, const void* data) {
    VkSpecializationInfo specializationInfo{};
    specializationInfo.mapEntryCount = mapEntryCount;
    specializationInfo.pMapEntries = mapEntries;
    specializationInfo.dataSize = dataSize;
    specializationInfo.pData = data;
    return specializationInfo;
}
```


4.4 Specialization Constant

파이프라인에 Specialization 정보 등록

- VkPipelineShaderStageCreateInfo

```
VkPipelineShaderStageCreateInfo shaderStageCreateInfo{};  
shaderStageCreateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;  
shaderStageCreateInfo.pSpecializationInfo = &specializationInfo;  
shaderStageCreateInfo.stage = VK_SHADER_STAGE_VERTEX_BIT;  
shaderStageCreateInfo.module = vertexShader;
```

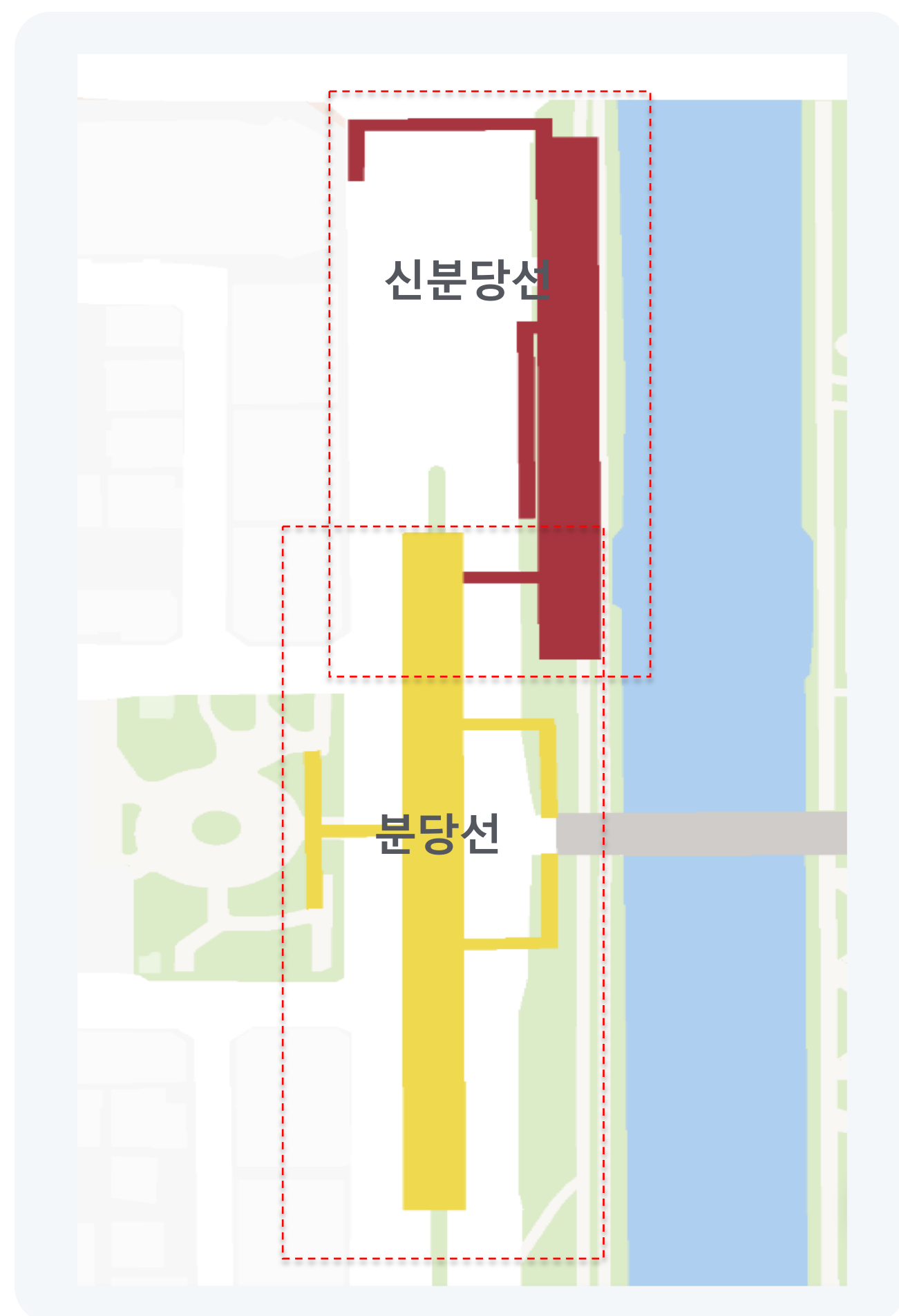
```
VkGraphicsPipelineCreateInfo pipelineCreateInfo{};  
pipelineCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;  
pipelineCreateInfo.pVertexInputState = &vertexInputStateCreateInfo;  
pipelineCreateInfo.pStages = &shaderStageCreateInfo;
```

```
vkCreateGraphicsPipelines(device, pipelineCache, 1, &pipelineCreateInfo, nullptr, &pipeline);
```

5. Vulkan 유니폼 데이터 바인딩

5.1 Uniform Buffer

바닥 면형 그리기(Vertex Shader)



Dynamic Uniform Buffer

Memory Alignment

Push Constant

vertex shader

```
layout (push_constant) uniform PushConsts {
    mat4 matrix;
} pushConsts;

layout (binding = 0) uniform UniformBufferObject {
    vec4 color;
    float opacity;
} ubo;

layout (constant_id = 0) const int attribute_color_defined = 0;
layout (constant_id = 1) const int attribute_opacity_defined = 0;

layout (location = 0) in ivec2 a_pos;
layout (location = 1) in vec4 a_color;
layout (location = 2) in float a_opacity;

layout (location = 0) out vec4 v_color;
layout (location = 1) out float v_opacity;

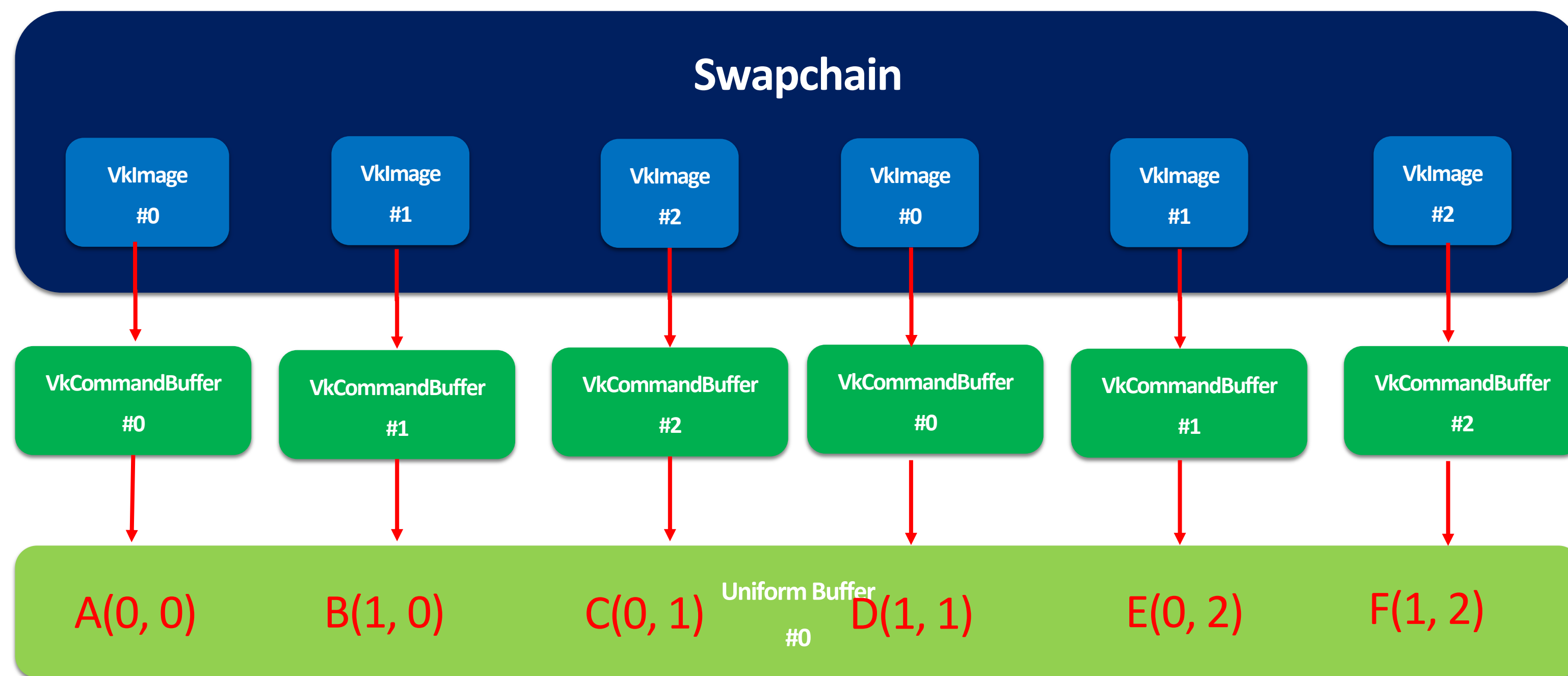
out gl_PerVertex {
    vec4 gl_Position;
};

void main() {
    v_color = (attribute_color_defined == 1) ? a_color : ubo.color;
    v_opacity = (attribute_opacity_defined == 1) ? a_opacity : ubo.opacity;

    gl_Position = pushConsts.matrix * vec4(a_pos, 0, 1);
}
```

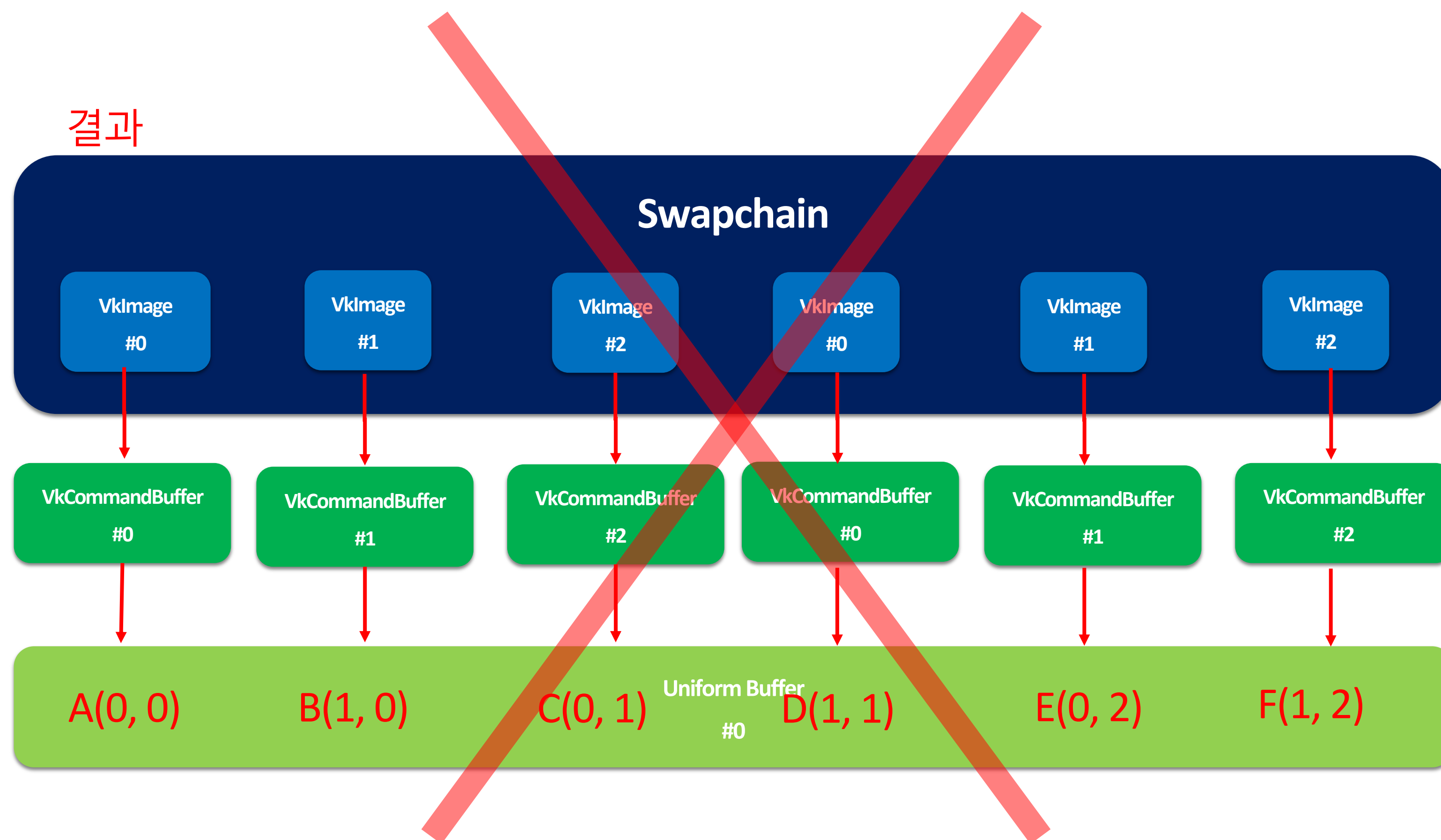
5.1 Uniform Buffer

Command Buffer에 개별 Uniform 업로드



5.1 Uniform Buffer

Command Buffer에 개별 Uniform 업로드시 문제점

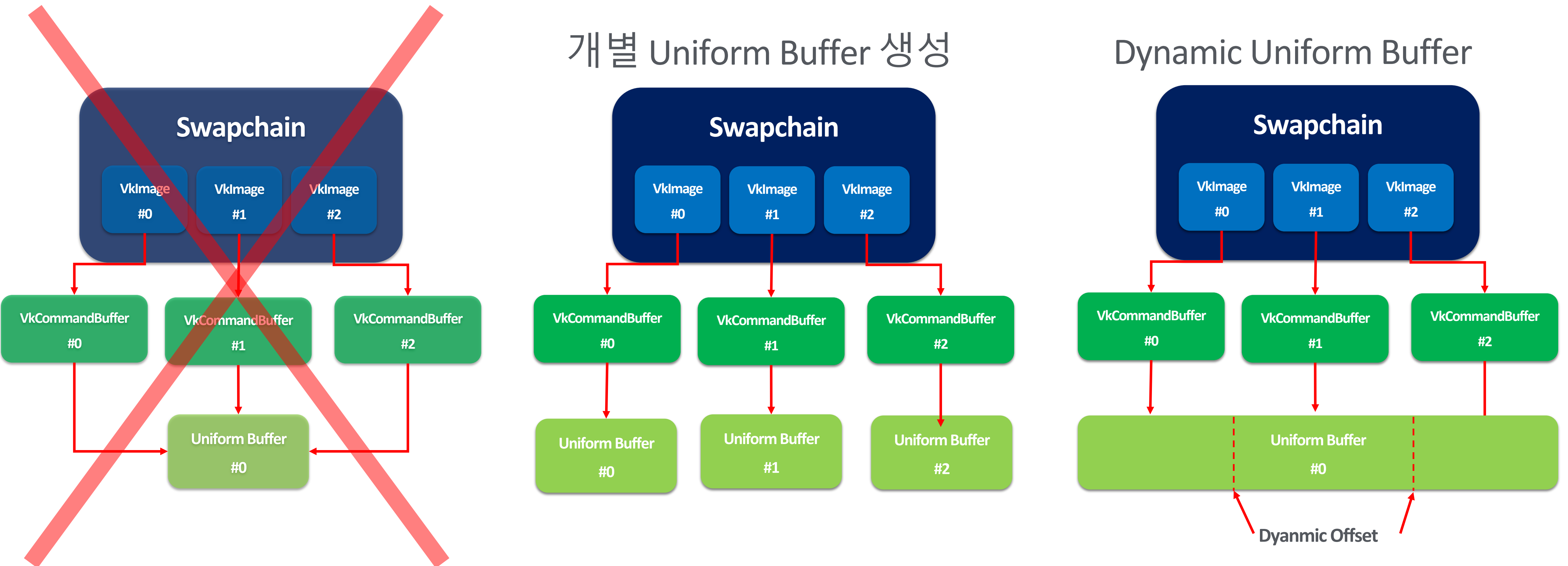


5.2 Dynamic Uniform Buffer

유니폼 버퍼를 어떻게 처리해야 하는가?

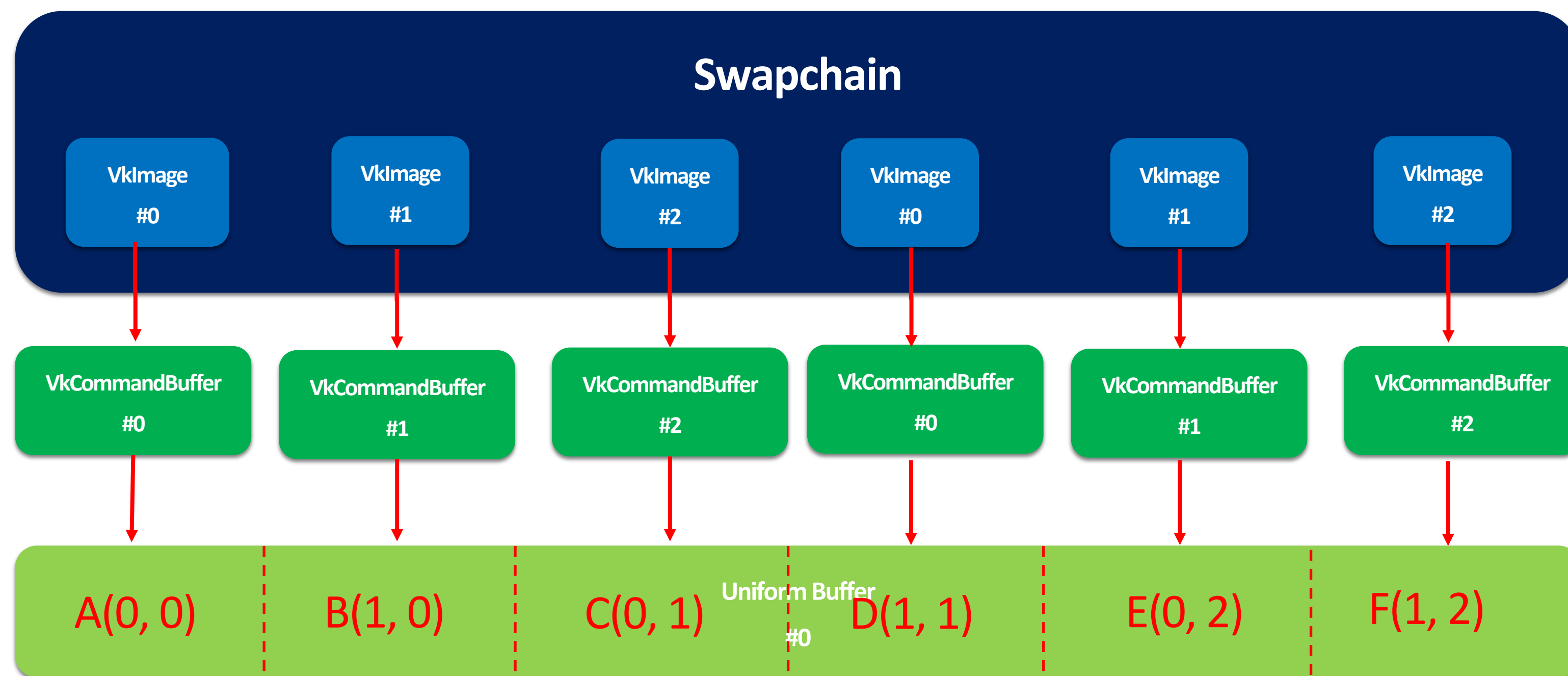
개별 Uniform Buffer 생성

Dynamic Uniform Buffer



5.2 Dynamic Uniform Buffer

Dynamic Uniform Buffer 적용



5.2 Dynamic Uniform Buffer

Dynamic Uniform Buffer는 어떻게 만드는가?



5.2 Dynamic Uniform Buffer

Binding Descriptor Set Layout 설정

- VkDescriptorSetLayoutBinding

```
std::vector<VkDescriptorSetLayoutBinding> descriptorSetLayoutBindings = {
    VkDescriptorSetLayoutBinding{
        .binding = 0,
        .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_VERTEX_BIT,
        .pImmutableSamplers = nullptr,
    },
    VkDescriptorSetLayoutBinding{
        .binding = 1,
        .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
        .pImmutableSamplers = nullptr,
    },
};
```

Dynamic Uniform Buffer

```
layout (binding = 0) uniform UniformBufferObject {
    vec3 units_to_pixels;
    vec2 patternscale_a;
    vec2 patternscale_b;
    float ratio;
    float tex_y_a;
    float tex_y_b;
    float opacity_t;
    float color_t;
    float width_t;
    float floorwidth_t;
    float gapwidth_t;
    float offset_t;
    float blur_t;
} ubo;
```

Vertex Shader

```
layout (binding = 1) uniform UniformBufferObject {
    vec4 color;
    float mix;
    float sdfgamma;
    float opacity;
    float blur;
} ubo;
```

Fragment Shader

5.2 Dynamic Uniform Buffer

Create Descriptor Set Layout 생성

- VkDescriptorSetLayoutCreateInfo

```
const VkDescriptorSetLayoutCreateInfo descriptorSetLayoutCreateInfo = {  
    .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO,  
    .pNext = nullptr,  
    .bindingCount = descriptorSetLayoutBindings.size(),  
    .pBindings = descriptorSetLayoutBindings.data(),  
};
```

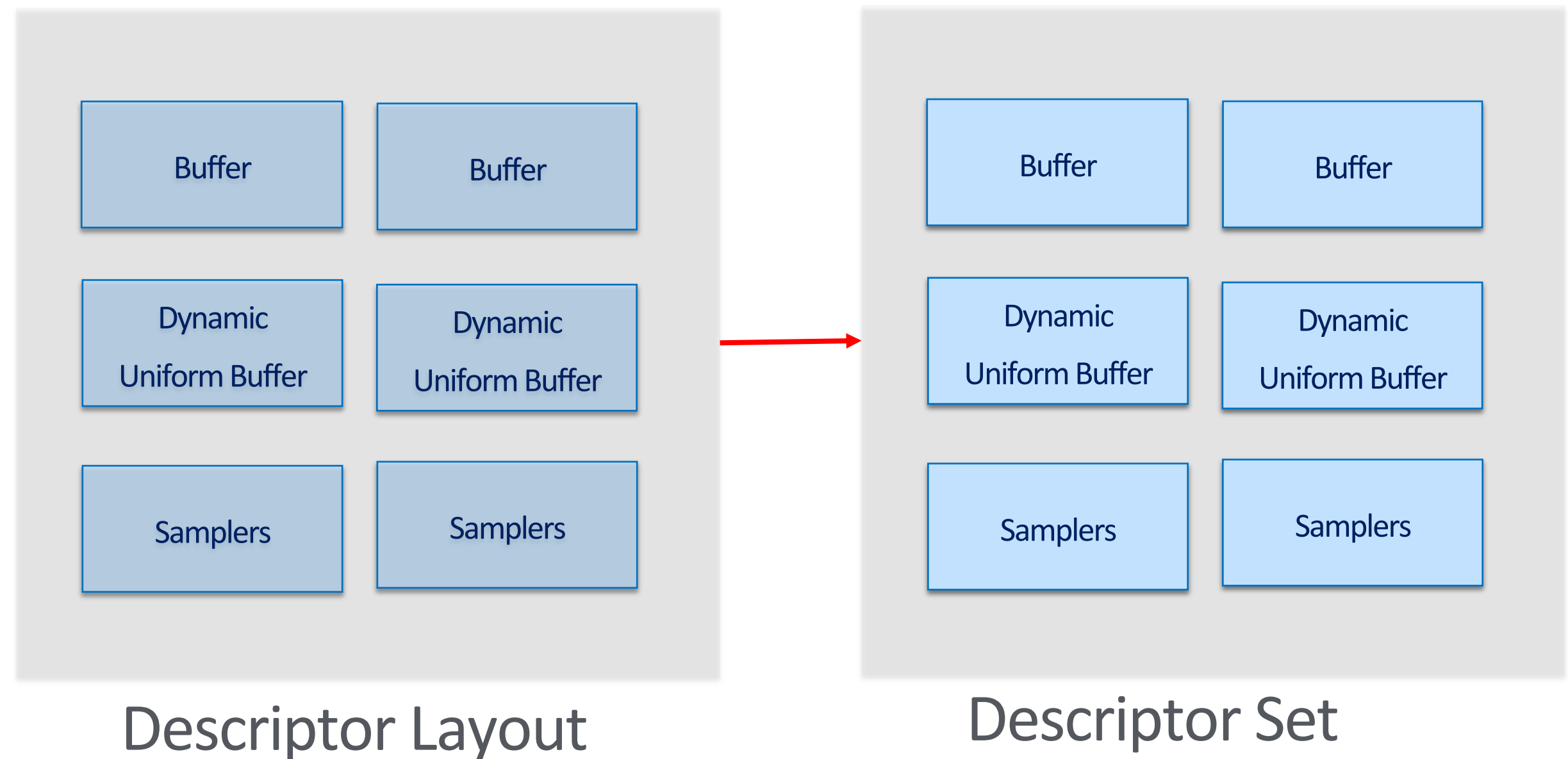
```
vkCreateDescriptorSetLayout(device,  
    &descriptorSetLayoutCreateInfo,  
    nullptr,  
    &pipelineInfo->descriptorLayout);
```

5.2 Dynamic Uniform Buffer

Create Descriptor Set 생성

- VkDescriptorSet

```
VkDescriptorSetAllocateInfo allocateInfo{
    .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO,
    .descriptorSetCount = 1,
    .pSetLayouts = &descriptorSetLayout,
    .descriptorPool = descriptorPool
};
VkDescriptorSet descriptorSet = VK_NULL_HANDLE;
vkAllocateDescriptorSets(device, &allocateInfo, &descriptorSet);
```



5.2 Dynamic Uniform Buffer

Write Descriptor Set 설정

- Vertex Shader 바인딩

```
std::vector<VkWriteDescriptorSet> writeDescriptorSets;
// Binding 0 : Vertex shader uniform buffer
writeDescriptorSets.push_back({
    .sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET,
    .pNext = nullptr,
    .dstSet = descriptorSet,
    .descriptorCount = 1,
    .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC,
    .pBufferInfo = &uboV.bufferInfo,
    .dstArrayElement = 0,
    .dstBinding = 0
});
```

Dynamic Uniform Buffer

```
layout (binding = 0) uniform UniformBufferObject {
    vec2 units_to_pixels;
    vec2 patternscale_a;
    vec2 patternscale_b;
    float ratio;
    float tex_y_a;
    float tex_y_b;
    float opacity_t;
    float color_t;
    float width_t;
    float floorwidth_t;
    float gapwidth_t;
    float offset_t;
    float blur_t;
} ubo;
```

Vertex Shader

64 bytes

```
layout (binding = 1) uniform UniformBufferObject {
    vec4 color;
    float mix;
    float sdfgamma;
    float opacity;
    float blur;
} ubo;
```

Fragment Shader

5.2 Dynamic Uniform Buffer

Write Descriptor Set 설정

- Fragment Shader 바인딩

// Binding 1 : Fragment shader uniform buffer

```
writeDescriptorSets.push_back({
    .sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET,
    .pNext = nullptr,
    .dstSet = descriptorSet,
    .descriptorCount = 1,
    .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC,
    .pBufferInfo = &uboF.bufferInfo,
    .dstArrayElement = 0,
    .dstBinding = 1
});
```

Dynamic Uniform Buffer

Vertex Shader

```
layout (binding = 0) uniform UniformBufferObject {
    vec2 units_to_pixels;
    vec2 patternscale_a;
    vec2 patternscale_b;
    float ratio;
    float tex_y_a;
    float tex_y_b;
    float opacity_t;
    float color_t;
    float width_t;
    float floorwidth_t;
    float gapwidth_t;
    float offset_t;
    float blur_t;
} ubo;
```

Fragment Shader

```
layout (binding = 1) uniform UniformBufferObject {
    vec4 color;
    float mix;
    float sdfgamma;
    float opacity;
    float blur;
} ubo;
```

32 bytes

5.2 Dynamic Uniform Buffer

Descriptor Sets를 업데이트하고 Drawing

```
// Update Write Descriptor Sets
vkUpdateDescriptorSets(device,
    writeDescriptorSets.size(),
    writeDescriptorSets.data(),
    0,
    nullptr);

vkCmdBindDescriptorSets(
    drawCmdBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
    pipelineInfo->pipelineLayout,
    0,
    1,
    &descriptorSet,
    dynamicOffsets.size(),
    dynamicOffsets.data());
```

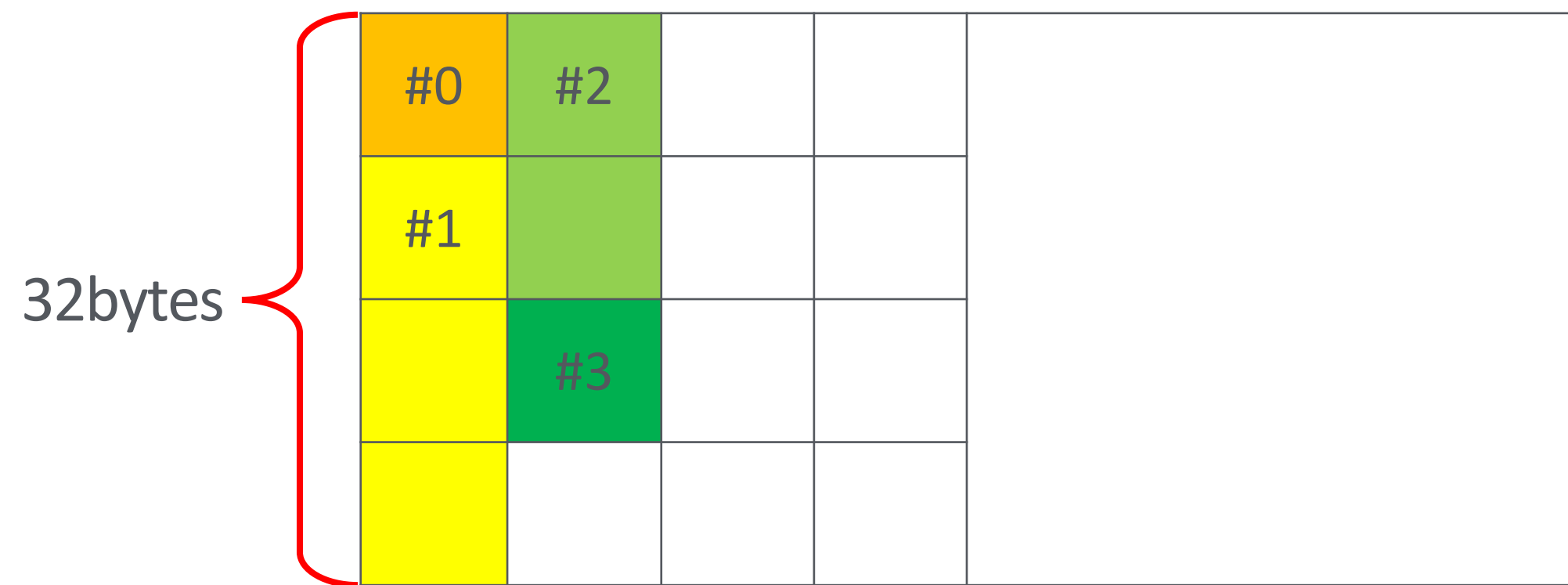
Dynamic offsets

		Offset #0	Offset #1	Offset #2	Offset #3	...
0	Vertex Shader	0	64	128	172	...
1	Fragment Shader	0	32	64	96	...

5.3 Memory Alignment

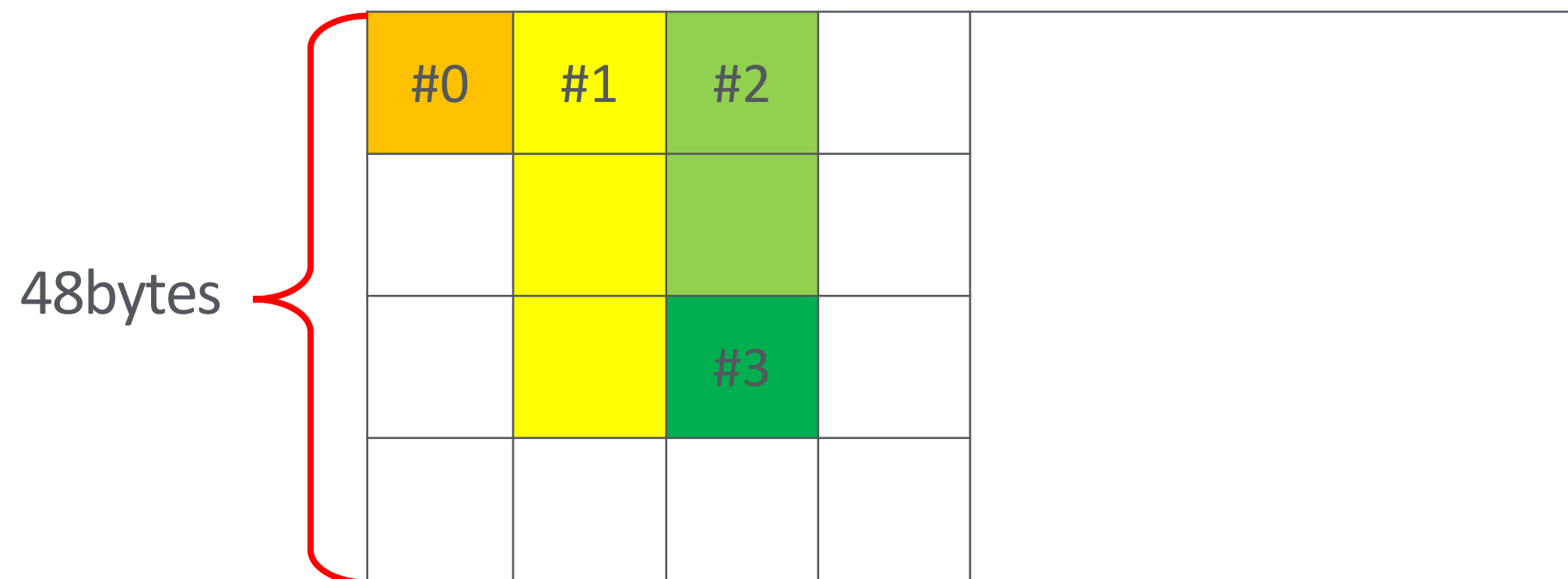
packed 기능을 지원하지 않는다

기대 값



packed(4) 미지원

결과



4bytes

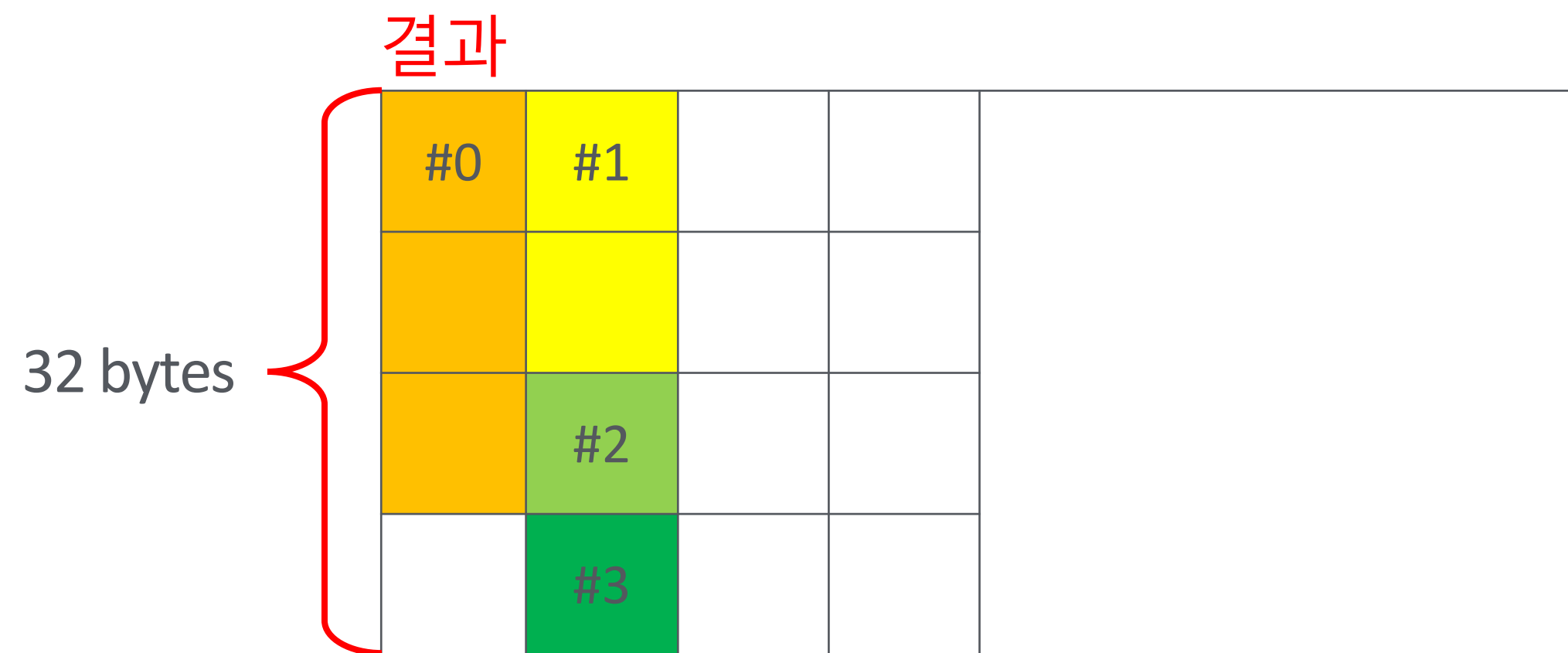


vertex shader

```
layout (binding = 0) uniform UniformBufferObject {
    float ratio; // #0
    vec3 color; // #1
    vec2 tex; // #2
    float opacity; // #3
} ubo;
```


5.3 Memory Alignment

Memory Alignment에 맞춰서 배치를 조정하자.

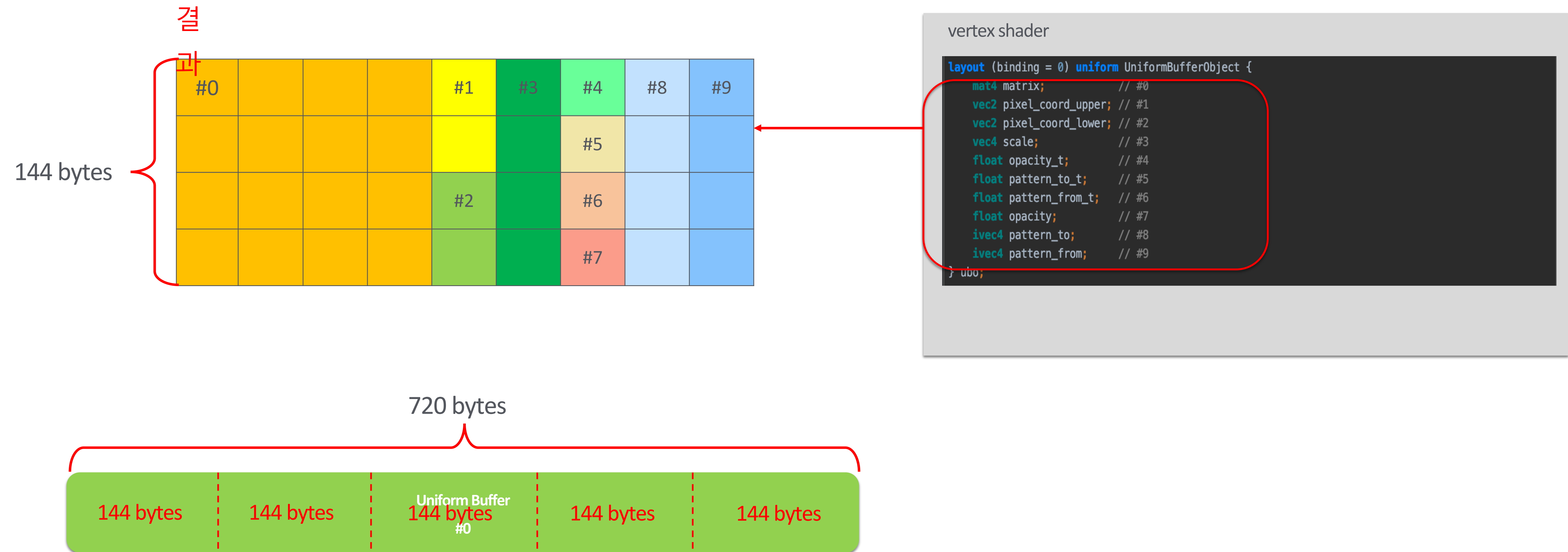


vertex shader

```
layout (binding = 0) uniform UniformBufferObject {
    vec3 color; // #0
    vec2 tex; // #1
    float opacity; // #2
    float ratio; // #3
} ubo;
```


5.4 Push Constant

팁: Uniform Buffer가 너무 비대해지는 경우?



5.4 Push Constant

push_constant로 16x16 matrix를 옮기자.

결과



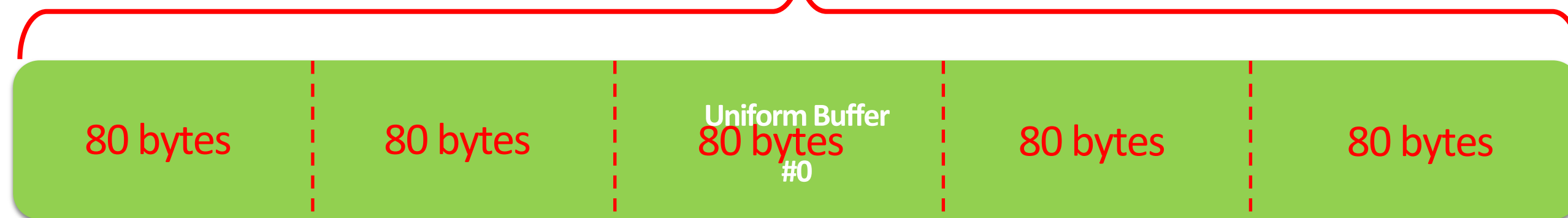
80bytes

vertex shader

```
layout (push_constant) uniform PushConsts {
    mat4 matrix;
} pushConsts;

layout (binding = 0) uniform UniformBufferObject {
    vec2 pixel_coord_upper; // #0
    vec2 pixel_coord_lower; // #1
    vec4 scale;              // #2
    float opacity_t;         // #3
    float pattern_to_t;      // #4
    float pattern_from_t;    // #5
    float opacity;           // #6
    ivec4 pattern_to;        // #7
    ivec4 pattern_from;       // #8
} ubo;
```

400bytes



5.4 Push Constant

장점: 사용이 쉽다.

```
// Update Model View Projection Matrix
vkCmdPushConstants(drawCmdBuffer,
    pipelineLayout,
    stageFlags,
    offset,
   .mvp.size()),
.mvp.data());
```

단점: 하지만 크기 제약 사항에 유의하자.

- Device Limit 체크: `VkPhysicalDeviceLimits::maxPushConstantsSize`

6. 결론 및 향후 계획

6.1 결론

어떤 부분이 개선 되었는가?

- **안정성:** Metal로 랩핑된 OpenGL 크래시가 제거되었습니다.
- **코드 가시화:** GPU Capture 기능을 이용한 Shader 디버깅이 가능합니다.
- **성장:** Modern Graphics 아키텍처를 이해함으로써 렌더링 엔진에 대한 깊이 있는 구현을 경험할 수 있습니다.

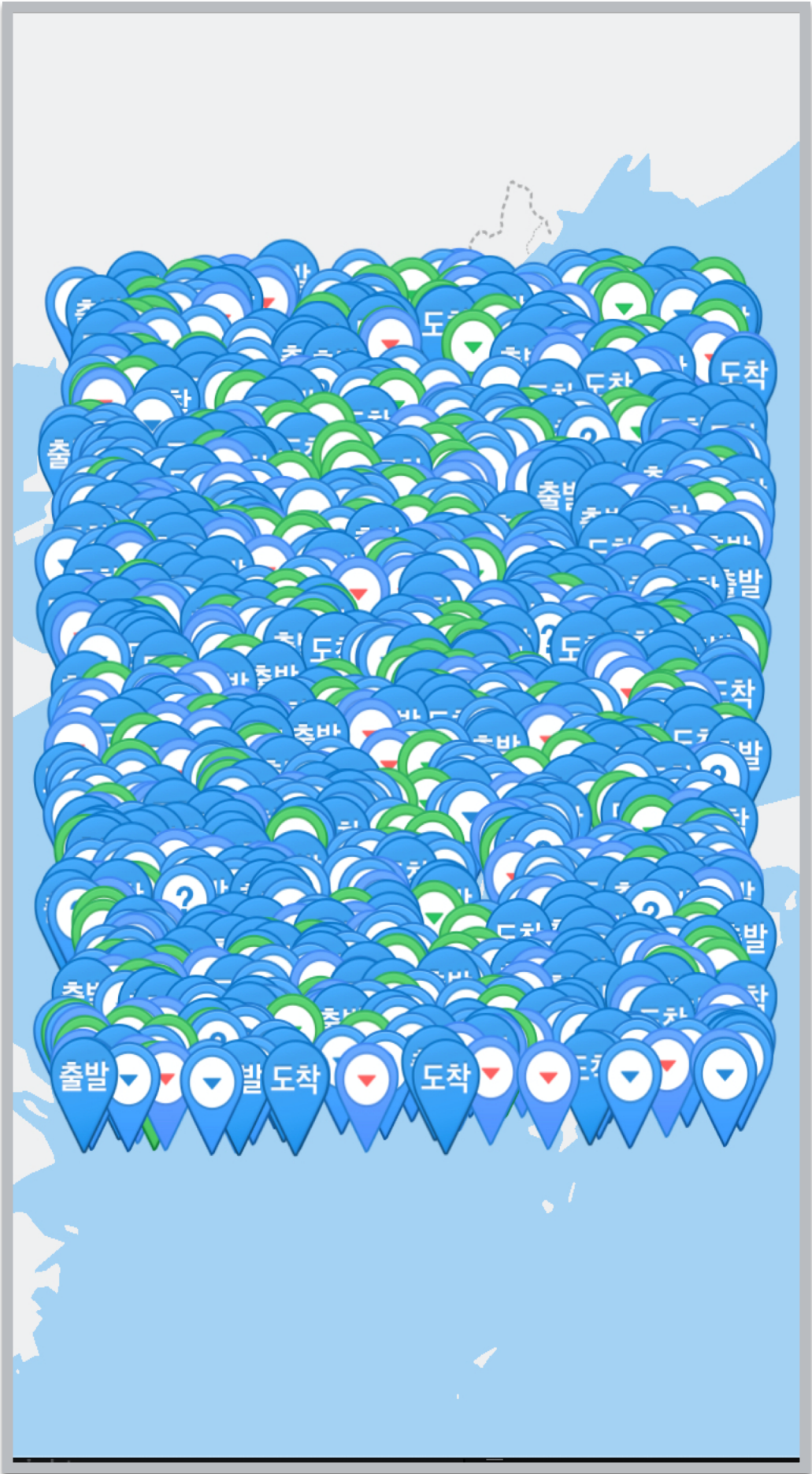
6.1 결론

어떤 부분이 개선 되었는가?

- Metal Profile

개별 오버레이 이미지 테스트

iPhone X				
	256x256(2000개)		512x512(1000개)	
FPS	OpenGL	Metal	OpenGL	Metal
	25.12fps	29.61fps (+14.86%)	26.51fps	32.02fps (+13.32%)
Memory	OpenGL	Metal	OpenGL	Metal
	784MB	594MB (-190MB, 24.23%)	1430MB	1070MB (-130MB, 25.17%)



6.1 결론

어떤 부분이 개선 되었는가?

- Vulkan Profile

FPS Test

	Galaxy Note 10		Pixel3 XL	
	OpenGL	Vulkan	OpenGL	Vulkan
FPS	58.51	59.50(+1.6%)	59.28	60.17(+1.5%)



6.2 향후 계획

앞으로 무엇을 할 예정인가?

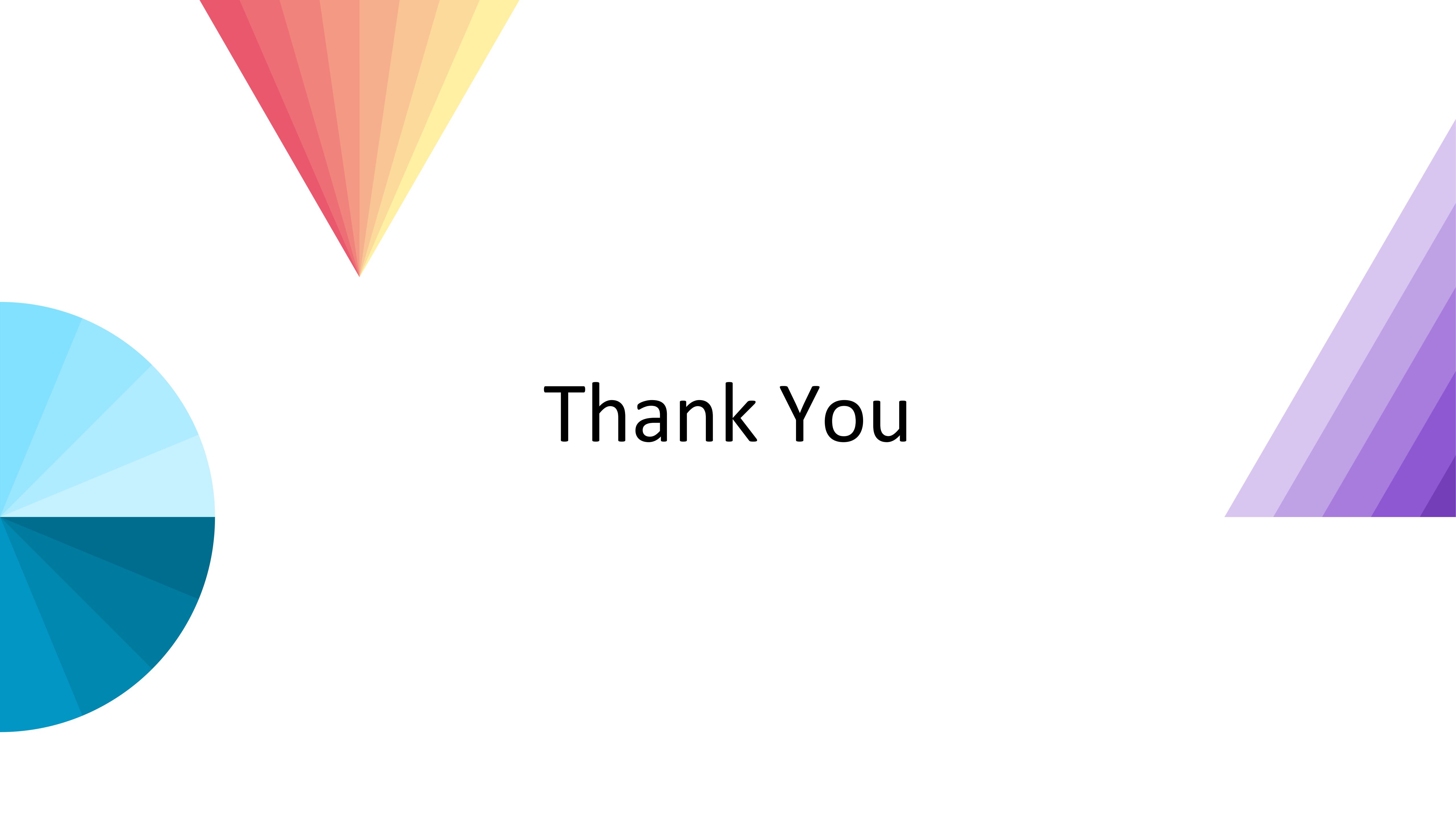
Full 3D 지도

Compute
Shader 활용

GPU Bound
Performance



Q & A



Thank You