3D게임프로그래밍
-CHAPTER3-

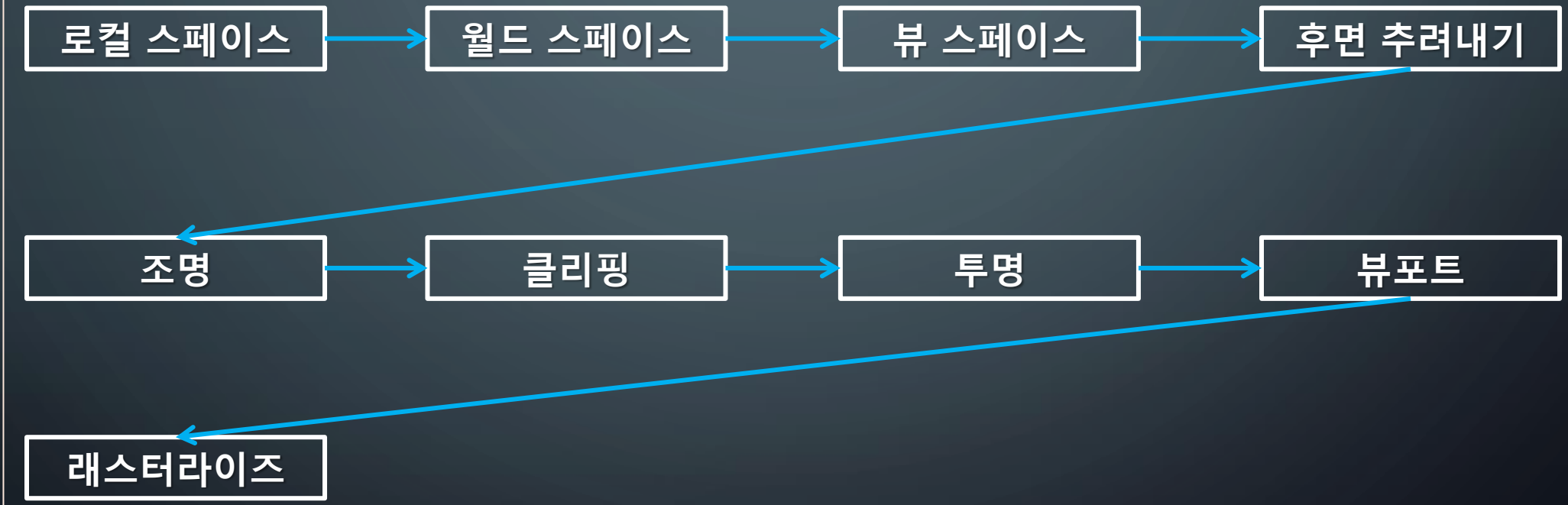
SOULSEEK

목차

랜더링 파이프라인

1. 랜더링 파이프라인

기하학적으로 **3D** 장면을 구성하고 가상 카메라를 설정한 뒤에는 모니터에 **2D** 표현을 만들어내는 과정을 수행하는 과정을 말한다.



1. 랜더링 파이프라인

로컬 스페이스
- 자신이 중심이 되는 좌표를 말하며 오브젝트 하나하나가 가지고 있는 모델링의 좌표를 말한다.
- 월드 상의 다른 물체와의 관계를 고려 하지 않는다.

월드 스페이스
- 로컬 스페이스에서 정의된 오브젝트를 월드 좌표상으로 옮겨 하나의 장면을 구성하는 것을 말한다.
- 이동, 회전, 크기 변형 등을 포함하는 월드 변환이라는 작업을 진행 한다.

```
D3DXMATRIXA16 matWorld;

UINT iTime = timeGetTime() % 1000; // float 연산의 정밀도를 위해서 1000으로 나머지 연산한다.

FLOAT fAngle = iTime * (2.0f * D3DX_PI) / 1000.0f; // 1000밀리초마다 한 바퀴씩(2 * pi) 회전
                                                   애니메이션 행렬을 만든다.

D3DXMatrixRotationY(&matWorld, fAngle); // Y축을 회전축으로 회전 행렬을 생성한다.

g_pd3dDevice->SetTransform(D3DTS_WORLD, &matWorld); // 생성한 회전 행렬을 월드 행렬로
                                                    디바이스에 설정한다.
```

1. 랜더링 파이프라인

뷰 스페이스(카메라)

- 카메라의 위치에서 월드 좌표상을 바라보게 하고 그 바라보는 관점에 따른 변환을 해줘야 하는 공간

```
D3DXMATRIX *D3DXMatrixLookAtLH(
                                    D3DXMATRIX* pOut              // 결과 행렬을 받을 포인터
                                    CONST D3DXVECTOR3* pEye,      // 월드 내의 카메라 위치
                                    CONST D3DXVECTOR3* pAt,       // 월드 내의 카메라가 보는 지점
                                    CONST D3DXVECTOR3* pUp);      // 월드의 업 벡터(0, 1, 0)

Device->SetTransform(D3DTS_VIEW, &v); // 실제 매트릭스를 적용 시켜준다.
```

후면 추려내기(Culling)

- 실제 현재 카메라가 보고 있는 면은 전면이 된다. 물체의 후면에 대해서는 랜더링을 해줘야 할 필요가 없기 때문에 후면에 대한 랜더링을 하지 않게 된다.

```
Device->SetREnderState(D3DRS_CULLMODE, value);
```

옵션 플래그

D3DCULL_NONE – 후면 추려내기를 완전히 끈다.
D3DCULL_CW – 시계 방향 두르기를 가진 삼각형을 추려낸다.
D3DCULL_CCW – 시계 반대 방향 두르기를 가진 삼각형을 추려낸다. 디폴트 상태.

1. 랜더링 파이프라인

클리핑

- 카메라가 바라보는 시야각에 의해 보여질 부분과 보여지지 않는 부분이 있으며 보여지지 않는 부분은 과감히 랜더링 되지 않아도 되기 때문에 이를 구분하고 처리하는 과정이다.

완전한 내부 – 삼각형이 완전히 절두체 내부에 위치하면 그대로 보존되어 다음 단계로 진행한다.
완전한 외부 – 삼각형이 완전히 절두체 외부에 위치하면 추려 내어진다.
부분적 내부(부분적 외부) – 삼각형이 부분적으로 절두체 내부에 위치하면 삼각형을 두 개의 부분으로 분리한다. 절두체 내부의 부분은 보존되며, 나머지는 추려 내어진다.

투영

- 시야범위에 있는 오브젝트들을 최소 범위와 최대 범위 사이에 존재하는 것들끼리 구분하여 원근감에 따라 깊이 관계를 판단하여 2D 좌표상으로 옮길 준비를 하는 것이다.

```
D3DXMATRIX* D3DXMatrixPerspectiveFovLH(D3DXMATRIX* pOut,      // 투영 행렬을 리턴
                                       FLOAT fovY,           // 시야각의 수직 영역(라디안)
                                       FLOAT Aspect,         // 종횡비 = 너비 / 높이
                                       FLOAT zn,             // 가까운 평면까지의 거리
                                       FLOAT zf);            // 먼 평면까지의 거리
```

MATRIX TRANSFORM

2. MATRIX TRANSFORM

컬링과 광원 설정

// 컬링 기능을 끈다, 삼각형의 앞면, 뒷면을 모두 랜더링한다.
g_pd3dDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);

//정점에 색깔값이 있으므로 광원 기능을 끈다.
g_pd3dDevice->SetRenderState(D3DRS_LIGHTING, FALSE);

행렬 설정

월드 스페이스
D3DXMATRIXA16 matWorld;

UINT iTime = timeGetTime() % 1000; // **float** 연산의 정밀도를 위해서 **1000**으로 나머지 연산한다.

FLOAT fAngle = iTime * (2.0f * D3DX_PI) / 1000.0f; // **1000**밀리초마다 한 바퀴씩**(2 * pi)** 회전 애니메이션 행렬을 만든다.

D3DXMatrixRotationY(&matWorld, fAngle); // **Y**축을 회전축으로 회전 행렬을 생성한다.

g_pd3dDevice->SetTransform(D3DTS_WORLD, &matWorld); // 생성한 회전 행렬을 월드 행렬로 디바이스에 설정한다.

2. MATRIX TRANSFORM

뷰 스페이스(카메라)

```
//뷰 행렬(카메라)을 정의하기 위해서는 3가지 값이 필요하다.
D3DXVECTOR3 vEyePt(0.0f, 3.0f, -5.0f); //눈의 위치(0.0f, 3.0f, -5.0f)
D3DXVECTOR3 vLookatPt(0.0f, 0.0f, 0.0f); // 눈이 바라보는 위치(0.0f, 0.0f, 0.0f)
D3DXVECTOR3 vUpVec(0.0f, 1.0f, 0.0f); // 윗 방향을 나타내는 상방 벡터(0.0f, 1.0f, 0.0f)

//뷰 행렬(카메라)
D3DXMATRIXA16 matView;
D3DXMatrixLookAtLH(&matView, &vEyePt, &vLookatPt, &vUpVec);// 3가지 Point를 이용해 뷰 행렬을 생성한다.

g_pd3dDevice->SetTransform(D3DTS_VIEW, &matView); // 생성한 뷰 행렬을 디바이스에 설정한다.
```

투영

```
// 프로젝션 행렬을 정의하기 위해서는 시야각(FOV = Field Of View)과 종횡비(aspect ratio), 클리핑 평면의 값이 필요하다.
D3DXMATRIXA16 matProj;
//첫 번째 : 설정될 행렬
//두 번째 : 시야각
//세 번째 : 종횡비
//네 번째 : 근접 클리핑
//다섯 번째 : 원거리 클리핑
D3DXMatrixPerspectiveFovLH(&matProj, D3DX_PI / 4, 1.0f, 1.0f, 100.0f);
g_pd3dDevice->SetTransform(D3DTS_PROJECTION, &matProj); // 생성한 프로젝션 행렬을 디바이스에 설정.
```

INDEX BUFFER

3. INDEX BUFFER

- 정점의 인덱스를 보관하기 위한 전용 버퍼.
- 여러 번 나열하는 것보다 메모리 소모량이 적다.
- 자주 사용하는 정점을 캐시로 저장해서 더 높은 효율을 낼 수 있다.

인덱스 버퍼 구성

LPDIRECT3DINDEXBUFFER9 g_pIB = NULL; - 인덱스를 보관할 인덱스 버퍼

정점들을 순서대로 인덱스와 시킬 구조체가 필요하다.

```
struct MYINDEX
{
    WORD _0, _1, _2; //DWORD도 현대에는 가능하다 예전에는 구형 그래픽에선 16비트가 최고 였기
                     때문이다 이와 같이 항상 하드웨어 성능을 고려하는 코딩이 되어야 한다.
}
```

3. INDEX BUFFER

인덱스 버퍼 생성

```
HRESULT InitIB()
{
	MYINDEX indices[] =
	{
		{0, 1, 2}, {0, 2, 3}, // 윗면
		{4, 5, 6}, {4, 7, 6}, // 아랫면
		{0, 3, 7}, {0, 7, 4}, // 왼면
		{1, 5, 6}, {1, 6, 2}, // 오른면
		{3, 2, 6}, {3, 6, 7}, // 앞면
		{0, 4, 5}, {0, 5, 1}, // 뒷면
	};

	//인덱스 버퍼 생성
	//D3DFMT_INDEX16은 인덱스의 단위가 16비트라는 것.
	//MYINDEX 구조체에서 WORD형으로 선언으므로 D3DFMT_INDEX16을 사용한다.
	if (FAILED(g_pd3dDevice->CreateIndexBuffer(12 * sizeof(MYINDEX), 0, D3DFMT_INDEX16, D3DPOOL_DEFAULT, &g_pIB,
                                                NULL)))
	{
		return E_FAIL;
	}

	//인덱스 버퍼를 값으로 채운다.
	void* pIndices;
	//인덱스 버퍼의 Lock() 함수를 호출하여 포인터를 얻어와서 정보를 채운다.
	if (FAILED(g_pIB->Lock(0, sizeof(indices), (void**)&pIndices, 0)))
	{
		return E_FAIL;
	}

	memcpy(pIndices, indices, sizeof(indices));

	g_pIB->Unlock();

	return S_OK;
}
```

3. INDEX BUFFER

학습과제

- 정점들을 생성하는 예제에서 연습 했던 것을 인덱스로 표현해보자.
- 2개 이상의 육면체를 인덱스로 생성해서 행렬을 적용해 보자
- 뷰 행렬의 값을 조정해서 어떻게 달라지는지 알아보자.