2023-2학기 겜마루 Direct3D11 스터디

숭실대학교 겜마루 27기 컴퓨터학부 22학번 김민규

mkkim0612@naver.com

스터디 수정 사항

- 빨강 형광펜 : 유일하게 고칠 코드
- 회색 형광펜 : 전에 이미 만들었던 코드

- 디버깅을 하지 않으려고 한다.
  - 한 분이 버그가 나도 넘어가려고 한다.
  - 진도를 많이 나기고 시간을 효율적으로 사용하기 위함
  - 1차시에서 삼각형 띄우기에 성공했다면 그 이후 진행에서의 오류의 원인은 99% 오타일 것임
  - 대신에 중간 도달점마다 통째로 복붙할 수 있는 파일 제공

- 추가 자료
  - https://microsoft.github.io/DirectX-Specs/d3d/archive/D3D11_3_FunctionalSpec.htm

Direct3D11 스터디
2회차 - 3D, Buffers, Light!

- 실습 - 3D 렌더링
- 실습 - 텍스쳐 매핑
- 실습 - 조명 계산

Direct3D11 스터디
Ch. 2-1 3D 렌더링

- 우리의 목표
- 렌더링 파이프라인 리뷰
- Vertex Shader
- 실습 : 정육면체 추가
- 버퍼
- 실습 : 3D 행렬
- 실습 : 카메라
- 실습 : 여러 개 오브젝트
- 실습 : 깊이 버퍼

우리의 목표

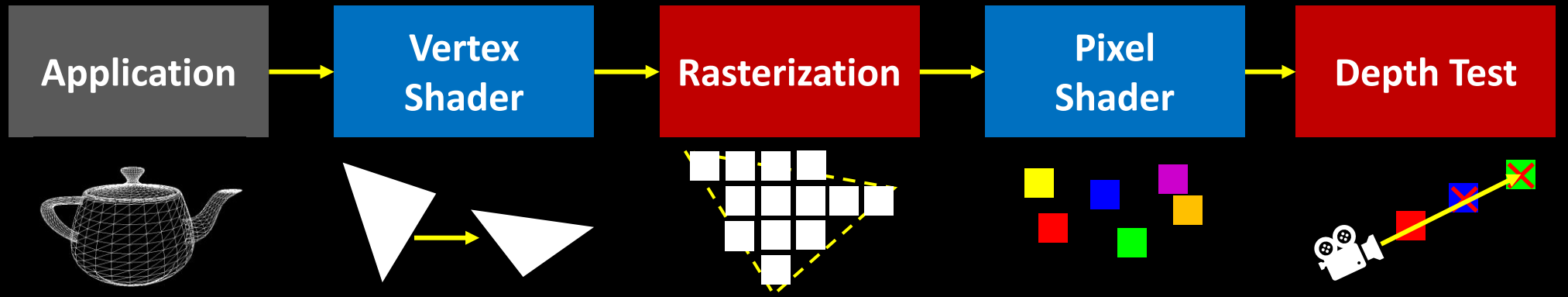
- 3D 정육면체를 렌더링할 것이다.

- 우전 정육면체 데이터를 GPU에 넣어서 2D로 렌더링한다
- 그 다음 3D로 렌더링되도록 GPU에 행렬을 넣어 계산을 할 것이다
- 그리고 카메라를 추가해 이동할 수 있도록 할 것이다

렌더링 파이프라인 리뷰

- 렌더링 파이프라인 중 핵심적인 일부 과정만 설명.

CPU에서 처리
프로그램 가능
설정만 가능



CPU에서 입력, 애니메이션, 충돌 처리 등 계산해 모델을 GPU에 넘김

삼각형의 정점들의 좌표를 3D에서 2D로 변환하고, 그 과정에서 삼각형의 여러 값들을 계산

2D로 변환된 삼각형을 픽셀 단위로 쪼갬

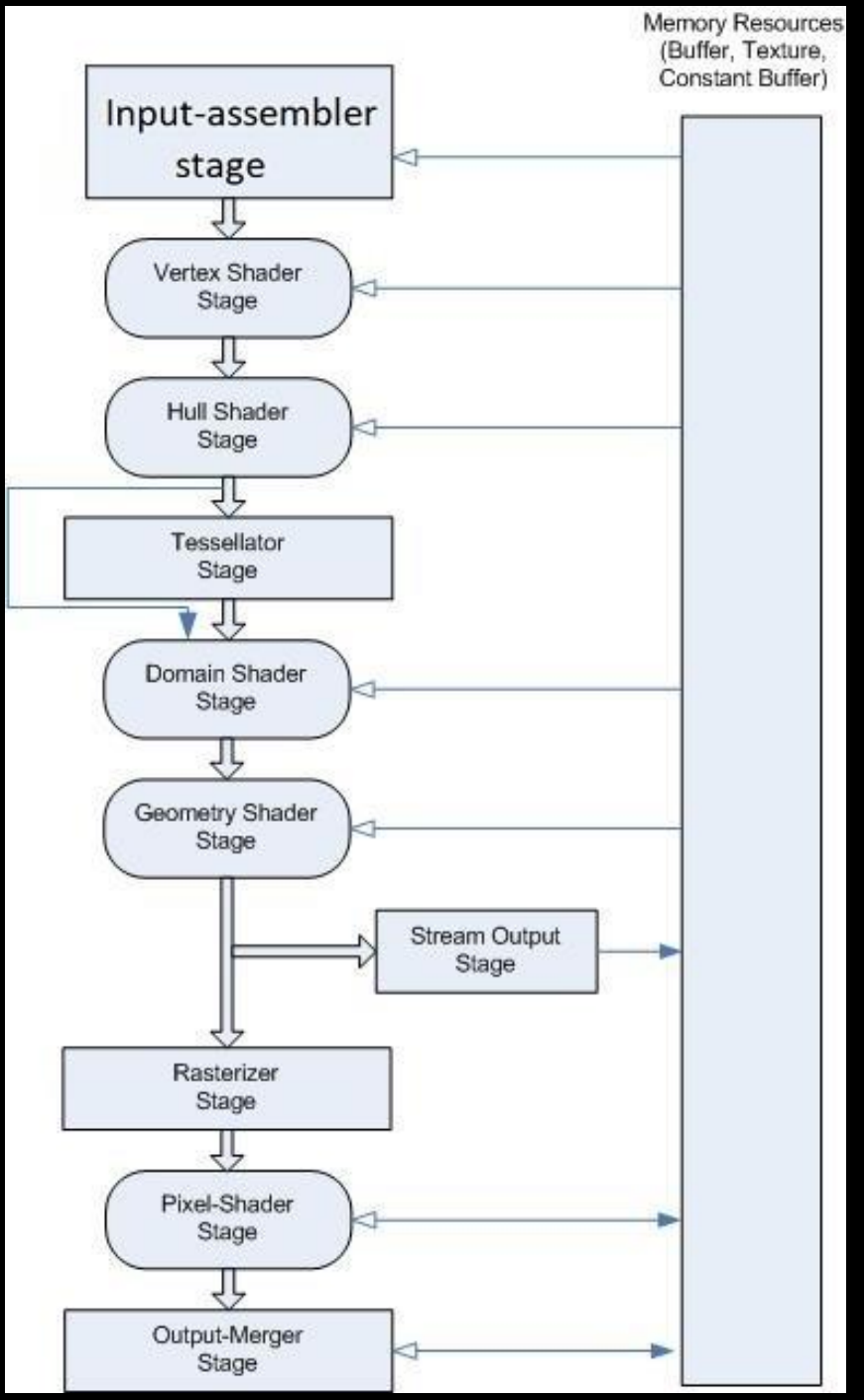
픽셀들의 색을 계산함

카메라에서 보이는 픽셀만 남겨 최종 이미지에 저장

- 셰이더는 GPU에서 돌아가는 특별한 함수(C언어의 함수)라고 생각하면 된다.
- 버텍스(정점) 셰이더부터는 모두 GPU에서 수행된다.
- 더 복잡한 단계들도 많지만 생략

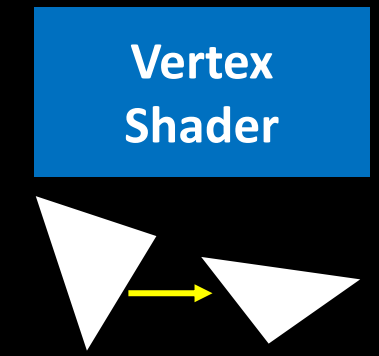
렌더링 파이프라인

- Input-Assembler(IA) : 버퍼에서 정점 데이터를 읽어 다른 곳에서 쓰도록 조립
- Vertex Shader(VS) : 입력된 정점들을 3D 변환
- Hull Shader/Tessellator/Domain Shader(HS/x/DS) : 한 도형을 여러 조각으로 쪼갬
- Geometry Shader(GS) : 한 도형을 만들거나 복제하거나 삭제함
- Stream Output(SO) : 처리된 정점들을 다른 용도로 쓰도록 다시 가져옴
- Rasterizer(RS) : 도형을 픽셀들로 쪼갬
- Pixel Shader(PS) : 픽셀별로 색을 계산함
- Output Merger(OM) : 픽셀들을 합쳐 이미지를 만듬



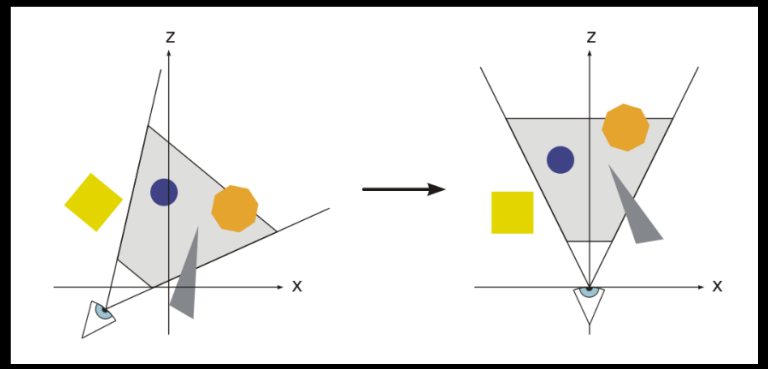
Vertex Shader

- 인스턴스 데이터의 값에 따라 모델을 알맞은 곳으로 변환한다
- 최종적으로는 삼각형들이 2D 좌표를 갖게 된다
- 좌표 변환 과정에서 색/조명/안개 등에 필요한 값들도 같이 계산한다



삼각형의 정점들의 좌표를 3D에서 2D로 변환하고,
그 과정에서 삼각형의 여러 값들을 계산

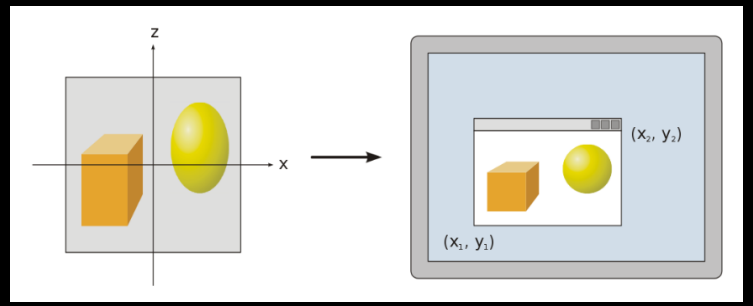
- 일단 카메라를 기준으로(카메라가 (0, 0, 0), 바라보는 곳이 +z) 이동시킨다(참고: d3d는 왼손좌표계)
  이때 z값이 카메라와의 거리가 되는데 뒤의 Depth Test에서 이 값을 사용한다



모델 | 인스턴스 데이터 | 텍스쳐

버텍스 셰이더

- 가까운 물체를 크게, 먼 물체를 작게 투영시킨다
- 화면의 해상도와 일치하는 크기의 2D 좌표로 변환한다



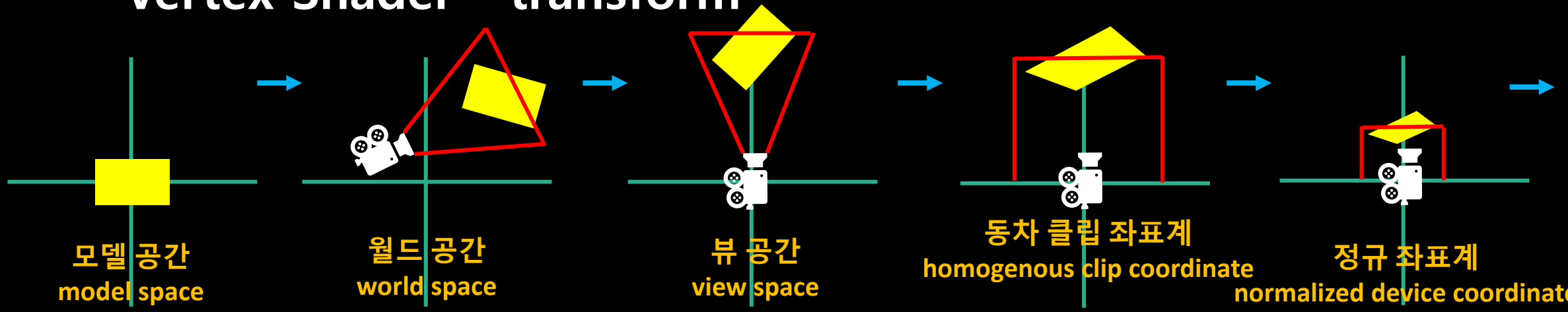
- 이 과정은 버텍스 셰이더를 통해 프로그래밍 가능함
- 버텍스(정점) 셰이더는 각 정점별로 적용된다

Vertex Shader

- Input Assembler(IA)로부터 정점들을 받아 처리한다
- 셰이더를 통해 프로그래밍 가능
- 정점별로 병렬적으로 GPU에서 실행된다
- 기본적 기능인 3D 변환뿐만 아니라 스키닝, 모핑, 정점 별 라이팅 등의 추가적 연산도 수행한다

- 각 정점은 최대 32bit 4원소 벡터 16 개를 가질 수 있다
- HW에서 생성한 VertexID와 InstanceID 값을 사용할 수 있다 (SV_semantics로 지정)

Vertex Shader - transform



모델 공간
model space

월드 공간
world space

뷰 공간
view space

동차 클립 좌표계
homogenous clip coordinate

정규 좌표계
normalized device coordinate

모델의 정점들이
각 인스턴스의 좌표로
이동하도록
모델 변환
w = 1

카메라가 [0, 0, 0]에
위치하고
+z 를 바라보도록
뷰 변환
w = 1

가까운 물체는 크게,
먼 물체는 크게
투시 투영
x, y ∈ [-w, w]
z ∈ [0, w]
w = something

범위에 들어가도록
각 원소를 w로 나누기
x, y ∈ [-1, 1]
z ∈ [0, 1]
w = 1

x, y값만 취해
창 크기만큼
커지도록
뷰포트 변환

Model transform → **View transform** → **Projection transform** → **Clipping** → **Divide** → **Viewport transform** → **Rasterize**

정점 셰이더에서
직접 프로그래밍해
작업

래스터라이저에서
고정된 HW로
작업

- 정점 셰이더에서는 모델/뷰/투영 행렬이 필요하다
- 정점 셰이더의 출력 정점은 동차 클립 좌표계에 있어야 한다

실습 - 모델 구조체 정의

```
// D3DDevice.cpp

ID3D11VertexShader* g_pVertexShader = NULL;
ID3D11PixelShader* g_pPixelShader = NULL;
ID3D11InputLayout* g_pInputLayout = NULL;
ID3D11Buffer* g_pVertexBuffer = NULL;

using namespace DirectX;
struct VertexStructure1 {
    XMFLOAT3 pos;
    XMFLOAT2 uv;
};

bool InitD3D(HWND hwnd)
{
```

- DirectX::XMFLOAT3을 편하게 쓰기 위해 using namespace를 해주자.
- uv는 지금 당장은 안 쓰지만 나중을 위해 미리 만들어둠.

실습 - 인풋 레이아웃 수정

- 전 차시에 만든 Input Layout을 방금 만든 구조체에 맞게 수정.

```
// D3DDevice.cpp InitD3D()

D3D11_INPUT_ELEMENT_DESC layouts[2];
UINT numElements = ARRAYSIZE(layouts);
layouts[0].SemanticName = "POSITION";
layouts[0].SemanticIndex = 0;
layouts[0].Format = DXGI_FORMAT_R32G32B32_FLOAT;
layouts[0].InputSlot = 0;
layouts[0].AlignedByteOffset = 0;
layouts[0].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
layouts[0].InstanceDataStepRate = 0;

layouts[1].SemanticName = "TEXCOORD";
layouts[1].SemanticIndex = 0;
layouts[1].Format = DXGI_FORMAT_R32G32_FLOAT;
layouts[1].InputSlot = 0;
layouts[1].AlignedByteOffset = sizeof(XMFLOAT3);
layouts[1].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
layouts[1].InstanceDataStepRate = 0;
```

실습 - 모델 데이터 만들기

- 전 차시에 만든 XMFLOAT3의 배열인 vertices를 수정하자.

```
// D3DDevice.cpp InitD3D()

VertexStructure1 vertices[] =
{
    // front
    { XMFLOAT3(-1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(1.0f, 0.0f) },
    { XMFLOAT3( 1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3(-1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3( 1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 1.0f) },

    // back
    { XMFLOAT3( 1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(1.0f, 0.0f) },
    { XMFLOAT3(-1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3( 1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3(-1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(0.0f, 1.0f) },

    // left
    { XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3(-1.0f,  1.0f, -1.0f), XMFLOAT2(1.0f, 0.0f) },
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3(-1.0f, -1.0f,  1.0f), XMFLOAT2(0.0f, 1.0f) },

    // right
    { XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3( 1.0f,  1.0f,  1.0f), XMFLOAT2(1.0f, 0.0f) },
    { XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3( 1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 1.0f) },

    // down
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3( 1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 0.0f) },
    { XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3(-1.0f, -1.0f,  1.0f), XMFLOAT2(0.0f, 1.0f) },

    // down
    { XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3(-1.0f,  1.0f, -1.0f), XMFLOAT2(1.0f, 0.0f) },
    { XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3( 1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 1.0f) },
};
UINT vertices_size = ARRAYSIZE(vertices);
```

실습 - 버퍼 정보, 렌더링 도형 수 수정

```cpp
// D3DDevice.cpp InitD3D()

D3D11_BUFFER_DESC bd;
ZeroMemory(&bd, sizeof(bd));
bd.Usage = D3D11_USAGE_DEFAULT;
bd.ByteWidth = sizeof(vertices);
bd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
bd.CPUAccessFlags = 0;
```

```cpp
// D3DDevice.cpp InitD3D() 거의 맨 아래쪽

UINT stride = sizeof(VertexStructure1);
UINT offset = 0;
g_pImmediateContext->IASetVertexBuffers(
    0, 1, &g_pVertexBuffer, &stride, &offset);
```

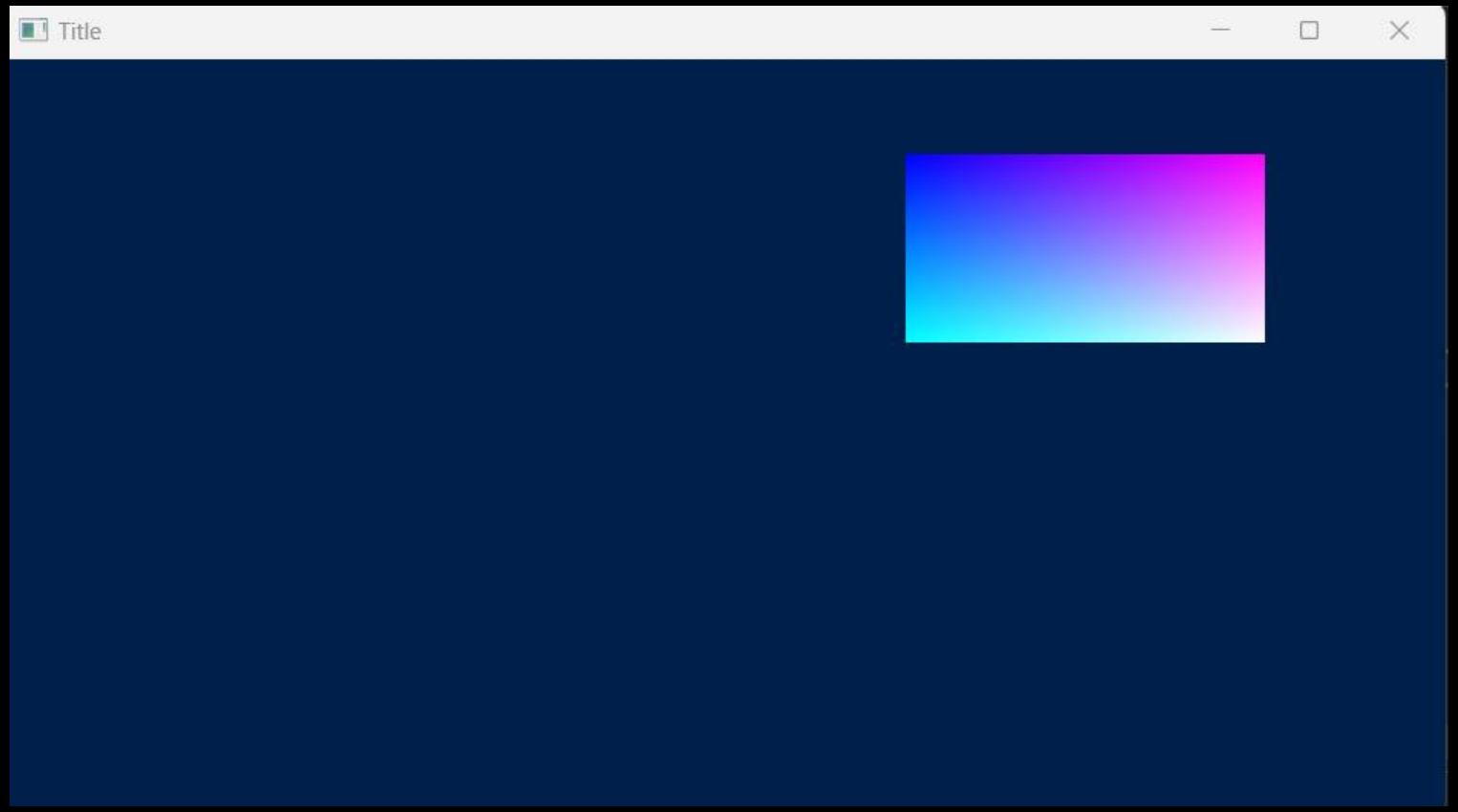
```cpp
// D3DDevice.cpp
void Render()
{
    float clearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
    g_pImmediateContext->ClearRenderTargetView(g_pRenderTargetView, clearColor);

    g_pImmediateContext->VSSetShader(g_pVertexShader, NULL, 0);
    g_pImmediateContext->PSSetShader(g_pPixelShader, NULL, 0);
    g_pImmediateContext->Draw(3 * 2 * 6, 0);

    g_pSwapChain->Present(0, 0);
}
```

실습 - 셰이더 수정

- output.Pos를 고치는 이유 : 입력 모델의 정점은 좌표가 -1~1인데 이러면 동차 클립 공간 기준으로 카메라에 보이지 않게 됨

```
// Shader1.fx

struct VS_Input
{
    float3 Pos : POSITION;
    float2 UV : TEXCOORD;
};

struct VS_Output
{
    float4 Pos : SV_POSITION;
    float2 UV : TEXCOORD;
};

VS_Output VS(VS_Input input)
{
    VS_Output output;
    output.Pos = float4(input.Pos * 0.25f + 0.5f, 1.0f);
    output.UV = input.UV;
    return output;
}

float4 PS(VS_Output input) : SV_Target
{
    return float4(input.UV, 1.0f, 1.0f);
}
```

실습 - 결과



D3D11의 버퍼

- 버퍼는 타입을 가진 데이터의 모음임.
- 종류 : Vertex Buffer, Index Buffer, Constant Buffer

- Vertex Buffer : 위치 벡터, 노멀 벡터, 텍스쳐 좌표 등 저장, 정점 셰이더에서 원소별로 각각 처리
- Index Buffer : 도형을 구성하는 정점을 나타내는 인덱스 저장
- Constant Buffer : 변환을 위한 행렬, 조명 정보 등 정점에 독립적인 정보를 저장

- 일반적인 버퍼는 UndateSubResource() 함수로 내용 갱신
  - 자주 업데이트되지 않을 거라고 예상되는 메모리는 이 방식이 좋다.
  - GPU의 메모리 중 빨리 접근할 수 있지만 수정 불가능한 곳에 메모리를 잡는다
  - 갱신할 때 기존 데이터를 GPU가 사용하지 않고 있다면 훨씬 빠르다.

- Dynamic Resource : 버퍼와 텍스쳐를 dynamic으로 지정 가능. CPU가 주기적으로 써넣을 수 있는 메모리.
  - CPU와 GPU 양쪽에 메모리를 잡아 놓고 주기적으로 CPU에서 GPU로 내용을 복사한다.
  - CPU에서 써넣을 땐 Map() 과 Unmap() 을 이용한다. (두 함수 사이에 GPU가 접근하지 못함)

- 두 방식 모두 현재 프레임 렌더링에 필요한 정보인 기존의 데이터는 그대로 두고 새 복제본을 만듬

상수 버퍼 패딩

- Constant Buffer의 경우 HLSL에서 자동으로 패딩을 한다
- Constant Buffer의 크기는 16의 배수가 된다
- 각 원소는 16바이트 (원소 4개) 경계에 걸리면 안 된다 (걸릴 경우 다음 16바이트부터 시작)
- CPU에서 구조체 만들 때 맞춰줘야 함 (안 맞추면 에러 혹은 데이터가 이상한 곳으로 들어감)

```
cbuffer B1
{
    float4 Val1;
    float2 Val2;
    float2 Val3;
};
```

Val1[0]	Val1[1]	Val1[2]	Val1[3]
Val2[0]	Val2[1]	Val3[0]	Val3[1]

```
cbuffer B2
{
    float2 Val4;
    float4 Val5;
    float2 Val6;
};
```

Val4[0]	Val4[1]	padd	padd
Val5[0]	Val5[1]	Val5[2]	Val5[3]
Val6[0]	Val6[1]	padd	padd

```
struct B1
{
    XMFLOAT4 Val1;
    XMFLOAT2 Val2;
    XMFLOAT2 Val3;
};

struct B2
{
    XMFLOAT2 Val4;
    XMFLOAT2 padd1;
    XMFLOAT4 Val5;
    XMFLOAT2 Val6;
};
```

실습 - 상수 버퍼 구조체 정의

```
// D3DDevice.cpp

using namespace DirectX;
struct VertexStructure1 {
    XMFLOAT3 pos;
    XMFLOAT2 uv;
};

struct VSConstBufferPerFrame {
    XMMATRIX world;
    XMMATRIX view;
};

struct VSConstBufferPerResize {
    XMMATRIX projection;
};

ID3D11Buffer* g_pVSConstBufferPerFrame = NULL;
ID3D11Buffer* g_pVSConstBufferPerResize = NULL;
```

실습 - 상수 버퍼 자원 해제

```
// D3DDevice.cpp

void ClearD3D()
{
    if (g_pImmediateContext != NULL) g_pImmediateContext->ClearState();

    if (g_pVSConstBufferPerResize != NULL) g_pVSConstBufferPerResize->Release();
    if (g_pVSConstBufferPerFrame != NULL) g_pVSConstBufferPerFrame->Release();
```

실습 - 상수 버퍼 할당

```
// D3DDevice.cpp InitD3D() 함수의 맨 아래

g_pImmediateContext->IASetPrimitiveTopology(
    D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);

ZeroMemory(&bd, sizeof(bd));
bd.Usage = D3D11_USAGE_DYNAMIC;
bd.ByteWidth = sizeof(VSConstBufferPerFrame);
bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
bd.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;

hr = g_pd3dDevice->CreateBuffer(
    &bd, NULL, &g_pVSConstBufferPerFrame);
if (FAILED(hr))
    return false;

g_pImmediateContext->VSSetConstantBuffers(
    0, 1, &g_pVSConstBufferPerFrame);
```

```
// D3DDevice.cpp InitD3D() 왼쪽 코드 이어서

VSConstBufferPerResize vsConstBufferPerResize;
// fill data

ZeroMemory(&bd, sizeof(bd));
bd.Usage = D3D11_USAGE_DEFAULT;
bd.ByteWidth = sizeof(VSConstBufferPerResize);
bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
bd.CPUAccessFlags = 0;

ZeroMemory(&InitData, sizeof(InitData));
InitData.pSysMem = &vsConstBufferPerResize;

hr = g_pd3dDevice->CreateBuffer(
    &bd, &InitData, &g_pVSConstBufferPerResize);
if (FAILED(hr))
    return false;

g_pImmediateContext->VSSetConstantBuffers(
    1, 1, &g_pVSConstBufferPerResize);

return true;
```

실습 - 투영 행렬 만들기

```
// D3DDevice.cpp InitD3D()

VSConstBufferPerResize vsConstBufferPerResize;
// fill data
vsConstBufferPerResize.projection = XMMatrixPerspectiveFovLH(
    XM_PI / 3.0f, (float)width / (float)height, 0.01f, 1000.0f);
```

실습 - 오브젝트, 카메라 이동 변수 만들기

```
// D3DDevice.cpp 거의 맨 위쪽

ID3D11Buffer* g_pVSConstBufferPerFrame = NULL;
ID3D11Buffer* g_pVSConstBufferPerResize = NULL;

float objPos[3];
float objPitchYawRoll[3];
float camPos[3] = { 1.0f, 2.0f, -7.0f };
float camPitchYawRoll[3] = { 0.3f, -0.5f, 0.0f };
```

실습 - 월드, 뷰 행렬 만들기

```
// D3DDevice.cpp Render() 함수 맨 위

float clearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
g_pImmediateContext->ClearRenderTargetView(g_pRenderTargetView, clearColor);

static long long t = 0;
++t;
VSConstBufferPerFrame vsConstBufferPerFrame;
vsConstBufferPerFrame.world = XMMatrixMultiply(
    XMMatrixRotationRollPitchYaw(
        t/1000.0f + objPitchYawRoll[0],
        t/2000.0f + objPitchYawRoll[1],
        t/3000.0f + objPitchYawRoll[2]),
    XMMatrixTranslation(objPos[0], objPos[1], objPos[2])
);
vsConstBufferPerFrame.view = XMMatrixMultiply(
    XMMatrixTranslation(-camPos[0], -camPos[1], -camPos[2]),
    XMMatrixTranspose(XMMatrixRotationRollPitchYaw(
        camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]))
);

D3D11_MAPPED_SUBRESOURCE resourceVSConstBuffer;
ZeroMemory(&resourceVSConstBuffer, sizeof(resourceVSConstBuffer));

g_pImmediateContext->Map(g_pVSConstBufferPerFrame, 0, D3D11_MAP_WRITE_DISCARD, 0, &resourceVSConstBuffer);
memcpy(resourceVSConstBuffer.pData, &vsConstBufferPerFrame, sizeof(VSConstBufferPerFrame));
g_pImmediateContext->Unmap(g_pVSConstBufferPerFrame, 0);
```

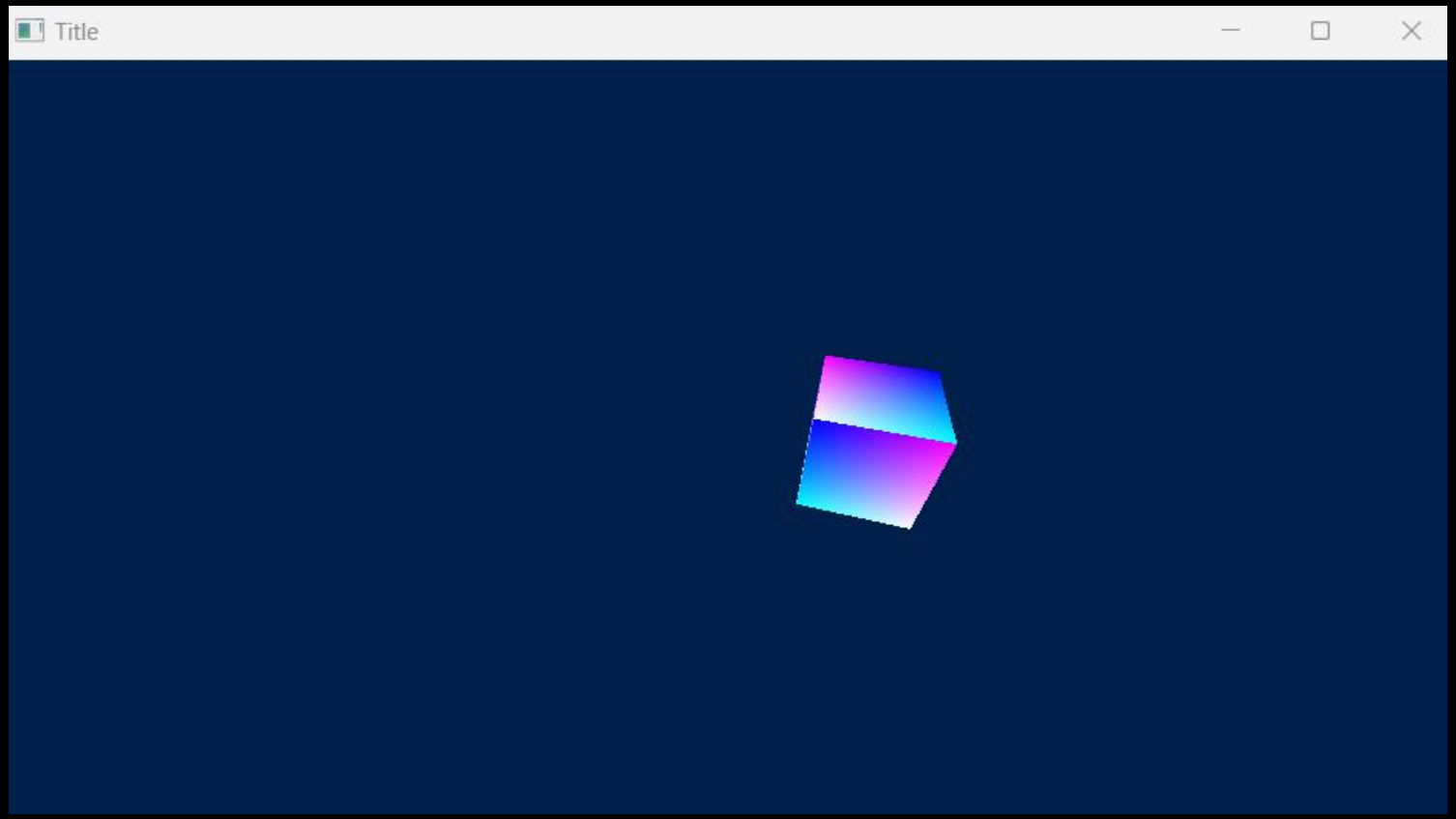
- DirectX에서 쓰는 변환 행렬들은 흔한 수학 행렬들의 전치이다.
- 그래서 XMMatrixTranslation/Rotation/Projection 등의 함수는 흔히 생각하는 행렬의 전치행렬을 리턴한다.
  행렬들은 row-major로 저장된다.

실습 - 셰이더 수정

- VS_Input, VS_Output, PS 는 그대로 남겨놓는다

```
// Shader1.fx

cbuffer VSConstBufferPerFrame : register(b0)
{
    float4x4 MatWorld;
    float4x4 MatView;
};

cbuffer VSConstBufferPerResize : register(b1)
{
    float4x4 MatProj;
};

VS_Output VS(VS_Input input)
{
    VS_Output output;
    output.Pos = float4(input.Pos, 1.0f);
    output.Pos = mul(MatWorld, output.Pos);
    output.Pos = mul(MatView, output.Pos);
    output.Pos = mul(MatProj, output.Pos);
    output.UV = input.UV;
    return output;
}
```

실습 - 결과



카메라

- 카메라를 만들 것임
- wasd이동, 마우스로 회전

```
// ProjectName.cpp 파일 맨 위

#include <Windows.h>
#include <cmath>
#include <DirectXMath.h>

bool InitD3D(HWND hwnd);
void ClearD3D();
void Render();

extern float camPos[3];
extern float camPitchYawRoll[3];
const float PI = 3.1415926535f;
```

카메라 - WndProc

```
// ProjectName.cpp WndProc() 함수 맨 위

switch(msg)
{
case WM_CLOSE:
    DestroyWindow(hwnd);
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
case WM_KEYDOWN:
    switch ((char)wParam)
    {
    case 'W':
    {
        auto mat = DirectX::XMMatrixRotationRollPitchYaw(
            camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]);
        camPos[0] += mat.r[2].m128_f32[0];
        camPos[1] += mat.r[2].m128_f32[1];
        camPos[2] += mat.r[2].m128_f32[2];
        break;
    }
    case 'S':
    {
        auto mat = DirectX::XMMatrixRotationRollPitchYaw(
            camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]);
        camPos[0] -= mat.r[2].m128_f32[0];
        camPos[1] -= mat.r[2].m128_f32[1];
        camPos[2] -= mat.r[2].m128_f32[2];
        break;
    }
    case 'A':
    {
        auto mat = DirectX::XMMatrixRotationRollPitchYaw(
            camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]);
        camPos[0] -= mat.r[0].m128_f32[0];
        camPos[1] -= mat.r[0].m128_f32[1];
        camPos[2] -= mat.r[0].m128_f32[2];
        break;
    }
    case 'D':
    {
        auto mat = DirectX::XMMatrixRotationRollPitchYaw(
            camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]);
        camPos[0] += mat.r[0].m128_f32[0];
        camPos[1] += mat.r[0].m128_f32[1];
        camPos[2] += mat.r[0].m128_f32[2];
        break;
    }
    case VK_SPACE:
    {
        auto mat = DirectX::XMMatrixRotationRollPitchYaw(
            camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]);
        camPos[0] += mat.r[1].m128_f32[0];
        camPos[1] += mat.r[1].m128_f32[1];
        camPos[2] += mat.r[1].m128_f32[2];
        break;
    }
    case VK_SHIFT:
    {
        auto mat = DirectX::XMMatrixRotationRollPitchYaw(
            camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]);
        camPos[0] -= mat.r[1].m128_f32[0];
        camPos[1] -= mat.r[1].m128_f32[1];
        camPos[2] -= mat.r[1].m128_f32[2];
        break;
    }
    case 'Q':
        camPitchYawRoll[1] = std::remainder(
            camPitchYawRoll[1] - 0.1f, 2.0f * PI);
        break;
    case 'E':
        camPitchYawRoll[1] = std::remainder(
            camPitchYawRoll[1] + 0.1f, 2.0f * PI);
        break;
    case 'R':
        camPitchYawRoll[0] = max(-0.5f * PI, min(0.5f * PI,
            camPitchYawRoll[0] - 0.1f));
        break;
    case 'F':
        camPitchYawRoll[0] = max(-0.5f * PI, min(0.5f * PI,
            camPitchYawRoll[0] + 0.1f));
        break;
    }
    break;
default:
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

여러 개 오브젝트 그리기

- Constant Buffer를 계속 업데이트해 여러 개 오브젝트를 그릴 것이다.
- 서로 겹치는 픽셀이 생기니 z버퍼를 사용해야 한다.

```
// D3DDevice.cpp 파일 위쪽

struct VSConstBufferPerObject {
    XMMATRIX world;
};

struct VSConstBufferPerFrame {
    XMMATRIX view;
};

struct VSConstBufferPerResize {
    XMMATRIX projection;
};

ID3D11Buffer* g_pVSConstBufferPerObject = NULL;
ID3D11Buffer* g_pVSConstBufferPerFrame = NULL;
ID3D11Buffer* g_pVSConstBufferPerResize = NULL;

float objPos[2][3] =
    { { 0.0f, 0.0f, 0.0f }, { 3.0f, 3.0f, 3.0f } };
float objPitchYawRoll[2][3];
```

```
// D3DDevice.cpp

void ClearD3D()
{
    if (g_pImmediateContext != NULL)
        g_pImmediateContext->ClearState();

    if (g_pVSConstBufferPerObject != NULL)
        g_pVSConstBufferPerObject->Release();
    if (g_pVSConstBufferPerResize != NULL)
        g_pVSConstBufferPerResize->Release();
    if (g_pVSConstBufferPerFrame != NULL)
        g_pVSConstBufferPerFrame->Release();
```

실습 - 버퍼 만들기

```
// D3DDevice.cpp InitD3D() 함수 중간

g_pImmediateContext->IASetPrimitiveTopology(
    D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);


ZeroMemory(&bd, sizeof(bd));
bd.Usage = D3D11_USAGE_DEFAULT;
bd.ByteWidth = sizeof(VSConstBufferPerObject);
bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
bd.CPUAccessFlags = 0;

hr = g_pd3dDevice->CreateBuffer(&bd, NULL, &g_pVSConstBufferPerObject);
if (FAILED(hr))
    return false;

g_pImmediateContext->VSSetConstantBuffers(2, 1, &g_pVSConstBufferPerObject);

ZeroMemory(&bd, sizeof(bd));
bd.Usage = D3D11_USAGE_DYNAMIC;
bd.ByteWidth = sizeof(VSConstBufferPerFrame);
bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
bd.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
```

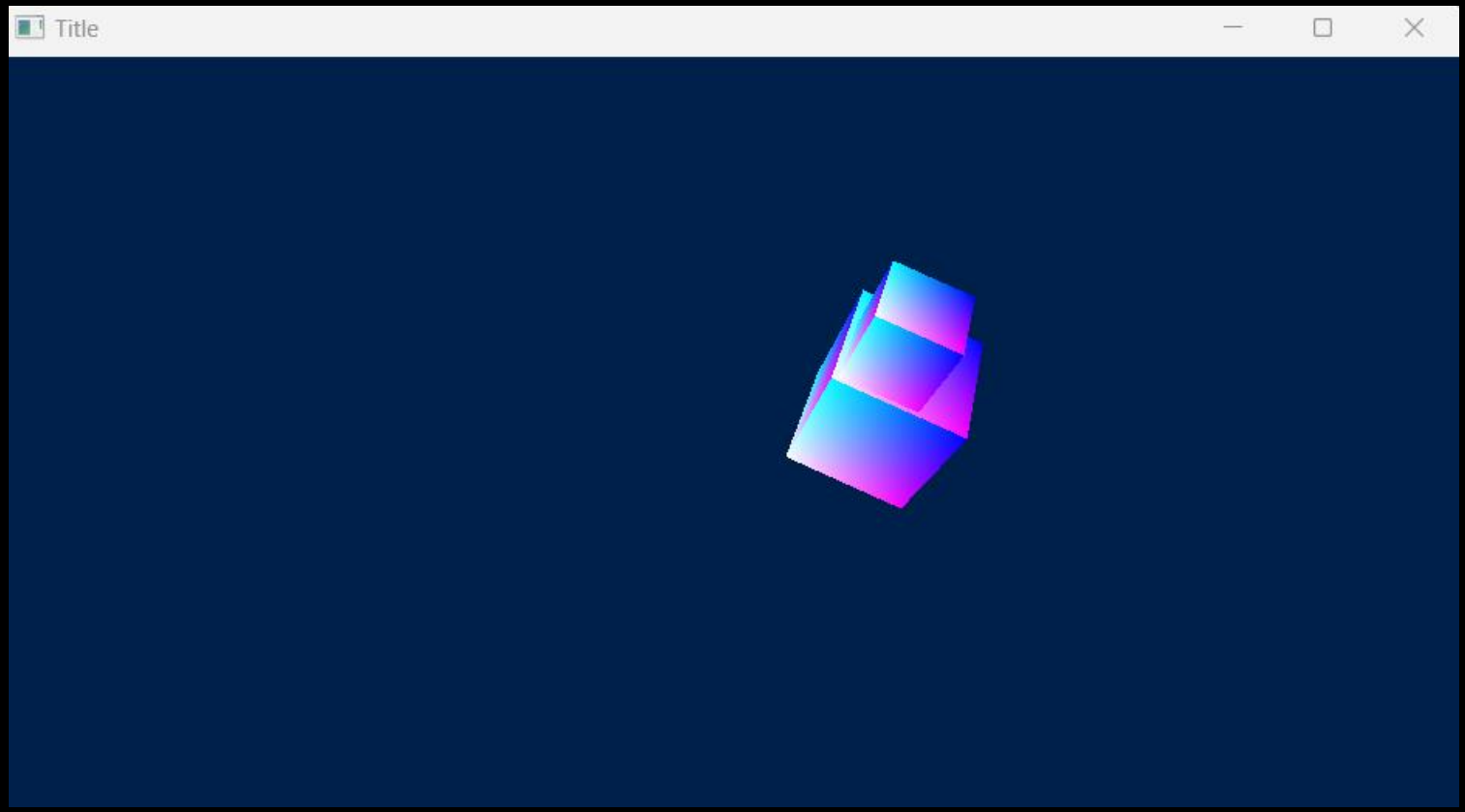
실습 - 렌더링

```
// D3DDevice.cpp Render() 함수 중간

// make matrices
static long long t = 0;
++t;

VSConstBufferPerFrame vsConstBufferPerFrame;
vsConstBufferPerFrame.view = XMMatrixMultiply(
    XMMatrixTranslation(-camPos[0], -camPos[1], -camPos[2]),
    XMMatrixTranspose(XMMatrixRotationRollPitchYaw(
        camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]))
);

D3D11_MAPPED_SUBRESOURCE resourceVSConstBuffer;
ZeroMemory(&resourceVSConstBuffer, sizeof(resourceVSConstBuffer));

g_pImmediateContext->Map(g_pVSConstBufferPerFrame, 0, D3D11_MAP_WRITE_DISCARD, 0, &resourceVSConstBuffer);
memcpy(resourceVSConstBuffer.pData, &vsConstBufferPerFrame, sizeof(VSConstBufferPerFrame));
g_pImmediateContext->Unmap(g_pVSConstBufferPerFrame, 0);

g_pImmediateContext->VSSetShader(g_pVertexShader, NULL, 0);
g_pImmediateContext->PSSetShader(g_pPixelShader, NULL, 0);

VSConstBufferPerObject vsConstBufferPerObject;
for (int i = 0; i < 2; ++i) {
    vsConstBufferPerObject.world = XMMatrixMultiply(
        XMMatrixRotationRollPitchYaw(
            t / 1000.0f + objPitchYawRoll[i][0],
            t / 2000.0f + objPitchYawRoll[i][1],
            t / 3000.0f + objPitchYawRoll[i][2]),
        XMMatrixTranslation(objPos[i][0], objPos[i][1], objPos[i][2]));
    g_pImmediateContext->UpdateSubresource(
        g_pVSConstBufferPerObject, 0, NULL, &vsConstBufferPerObject, 0, 0);
    g_pImmediateContext->Draw(3 * 2 * 6, 0);
}
g_pSwapChain->Present(0, 0);
```

실습 - 셰이더 수정

```
// Shader1.fx 파일의 맨 위

cbuffer VSConstBufferPerObject : register(b2)
{
    float4x4 MatWorld;
};

cbuffer VSConstBufferPerFrame : register(b0)
{
    float4x4 MatView;
};

cbuffer VSConstBufferPerResize : register(b1)
{
    float4x4 MatProj;
};
```

실습 - 결과



실습 - 깊이 버퍼

```
// D3DDevice.cpp 파일의 거의 제일 위쪽

ID3D11Device* g_pd3dDevice = NULL;
ID3D11DeviceContext* g_pImmediateContext = NULL;
IDXGISwapChain* g_pSwapChain = NULL;
ID3D11RenderTargetView* g_pRenderTargetView = NULL;

ID3D11Texture2D* g_pDepthStencil = NULL;
ID3D11DepthStencilView* g_pDSV = NULL;

ID3D11VertexShader* g_pVertexShader = NULL;
ID3D11PixelShader* g_pPixelShader = NULL;
ID3D11InputLayout* g_pInputLayout = NULL;
ID3D11Buffer* g_pVertexBuffer = NULL;
```

```
// D3DDevice.cpp ClearD3D() 함수의 맨 아래쪽

if (g_pDSV != NULL) g_pDSV->Release();
if (g_pDepthStencil != NULL) g_pDepthStencil->Release();

if (g_pRenderTargetView != NULL) g_pRenderTargetView->Release();
if (g_pSwapChain != NULL) g_pSwapChain->Release();
if (g_pImmediateContext != NULL) g_pImmediateContext->Release();
if (g_pd3dDevice != NULL) g_pd3dDevice->Release();
```

실습 - 깊이 버퍼 만들기

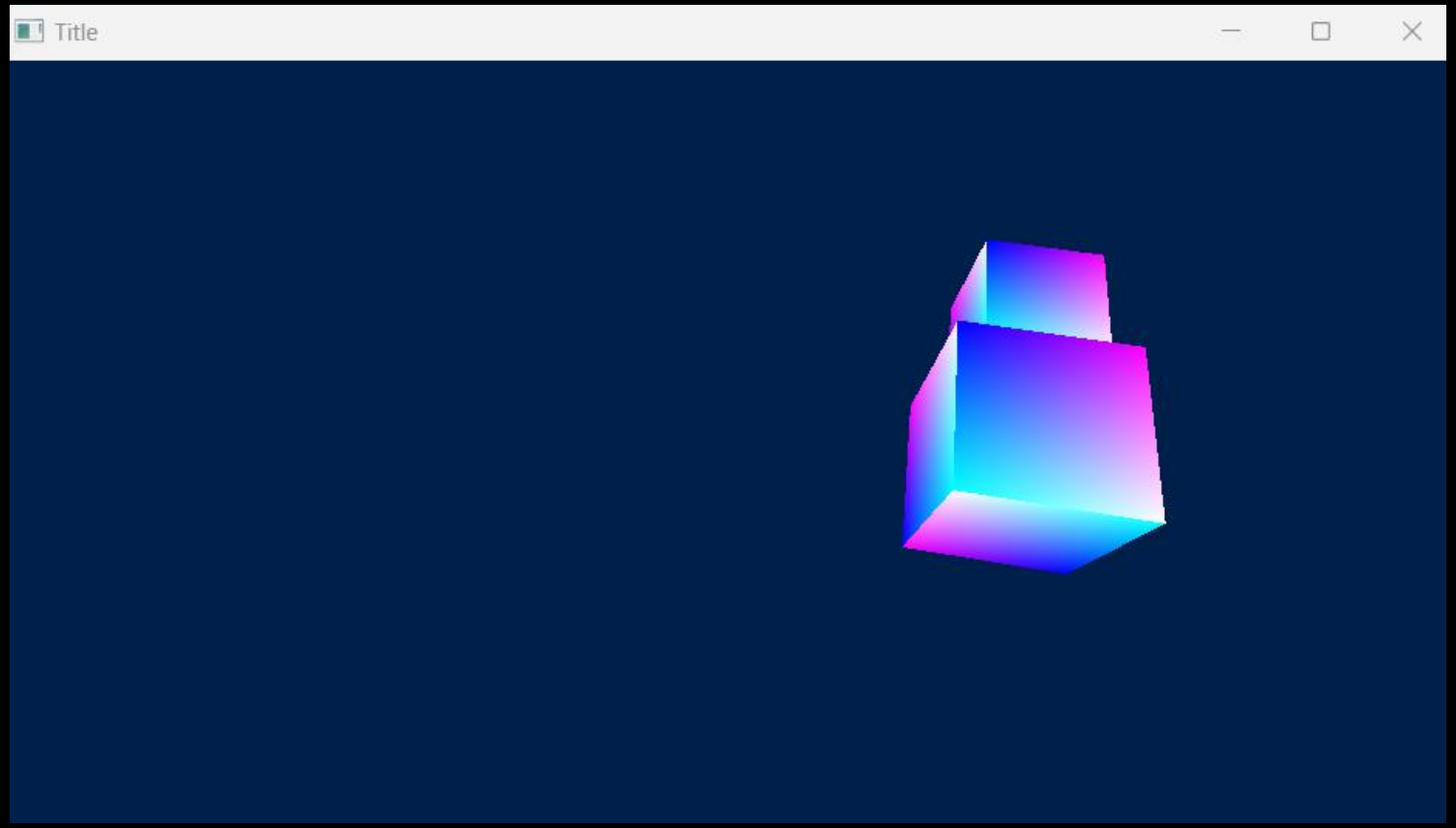
```
// D3DDevice.cpp InitD3D() 함수의 ViewPort 만드는 코드의 위쪽

D3D11_TEXTURE2D_DESC descDepth;
descDepth.Width = width;
descDepth.Height = height;
descDepth.MipLevels = 1;
descDepth.ArraySize = 1;
descDepth.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;
descDepth.SampleDesc.Count = 1;
descDepth.SampleDesc.Quality = 0;
descDepth.Usage = D3D11_USAGE_DEFAULT;
descDepth.BindFlags = D3D11_BIND_DEPTH_STENCIL;
descDepth.CPUAccessFlags = 0;
descDepth.MiscFlags = 0;

hr = g_pd3dDevice->CreateTexture2D(&descDepth, NULL, &g_pDepthStencil);
if (FAILED(hr))
    return false;

D3D11_DEPTH_STENCIL_VIEW_DESC descDSV;
ZeroMemory(&descDSV, sizeof(descDSV));
descDSV.Format = descDepth.Format;
descDSV.ViewDimension = D3D11_DSV_DIMENSION_TEXTURE2D;
descDSV.Texture2D.MipSlice = 0;

hr = g_pd3dDevice->CreateDepthStencilView(g_pDepthStencil, &descDSV, &g_pDSV);
if (FAILED(hr))
    return false;

g_pImmediateContext->OMSetRenderTargets(1, &g_pRenderTargetView, g_pDSV);
```

- 기존에 Render Target View를 만든 후 호출하는 `OMSetRenderTargets` 함수는 지우고 아래쪽에서 다른 인자로 새로 호출한다

실습 - 깊이 버퍼 클리어

```
// D3DDevice.cpp

void Render()
{
    float clearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
    g_pImmediateContext->ClearRenderTargetView(g_pRenderTargetView, clearColor);
    g_pImmediateContext->ClearDepthStencilView(g_pDSV, D3D11_CLEAR_DEPTH, 1.0f, 0);
```

실습 - 결과

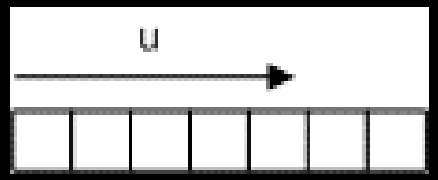


Direct3D11 스터디
Ch. 2-2 텍스쳐 매핑

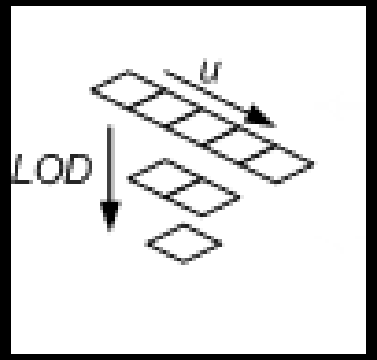
- 실습 - 텍스쳐 만들기

D3D11의 텍스쳐

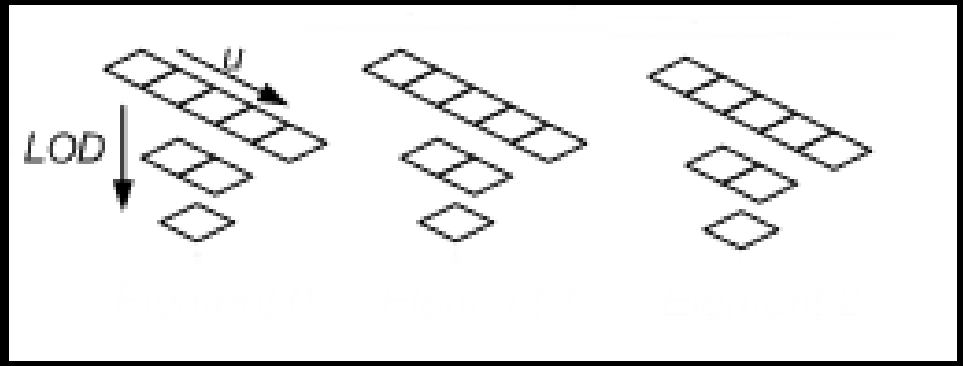
- 텍스쳐는 텍셀을 저장하기 위한 데이터의 모음임.
- 텍스쳐는 버퍼와 달리 샘플러에 의해 필터링될 수 있음
- 텍스쳐는 버퍼와 달리 밉맵(상위 레벨보다 2^n배만큼 작은 텍스쳐)을 가질 수 있음
- 텍스쳐는 형식에 따라서 여러 방법으로 압축되어 저장됨
- 종류 : 1D, 1D Array, 2D, 2D Array, 3D



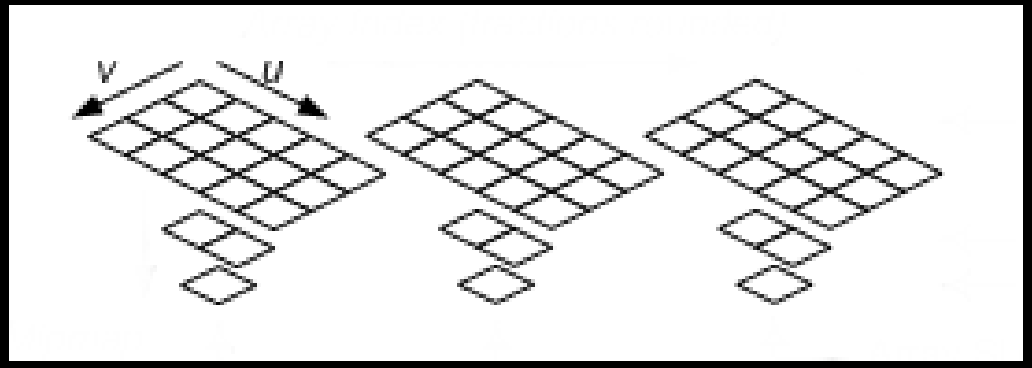
1D



1D with Mipmap

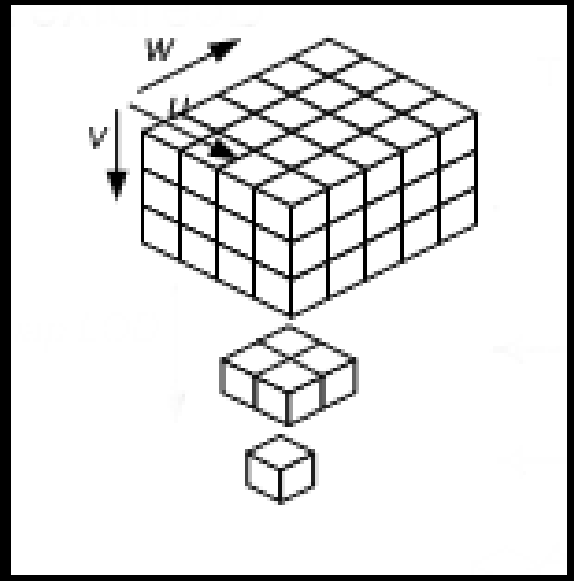


1D Array with Mipmap



2D Array with Mipmap

3D with Mipmap
(no 3D Array)



실습 - 텍스쳐

```
// D3DDevice.cpp 파일의 거의 제일 위쪽

ID3D11Texture2D* g_pTexture1 = NULL;
ID3D11ShaderResourceView* g_pSRV1 = NULL;
ID3D11SamplerState* g_pSamplerState = NULL;
```

```
// D3DDevice.cpp ClearD3D() 함수의 제일 위쪽

void ClearD3D()
{
    if (g_pImmediateContext != NULL) g_pImmediateContext->ClearState();

    if (g_pSamplerState != NULL) g_pSamplerState->Release();
    if (g_pSRV1 != NULL) g_pSRV1->Release();
    if (g_pTexture1 != NULL) g_pTexture1->Release();
```

실습 - 텍스쳐 만들기

```
// D3DDevice.cpp InitD3D() 함수의 맨 아래쪽 return true; 위

D3D11_TEXTURE2D_DESC descTexture1;
descTexture1.Width = 256;
descTexture1.Height = 256;
descTexture1.MipLevels = 1;
descTexture1.ArraySize = 1;
descTexture1.Format = DXGI_FORMAT_R32G32B32_FLOAT;
descTexture1.SampleDesc.Count = 1;
descTexture1.SampleDesc.Quality = 0;
descTexture1.Usage = D3D11_USAGE_DEFAULT;
descTexture1.BindFlags = D3D11_BIND_SHADER_RESOURCE;
descTexture1.CPUAccessFlags = 0;
descTexture1.MiscFlags = 0;
```

실습 - 텍스쳐 만들기

```
// D3DDevice.cpp InitD3D() 함수의 맨 아래쪽 return true; 위

static float arr[256][256][3];
for (int i = 0; i < 256; ++i)
    for (int j = 0; j < 256; ++j) {
        arr[i][j][0] = i / 256.0f;
        arr[i][j][1] = j / 256.0f;
        arr[i][j][2] = ((i - 128.0f) * (i - 128.0f) + (j - 128.0f) * (j - 128.0f))
            / (128.0f * 128.0f * 2.0f);
    }
D3D11_SUBRESOURCE_DATA initTexture1;
initTexture1.pSysMem = arr;
initTexture1.SysMemPitch = 256 * 3 * 4;
initTexture1.SysMemSlicePitch = 256 * 256 * 3 * 4;

hr = g_pd3dDevice->CreateTexture2D(&descTexture1, &initTexture1, &g_pTexture1);
if (FAILED(hr))
    return false;
```

실습 - 셰이더 리소스 뷰 만들기

```
// D3DDevice.cpp InitD3D() 함수의 맨 아래쪽 return true; 위

D3D11_SHADER_RESOURCE_VIEW_DESC descSRV1;
ZeroMemory(&descSRV1, sizeof(descSRV1));
descSRV1.Format = descTexture1.Format;
descSRV1.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
descSRV1.Texture2D.MipLevels = 1;

hr = g_pd3dDevice->CreateShaderResourceView(g_pTexture1, &descSRV1, &g_pSRV1);
if (FAILED(hr))
    return false;

g_pImmediateContext->PSSetShaderResources(0, 1, &g_pSRV1);
```

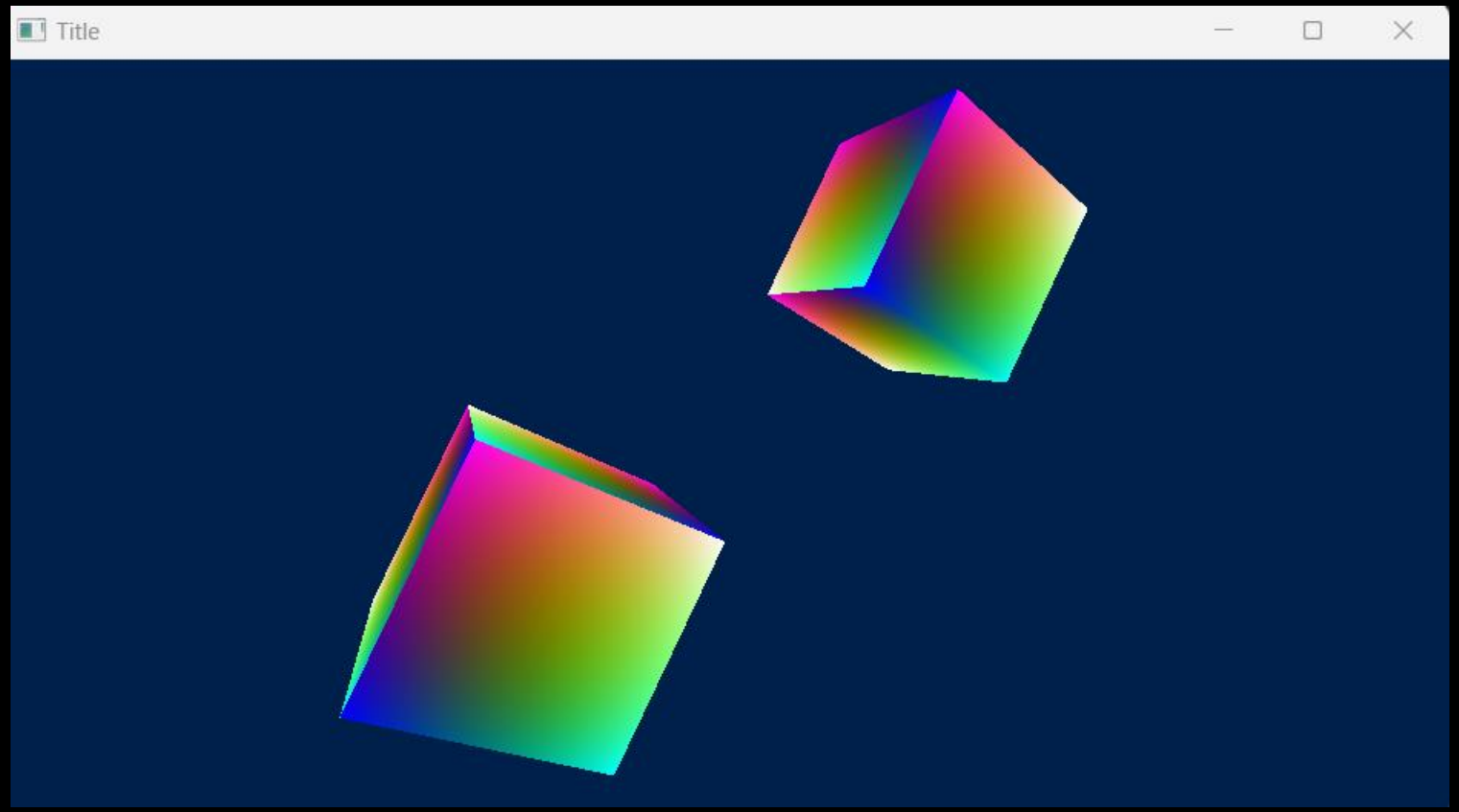
실습 - 샘플러 만들기

```
// D3DDevice.cpp InitD3D() 함수의 맨 아래쪽 return true; 위

D3D11_SAMPLER_DESC descSampler;
ZeroMemory(&descSampler, sizeof(descSampler));
descSampler.Filter = D3D11_FILTER_MIN_MAG_MIP_POINT;
descSampler.AddressU = D3D11_TEXTURE_ADDRESS_WRAP;
descSampler.AddressV = D3D11_TEXTURE_ADDRESS_WRAP;
descSampler.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;
descSampler.ComparisonFunc = D3D11_COMPARISON_NEVER;
descSampler.MinLOD = 0;
descSampler.MaxLOD = D3D11_FLOAT32_MAX;

hr = g_pd3dDevice->CreateSamplerState(&descSampler, &g_pSamplerState);
if (FAILED(hr))
    return false;

g_pImmediateContext->PSSetSamplers(0, 1, &g_pSamplerState);
```

실습 - 샘플러 만들기

```
// Shader1.fx 파일의 맨 아래쪽 PS() 수정

Texture2D texture1 : register(t0);
SamplerState samplerPoint : register(s0);

float4 PS(VS_Output input) : SV_Target
{
    return texture1.Sample(samplerPoint, input.UV);
}
```

실습 - 결과



Direct3D11 스터디
Ch. 2-2 조명 계산

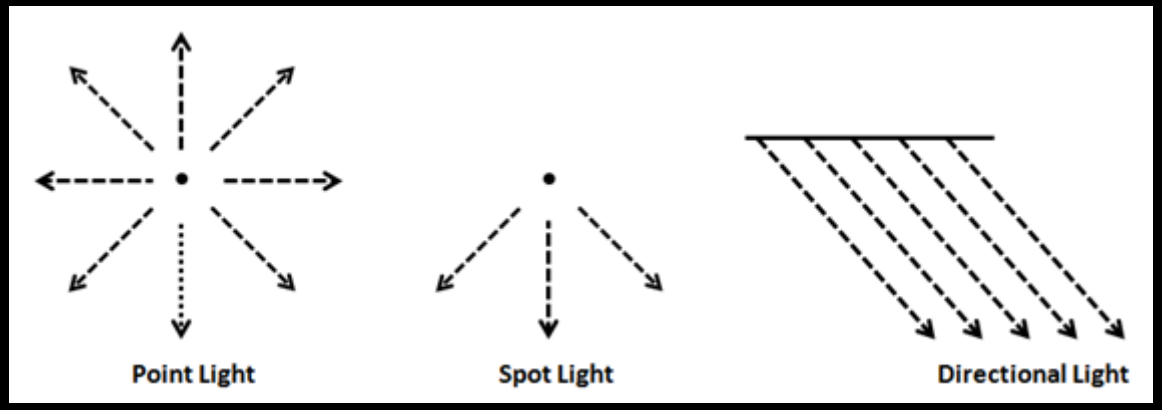
- 광원
- 람베르트 코사인 법칙
- 빛의 감쇠
- Phong Lighting Model
- 실습 - 조명 계산

광원

- 빛이 눈에 들어와 물체를 볼 수 있다
- 광원은 빛을 내뿜는다
- 빛은 파동과 입자의 성질을 동시에 갖지만 렌더링에서는 주로 빛을 입자로 가정

- 광원의 종류 : 점광(point), 점적광(spot), 평행광(지향, directional, parallel), 선 광원, 면 광원(area) 등등

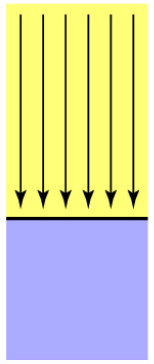
- 점광 : 한 점에서 모든 방향으로 빛이 나옴. 거리가 멀어질수록 세기가 약해짐.
- 점적광 : 특정한 방향으로만 비춰지는 점광. 거리뿐만 아니라 방향이 중심에서 멀수록 세기가 약해짐.
- 평행광 : 태양같이 매우 먼 거리의 강한 점광을 근사해 평행한 방향으로 비춰진다고 가정하는 광원.
  - 거리에 상관없이 세기가 일정함.



람베르트 코사인 법칙

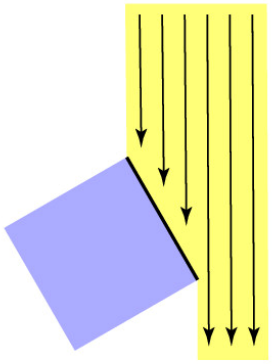
- 면에서 한 점이 받는 빛의 양은 그 면의 노멀 벡터와 점에서 빛까지의 벡터를 내적한 값에 비례한다

Lambert's Cosine Law



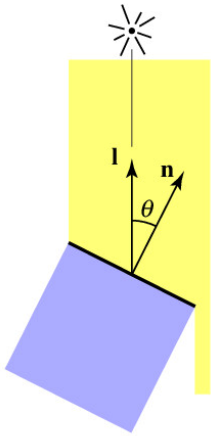
Top face of cube receives a certain amount of power

$$E = \frac{\Phi}{A}$$



Top face of 60° rotated cube receives half power

$$E = \frac{1}{2}\frac{\Phi}{A}$$

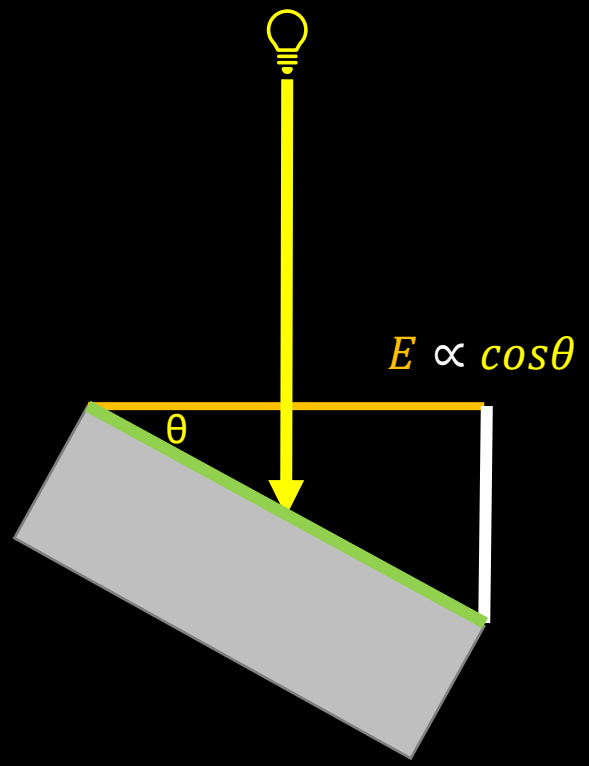


In general, power per unit area is proportional to $\cos\theta = l \cdot n$

$$E = \frac{\Phi}{A}\cos\theta$$

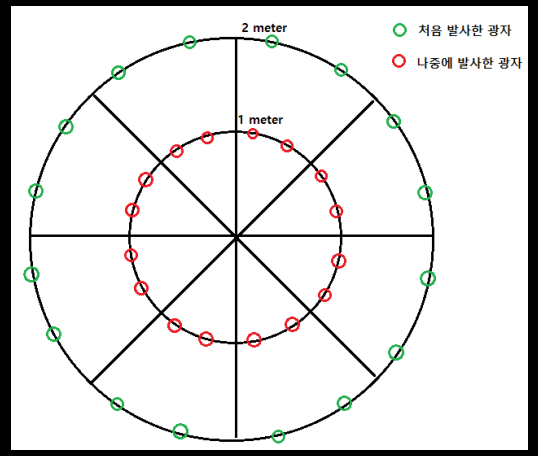
Irradiance at surface is proportional to cosine of angle between light direction and surface normal.

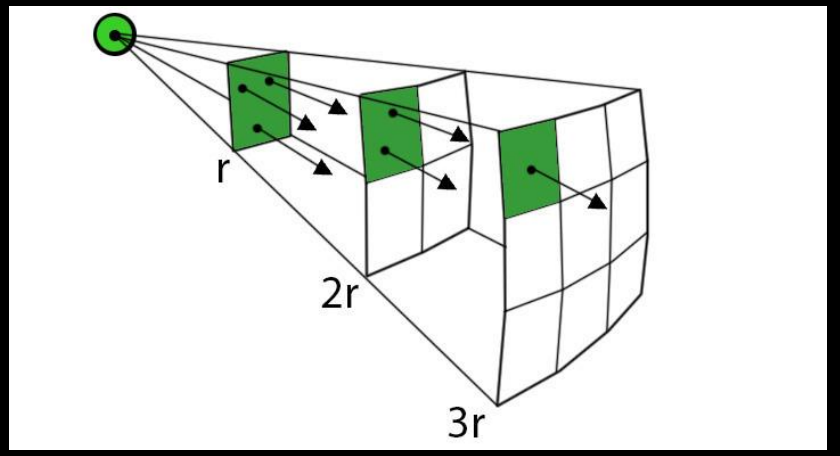
CS184/284A Ng & O'Brien



빛의 감쇠

- 점광원의 세기는 거리가 멀어질수록 약해진다.



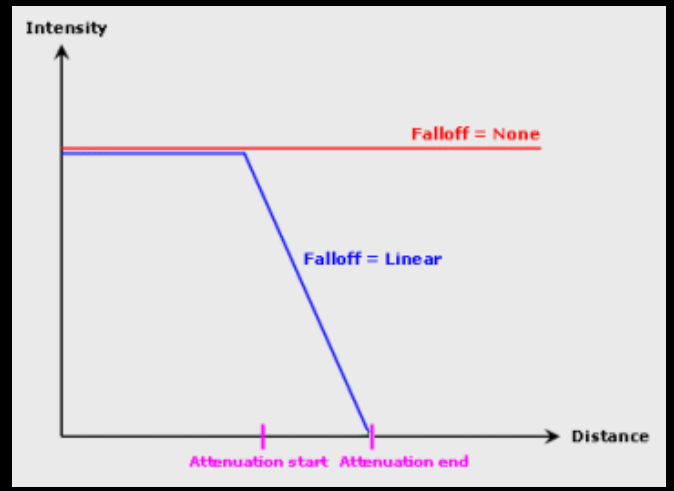


- 일반적으로 거리의 제곱에 반비례한다.

$$I \propto \frac{1}{r^2}$$

- 하지만 이 경우 지나치게 가까우면 값이 지나치게 커지고,
  거리가 무한이 되어도 0이 되지 않아 다른 함수로 근사해 사용하는 경우도 있다.

$$I \approx \text{saturate}\left(\frac{fall\_end - \text{distance}}{fall\_end - fall\_start}\right)$$

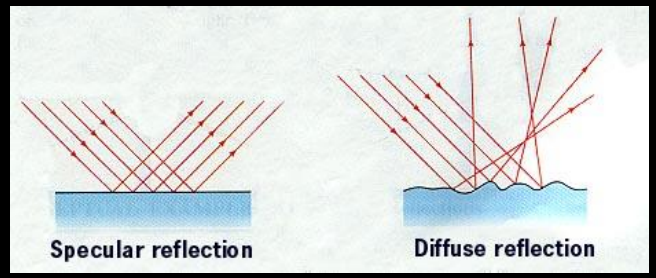


- 방향광의 경우 매우 먼 거리에서 온다고 가정하므로 거리의 차는 고려하지 않는다.

Phong Lighting Model

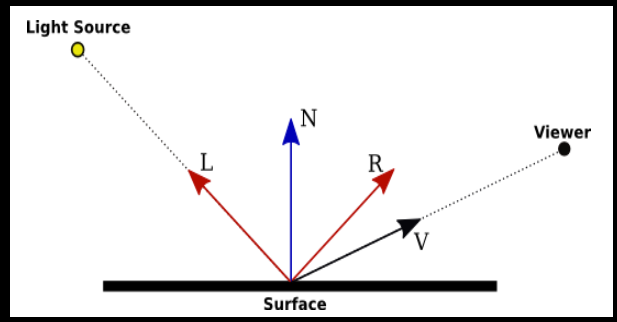
- 표면에 있는 점의 색을 계산하기 위한 모델

$$I = ambient + \sum_{m \in lights} (diffuse_m + specular_m)$$



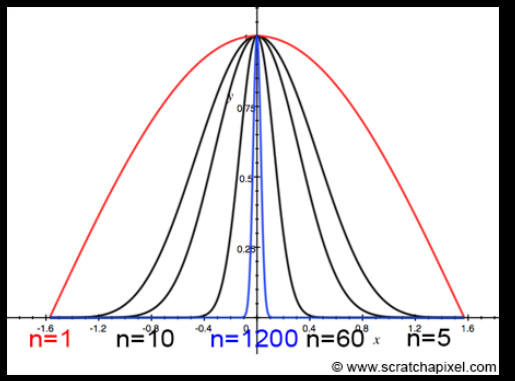
- specular (정반사광) : 빛이 매끄러운 표면에서 법선을 기준으로 반사

$$specular_m = k_s(\overrightarrow{R_m} \cdot \vec{V})^{\alpha} \qquad \overrightarrow{R_m} = -\overrightarrow{L_m} + 2(\overrightarrow{L_m} \cdot \vec{N})\vec{N}$$



- diffuse (난반사광) : 빛이 거친 표면에서 모든 방향으로 균등하게 반사

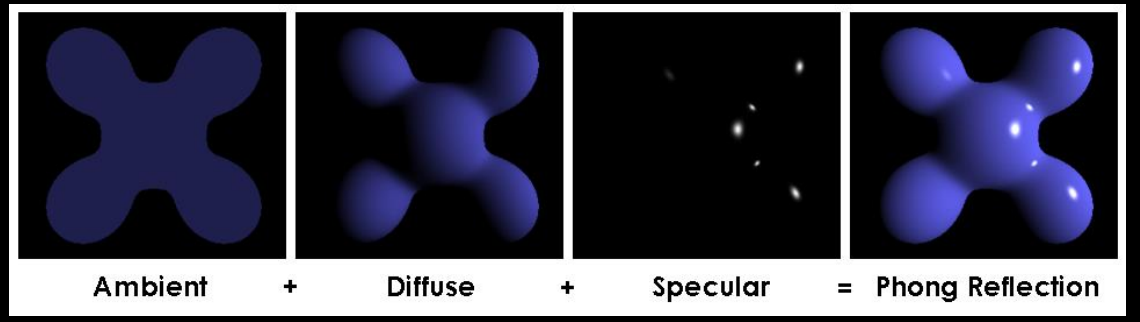
$$diffuse_m = k_d(\overrightarrow{L_m} \cdot \vec{N})$$



- ambient (주변광) : 주변에서 일정하게 들어온다고 가정하는 빛

$$ambient_m = k_a$$

$\vec{N}$: 표면의 법선 벡터 (길이 1)
$\overrightarrow{L_m}$: 빛에서 표면 위 점으로의 벡터 (길이 1)
$\vec{V}$: 표면 위 점에서 카메라로의 벡터 (길이 1)
k_a, k_d, k_s : 주변광 색 / 난반사 색 / 정반사 색
  - 보통 주변광/난반사 색은 텍스쳐에서, 정반사 색은 흰색.
α : 표면이 덜 거친 정도



실습 - 버퍼

```
// D3DDevice.cpp 파일의 거의 맨 위
using namespace DirectX;
struct VertexStructure1 {
    XMFLOAT3 pos;
    XMFLOAT2 uv;
    XMFLOAT3 normal;
};

struct VSConstBufferPerObject {
    XMMATRIX world;
};

struct VSConstBufferPerFrame {
    XMMATRIX view;
    XMFLOAT3 camPos;
};

struct VSConstBufferPerResize {
    XMMATRIX projection;
};

struct VSConstBufferPerScene {
    XMFLOAT3 lightPos;
    float padding1;
    XMFLOAT3 lightColor;
    float padding2;
};
```

```
// D3DDevice.cpp, 왼쪽 내용 아래쪽

ID3D11Buffer* g_pVSConstBufferPerObject = NULL;
ID3D11Buffer* g_pVSConstBufferPerFrame = NULL;
ID3D11Buffer* g_pVSConstBufferPerResize = NULL;
ID3D11Buffer* g_pVSConstBufferPerScene = NULL;

// D3DDevice.cpp ClearD3D() 중간

if (g_pVSConstBufferPerScene != NULL)
    g_pVSConstBufferPerScene->Release();
if (g_pVSConstBufferPerObject != NULL)
    g_pVSConstBufferPerObject->Release();
if (g_pVSConstBufferPerResize != NULL)
    g_pVSConstBufferPerResize->Release();
if (g_pVSConstBufferPerFrame != NULL)
    g_pVSConstBufferPerFrame->Release();
```

실습 - 노멀 데이터 추가

```
VertexStructure1 vertices[] =
{

// front
{ XMFLOAT3(-1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },
{ XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },
{ XMFLOAT3( 1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },
{ XMFLOAT3(-1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },
{ XMFLOAT3( 1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },
{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },

// back
{ XMFLOAT3( 1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },
{ XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },
{ XMFLOAT3(-1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },
{ XMFLOAT3( 1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },
{ XMFLOAT3(-1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },
{ XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },

// left
{ XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },
{ XMFLOAT3(-1.0f,  1.0f, -1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },
{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },
{ XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },
{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },
{ XMFLOAT3(-1.0f, -1.0f,  1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },

// right
{ XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },
{ XMFLOAT3( 1.0f,  1.0f,  1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },
{ XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },
{ XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },
{ XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },
{ XMFLOAT3( 1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

// down
{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },
{ XMFLOAT3( 1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },
{ XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },
{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },
{ XMFLOAT3( 1.0f, -1.0f,  1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },
{ XMFLOAT3(-1.0f, -1.0f,  1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },

// down
{ XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },
{ XMFLOAT3(-1.0f,  1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },
{ XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },
{ XMFLOAT3( 1.0f,  1.0f, -1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },
{ XMFLOAT3(-1.0f,  1.0f,  1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },
{ XMFLOAT3( 1.0f,  1.0f,  1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },

};
```

실습 - 인풋 레이아웃 수정

```
// D3DDevice.cpp InitD3D() input layout 부분

D3D11_INPUT_ELEMENT_DESC layouts[3];
UINT numElements = ARRAYSIZE(layouts);
layouts[0].SemanticName = "POSITION";
layouts[0].SemanticIndex = 0;
layouts[0].Format = DXGI_FORMAT_R32G32B32_FLOAT;
layouts[0].InputSlot = 0;
layouts[0].AlignedByteOffset = 0;
layouts[0].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
layouts[0].InstanceDataStepRate = 0;

layouts[1].SemanticName = "TEXCOORD";
layouts[1].SemanticIndex = 0;
layouts[1].Format = DXGI_FORMAT_R32G32_FLOAT;
layouts[1].InputSlot = 0;
layouts[1].AlignedByteOffset = sizeof(XMFLOAT3);
layouts[1].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
layouts[1].InstanceDataStepRate = 0;

layouts[2].SemanticName = "NORMAL";
layouts[2].SemanticIndex = 0;
layouts[2].Format = DXGI_FORMAT_R32G32B32_FLOAT;
layouts[2].InputSlot = 0;
layouts[2].AlignedByteOffset = sizeof(XMFLOAT3) + sizeof(XMFLOAT2);
layouts[2].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
layouts[2].InstanceDataStepRate = 0;
```

실습 - 픽셀 셰이더에도 프레임별 상수 버퍼 전달

```
// D3DDevice.cpp InitD3D() 버퍼 만드는 부분

g_pImmediateContext->VSSetConstantBuffers(0, 1, &g_pVSConstBufferPerFrame);
g_pImmediateContext->PSSetConstantBuffers(0, 1, &g_pVSConstBufferPerFrame);
```

실습 - 장면별 상수 버퍼 만들기

```
// D3DDevice.cpp InitD3D() 버퍼 만드는 부분(전 슬라이드 내용에 이어서)

ZeroMemory(&bd, sizeof(bd));
bd.Usage = D3D11_USAGE_DEFAULT;
bd.ByteWidth = sizeof(VSConstBufferPerScene);
bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
bd.CPUAccessFlags = 0;

VSConstBufferPerScene vsConstBufferPerScene{
    { 0.0f, 3.0f, -2.0f }, 0.0f,
    { 1.0f, 1.0f, 1.0f }, 0.0f,
};
ZeroMemory(&InitData, sizeof(InitData));
InitData.pSysMem = &vsConstBufferPerScene;

hr = g_pd3dDevice->CreateBuffer(&bd, &InitData, &g_pVSConstBufferPerScene);
if (FAILED(hr))
    return false;

g_pImmediateContext->VSSetConstantBuffers(3, 1, &g_pVSConstBufferPerScene);
g_pImmediateContext->PSSetConstantBuffers(3, 1, &g_pVSConstBufferPerScene);
```

실습 - 프레임별 상수 버퍼에 카메라 위치 넣기

```
// D3DDevice.cpp Render()

VSConstBufferPerFrame vsConstBufferPerFrame;
vsConstBufferPerFrame.view = XMMatrixMultiply(
    XMMatrixTranslation(-camPos[0], -camPos[1], -camPos[2]),
    XMMatrixTranspose(XMMatrixRotationRollPitchYaw(
        camPitchYawRoll[0], camPitchYawRoll[1], camPitchYawRoll[2]))
);
vsConstBufferPerFrame.camPos = XMFLOAT3(camPos[0], camPos[1], camPos[2]);
```

실습 - 셰이더

```
// Shader1.fx 맨 위

cbuffer VSConstBufferPerObject : register(b2)
{
    float4x4 MatWorld;
};

cbuffer VSConstBufferPerFrame : register(b0)
{
    float4x4 MatView;
    float3 CamPos;
};

cbuffer VSConstBufferPerResize : register(b1)
{
    float4x4 MatProj;
};

cbuffer VSConstBufferPerScene : register(b3)
{
    float3 LightPos;
    float3 LightColor;
}
```

```
// Shader1.fx 왼쪽 내용에 이어서

struct VS_Input
{
    float3 Pos : POSITION;
    float2 UV : TEXCOORD;
    float3 Norm : NORMAL;
};

struct VS_Output
{
    float4 Pos : SV_POSITION;
    float2 UV : TEXCOORD;
    float3 Norm : NORMAL;
    float3 Diffuse : DIFFUSE;
    float3 WorldPos : POSITION;
};
```

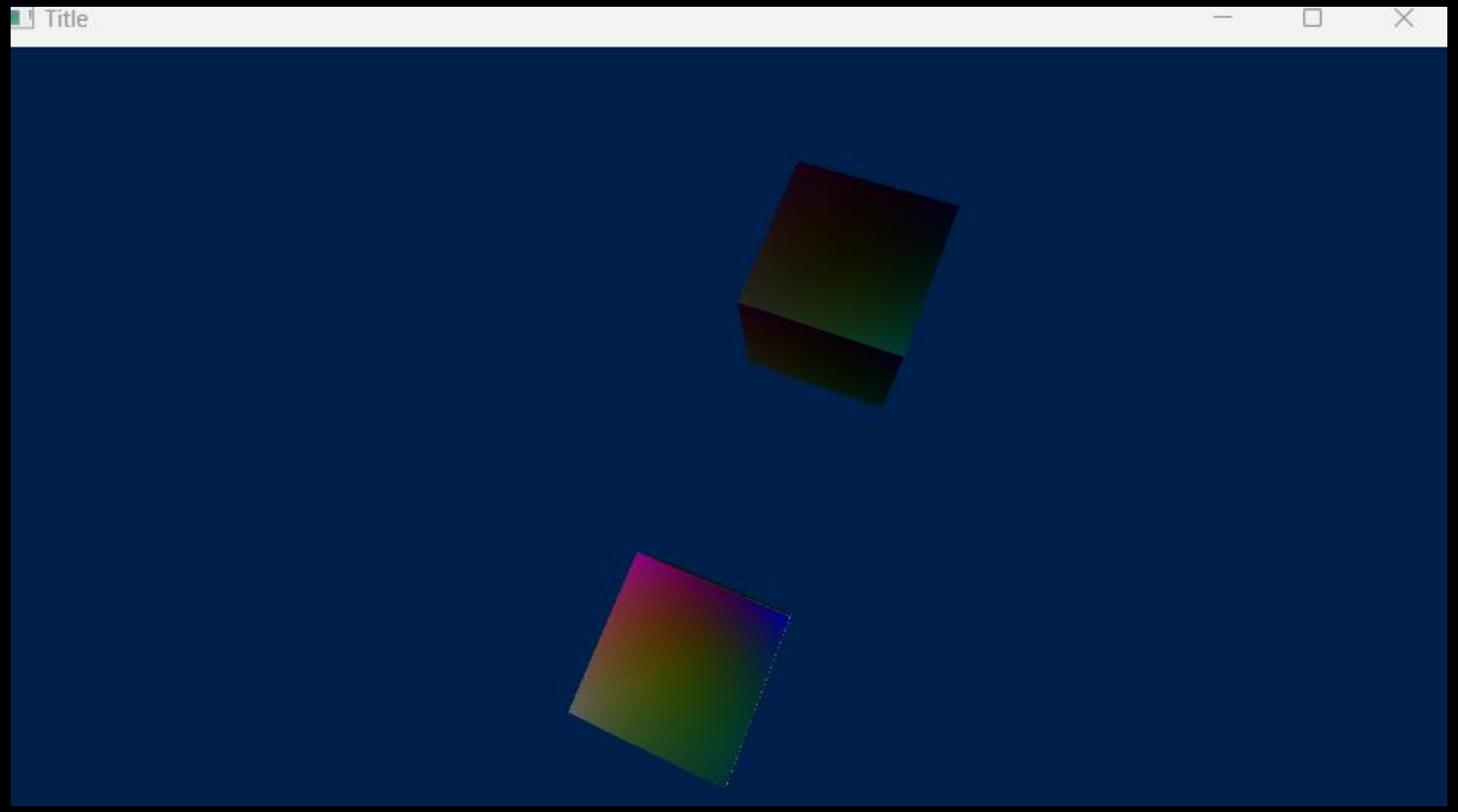
실습 - 버텍스 셰이더

```
// Shader1.fx 전 슬라이드 내용에 이어서 (VS의 내용 전체를 이걸로 바꾸기)

VS_Output VS(VS_Input input)
{
    VS_Output output;
    output.Pos = float4(input.Pos, 1.0f);
    output.Pos = mul(MatWorld, output.Pos);
    output.WorldPos = output.Pos.xyz;

    float3 lightVec = LightPos - output.WorldPos;
    float lightDist = length(lightVec);
    float lightIntensity = saturate((5.0f - lightDist) / (5.0f - 2.0f));

    output.Pos = mul(MatView, output.Pos);
    output.Pos = mul(MatProj, output.Pos);
    output.UV = input.UV;
    output.Norm = input.Norm;
    output.Diffuse = LightColor * lightIntensity * max(0, dot(normalize(lightVec), input.Norm));

    return output;
}
```

실습 - 픽셀 셰이더

```
// Shader1.fx 전 슬라이드 내용에 이어서 (PS의 내용 전체를 이걸로 바꾸기)

Texture2D texture1 : register(t0);
SamplerState samplerPoint : register(s0);

float4 PS(VS_Output input) : SV_Target
{
    float3 lightVec = input.WorldPos - LightPos;
    float lightDist = length(lightVec);
    float lightIntensity = saturate((10.0f - lightDist) / (10.0f - 2.0f));
    float3 reflected = reflect(lightVec, normalize(input.Norm));

    float3 specular = LightColor * lightIntensity * pow(max(0,
        dot(normalize(reflected), normalize(CamPos - input.WorldPos))), 16.0f);
    float3 ambient = float3(0.1f, 0.1f, 0.1f);

    float4 sampledColor = texture1.Sample(samplerPoint, input.UV);
    float3 color = (ambient + input.Diffuse) * sampledColor.xyz + specular;

    return float4(color, 1.0f);
}
```

실습 - 결과



끝!

- 3D 정육면체를 렌더링했다
- 행렬을 만들어 3차원 변환을 했다
- 카메라 이동을 만들었다
- z버퍼를 만들어 깊이 테스트를 했다
- 정육면체에 텍스쳐를 입혔다
- 조명을 계산했다

- 평가
  - D3D11의 기본 내용은 다 다룸
  - 심화 주제가 남아있지만 (인스턴싱, 디퍼드 렌더링, 기하 셰이더, 테셀레이션, 컴퓨트 셰이더, 멀티스레드 렌더링, 타일드 리소스 등등 할 건 넘친다) 이 정도만 해도 초반 진입장벽은 넘은 상태라고 생각
  - 필요한 자원을 생각하고, 구조체를 채우고, 자원을 할당하고, 파이프라인에 묶는 반복되는 패턴
  - 자원의 스펙은 어떻게 되는지, 구조체를 어떻게 채울지는 공식 문서에 다 나와있다
    https://learn.microsoft.com/en-us/windows/win32/direct3d11/atoc-dx-graphics-direct3d-11
  - 이제 많아진 코드들을 쉽게 사용할 수 있도록 잘 추상화시키는 걸 해볼 수 있다

Direct3D11 스터디
2회차 - 3D, Buffers, Light!

숭실대학교 겜마루 27기 컴퓨터학부 22학번 김민규

mkkim0612@naver.com

- 실습 - 3D 렌더링
- 실습 - 텍스쳐 매핑
- 실습 - 조명 계산