

1.15

▼ DX

(복습)

- ▼ 교환 사슬에서 얻은 텍스처 인터페이스를 사용하기 위해 필요한 것 + 특성 2
텍스처를, 파이프 라인의 렌더 대상으로 묶기 위한 **자원 뷰**.
즉, **렌더 대상 뷰..**라는 자원 뷰가 필요하다

다른 대부분의 객체들처럼

- 이 객체도 장치 인터페이스를 통해 생성하고
- 객체로 어떤 작업을 수행할 때에는, 장치 문맥을 거치게 된다.

- ▼ 응용 프로그램 메시지 루프 : 3가지 단위

- 남아있는 메시지 처리 + 갱신 + 렌더링

- ▼ 3번째 관련 변수 2개 + 함수 1개 + 특성 3가지

```
ID3D11RenderTargetView* RenderTargetViews[1] = { pView };
ID3D11DepthStencilView* DepthTargetView = pDepthView;

pContext->OMSetRenderTargets( 1, RenderTargetViews, DepthTargetView );
```

응용 프로그램의 상태를 갱신한 후에는 (즉, Update ? 를 진행한 후에는)
현재 장면을 사용자가 볼 수 있도록, 표시하는 작업
즉, 렌더링 작업을 수행한다.

- 이 과정은, 렌더 대상뷰와 깊이-스텐실 뷰를 파이프라인에 묶는 것으로 시작해서

- 이렇게 연결해야만, **렌더링 파이프라인의 연산 결과가**

깊이 스템실 버퍼, 렌더 대상 버퍼에 전달된다.

- 이 연결과정은 파이프라인을 조작하는 것이므로, 장치 문맥을 통해서 수행된다.

▼ 3번째 3개의 과정

▼ 1번째 :

- OMSetRenderTargets : 그린다

▼ 2번째

- 비운다

```

if ( pDepthStencilView )
{
    m_pContext->ClearDepthStencilView( pDepthStencilView,
                                         D3D11_CLEAR_DEPTH, depth, stencil );
}

// 깊이 · 스템실 뷰를 해제한다.
SAFE_RELEASE( pDepthStencilView );

```

▼ 3번째

```
pSwapChain->Present( 0, 0 );
```

목록 1.13. 교환 버퍼 자원의 내용을 창에 표시하는 코드.

- 보여준다
- 렌더링 결과를 담은 버퍼의 내용을 창의 클라이언트 영역에 표시한다.

- ▼ 모든 자원 생성 메서드의 공통 3가지 매개변수
 - 1번째 : 자원 서술 = 자원 생성에 대한 모든 옵션 지정
 - 2번째 : D3D11_SUBRESOURCE_DATA 구조체를 가리키는 포인터
 - 3번째 : 자원 종류에 맞는 자원 인터페이스를 가리키는 포인터의 포인터 = 자원 생성이 성공하면, 해당 자원을 가리키는 포인터가 이 매개변수에 설정된다.

- ▼ 자원 서술 구조체

- ▼ 자원 서술 구조체 필드 첫번째

```
enum D3D11_USAGE {  
    D3D11_USAGE_DEFAULT,  
    D3D11_USAGE_IMMUTABLE,  
    D3D11_USAGE_DYNAMIC,  
    D3D11_USAGE_STAGING  
}
```

- 용도 서술

- ▼ 설명

응용 프로그램이 자원을 어떤 용도로 사용할 것인가

자원은 컴퓨터 안의 어딘가에 있는 메모리 블록이다.

- 1) 자원이 존재하는 “메모리”는 비디오 카드 메모리 일수도 있고, 시스템의 주 메모리일수도 있다.
- 2) Direct3D 실행 시점 모듈이, 자원을 비디오 카드 메모리, 시스템 주 메모리 사이에서 이동시킬 수도 있으므로, 실행시점 모듈 / 비디오 하드웨어 사용자 구동기가, 자원을 최적의 장소에 두고 내부적으로 자원을 효율적으로 처리할 수 있게 하기 이 필드를 지정

▼ 첫번째 필드 4가지 변수 및 특성

자원 용도	DEFAULT	DYNAMIC	IMMUTABLE	STAGING
GPU 읽기	예	예	예	예
GPU 쓰기	예			예
CPU 읽기				예
CPU 쓰기		예		예

▼ IMMutable 3가지

- 이 용도를 지정해서 생성한 자원은 오직 GPU만 읽을 수 있고, CPU는 아예 접근할 수 없다
- GPU와 CPU 모두 이 자원에 기록하지 못한다
- 정적인 상수 버퍼, 정점 버퍼, 색인 버퍼가 예시

▼ Default 4가지

- gpu의 읽기쓰기는 허용
- cpu 접근은 모두 거부
- 렌더 대상 텍스쳐 (gpu의 렌더링 결과가 기록되고, gpu가 다시 읽어들인다)
- 스트림 출력 정점 버퍼 (gpu 처리 결과가 기록, gpu가 렌더링을 위해 읽어 \| 들임)

▼ Dynamic 3가지

- 동적 용도 = cpu가 자원의 내용을 생산하고, gpu가 그것을 소비한다
- **프레임마다 바뀔 수 있는 렌더링 자료 (변환 행렬)** 을 프로그램 가능 셰이더 단계에 공급하기 위한 **상수 버퍼**
- cpu가 자원의 내용을 읽어들이지는 못한다. 즉, 자료는 cpu에서 gpu로 한방향으로만 흘러간다.

▼ Staging 1

예비 용도

- 원하는 자원을 gpu로 조작하고, 그것을 또 다른 예비 용도 자원에 복사하고, cpu에서 읽어들이는 것이 기본적인 사용 패턴

▼ 두번째 필드 3가지

- cpu 접근 플래그 : 자원의 용도를 지정했다면 자원에 대한 cpu 접근 방식을 지정해야 한다.
- 총 2가지

```
enum D3D11_CPU_ACCESS_FLAG {  
    D3D11_CPU_ACCESS_WRITE,  
    D3D11_CPU_ACCESS_READ  
}
```

- 용도 플래그와 관련, staging // dynamic의 경우, write 이 가능하지만, 그외에는 read로 지정해야 한다.

▼ 세번째 필드 특성 2 + 종류 8가지 + 각 종류의 특성

- 연결 플래그 = bind flag
- 자원을 파이프라인의 어디에 연결할 수 있는지

```
enum D3D11_BIND_FLAG {
    D3D11_BIND_VERTEX_BUFFER,
    D3D11_BIND_INDEX_BUFFER,
    D3D11_BIND_CONSTANT_BUFFER,
    D3D11_BIND_SHADER_RESOURCE,
    D3D11_BIND_STREAM_OUTPUT,
    D3D11_BIND_RENDER_TARGET,
    D3D11_BIND_DEPTH_STENCIL,
    D3D11_BIND_UNORDERED_ACCESS
}
```

1번째 2번째 : 정점 버퍼와 색인 버퍼 —> 파일 파인에 기하구조 자료를 공급하기 위한 입력 조립기 단계에 부착할 자원에 쓰인다

6,7 번째 = 렌더 대상 flag, 깊이 스텐실 버퍼 flag 는 파이프 라인의 렌더링 결과를 받는 출력 병합기 단계에 연결할 자원을 위한 것

5번째 스트림 출력 flag = 파이프라인으로 부터의 출력을 위한 것이지만, 래스터화된 이미지 자료가 아니라, 기하구조 자료를 받는다는 점이 다르다.

그외 constant, shader_resourect, unordred_access 는 자원을 프로그램 가능 세이더 단계들 전부 또는 일부에 연결해서 세이더 프로그램 안에서 사용할 수 있다는 것을 의미한다.

핵심 : 원하는 파이프라인 연결 지점에 맞는 적절한 연결 플래그 설정하는 것이 중요하다

▼ 3.3.1 입력 조립기 단계의 파이프라인 입력

▼ 연결 해제 방법 및 중요성

방법 : 어떤 정점 버퍼나 색인 버퍼를 파이프라인에 연결한 후에, 해당 입력 슬롯에 **NULL 포인터** 값을 설정해서 같은 메서드를 호출하면 연결이 해제된다.

중요성 : 여러 그리기 호출들 사이에서 파이프라인을 재구성할 때

이런 해제가 더 중요해진다. **다중 정점 버퍼 구성**을 이용해서 **파이프라인을 실행**한 후, 그 다음 그리기 호출에서 더 적은 수의 정점 버퍼들을 사용한다면, 더 이상 쓰지 않는

슬롯들에 대해 **NULL 포인터**를 지정해서, 해당 버퍼들을 반드시 떼어내야 한다.

▼ 3.3.2 입력 조립기 단계의 상태 구성

▼ 입력 배치

Input Layout

▼ 역할 2

입력 조립기가 정점들을 생산하는데 사용하는 청사진 같은 것이다.

입력 조립기에 동시에 여러개의 정점 버퍼들을 묶을 수 있고, 각 버퍼에 정점의

다른 특성들을 지정할 수 있는 상황에서, 어떤 버퍼에 어떤 정점 특성(위치)이 들어 있는지를 입력 조립기에 알려줄 필요가 있다.

또한, 그 특성들을 어떤 순서로 조립(순서)해서 정점 자료를 완성시켜야 하는지도 알려주어야 한다.

▼ 3.3.4 입력 조립기 단계의 파이프라인 출력

▼ 입력 조립기가 출력할 수 있는, 3가지 시스템 값 의미소들 + 특성 1

- SV_VertexID
- SV_PrimitiveID
- SV_InstanceID

이 시스템 값 의미소들은 **응용 프로그램이**

제공하는 것이 아니라, 입력 조립기가 출력 자료 스트림을 만드는 과정에서 생성하는 것이다.

▼ 각각의 특성

- SV_VertexID : 부호 없는 정수, 파이프 라인 이후 단계들에서 개별 정점을

간단하게 식별하는 수단으로 쓰인다.

- SV_PrimitiveID : 기본 도형 스트림의 각 기본도형을 식별할 수 있다

정점 세이더는 기본도형 수준의 정보를 사용하지 않으므로

이 값이 처음으로 쓰이는 단계는 덮개 세이더 단계이다.

- SV_Instance_ID : 인스턴싱 방식의 그리기 호출에서, 기하구조의 각 인스턴스를 식별하는 용도, 사용 가능한 첫 단계는 정점 세이더 단계이다.

▼ Output Merger ?

Out Merger State 는, 픽셀 세이더에서 출력된 픽셀값을 렌더 타겟에 쓰는 스테이지이다.

▼ Output Merger 단계에서의 주요 2가지 State

- **Blend State** 의 설정에 따라, 최종적으로 쓰여지는 값을 계산하고
- **Depth Stencil State**에 따라, 픽셀값을 쓸것인가를 최종적으로 판정한다.

▼ Output merger 출력 과정 크게 2가지 범주.

(리소스)

- > 텍스쳐 리소스
- > 렌더 타겟 뷰 + 깊이 스텐실 뷰
- ID3D11RenderTargetView + ID3D11DepthStencilView
- > ID3D11DeviceContext::**OMSetRenderTarget()**

(출력 관련 요소)

1. BlendState

- > **D3D11_BLEND_DESC** 구조체
- > ID3D11DeviceContext::**CreateBlendState()** : 블렌드 스테이트 오브젝트 생성
- > **ID3D11BlendState** * 인터페이스 얻기 == 혼합 상태 설정 Control하는 인터페이스
- > ID3D11DeviceContext::**OMSetBlendState()** : ID3D11BlendState 에 세팅된 설정값 파이프라인에 세팅하기

2. DepthStencilState

- > D3D11_Depth_Steril_DESC 구조체
- > ID3D11DeviceContext::CreateDepthStencilState() : 깊이/스텐실 스테이트 오브젝트 생성
- > ID3D11DepthStencilState* 인터페이스 얻기 == 깊이 / 스템실 정보 Control하는 인터페이스
- > ID3D11DeviceContext::**OMSetDepthStencilState()** : ID3D11DepthStencilState 에 세팅된 설정값 파이프라인에 세팅하기

(New)

- ▼ 정점 셰이더 시스템 값들
 - ▼ 정점 셰이더 단계가 출력할 수 있는 새로운 시스템 값 2개
 - SV_ClipDistance, SV_CullDistance
 - ▼ 특성 , 사용 용도
 - 정점 셰이더를 비로산 **레스터화 이전 단계**들은, 모두 이 시스템 값을 변경할 수 있다.

- 이 값들은 레스터화기에서 “절단(Clipping)” 과 “선별 제외(Culling == 선별)”에 쓰인다.

▼ 절단, 선별에 사용되는 점과 평면 사이의 거리 공식 + 의미 3개

나뉘는 두 공간 중 법선이 가리키는 쪽의 공간에 있는 것이고 0이면 평면 바로 위에 있는 것이다. 그리고 음이면 법선 벡터의 반대쪽 공간에 있는 것이다.* 그림 3.19는 첫 번째 경우에 해당한다.

$$D = ax_1 + by_1 + cz_1 - d \quad (3.1)$$

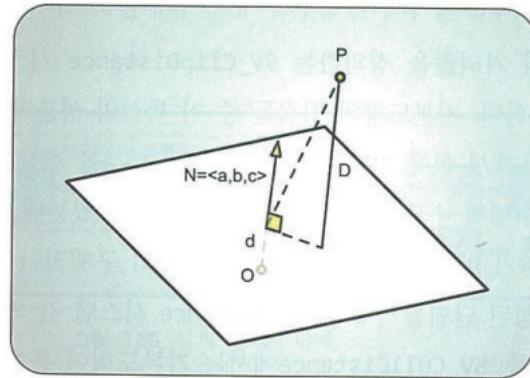


그림 3.19. 한 점과 평면의 거리 계산.

이 거리가 양수이면, 점은 평면으로 나뉘는 2 공간 중에서 법선이 가리키는 쪽의 공간에 있는 것이고

0이면, 평면 바로 위에 있는 것이다.

음이면, 법선 벡터 반대쪽 공간에 있는 것이다.

(평면의 법선 방향에 있는 공간을 평면의 안쪽, 그 반대 방향의 공간을 평면의 바깥쪽. 이라고 부른다.)

▼ SV_CullDistance 작동 원리 5

- 어떤 기본도형을 이후의 처리에서 안심하고 완전히 제외시킬 수 있으려면 기본도형의 모든 정점 위치가 절단 공간을 정의하는 여섯 평면들 중 적어도 하나의 바깥쪽에 있어야 한다.

그러면, 어차피 해당 기본 도형은 보이지 않을 것이므로, 안심하고 폐기할 수 있다.

- 이 시스템값 의미소의 각 성분은, 한 선별 판정 평면과 정점 사이의 거리를 나타낸다.
- 레스터화기 단계는 이 값들을 이용해서 절단 공간 선별 판정을 수행해서, 같은 레지스터 안에 음의 값이 있는, 모든 정점을 제거한다.

```
struct VS_OUT
{
    float4 position : SV_Position;
    float4 clips    : SV_CullDistance;
};
```

- 예를 들어서, **clips** 특성의 어떤 한 성분이 기본도형의 모든 정점에 대해 음수이면, 그 기본도형은 레스터화 연산에서 제외된다.
- 이 예의 경우, **clips** 특성에 최대 4개의 선별 판정 결과를 담을 수 있다.

▼ SV_ClipDistance 3

- 기본도형의 정점들 중 이 특성의 성분이 음인 것이 하나라도 있으면, 그 성분을 이용해서 현재 기본도형에서 잘라낼 부분을 계산한다.
 - 레스터화에 의해 생성된 단편들의 SV_ClipDistance 시스템 값 임소에 음의 값이 포함되는 경우는 없다.
 - SV_CullDistance 시스템 값 의미소에 음의 값이 포함되는 경우는 없다. ↩

즉, SV_CullDistance에서는 기본도형이 통채로 포함되거나 제외되는 반면, SV_ClipDistance에서는 기본도형의 일부만 절단된다.

(SV_ClipDistance 특성이 음이 아닌 부분만 남도록)

▼ 공통 특성 3

- 절단, 선별 의미소 특성들은 한 파이프라인 구성에서 최대 2개까지 사용할 수 있다

- 그리고 특성단 최대 4개의 성분들을 포함할 수 있다.
- 결과적으로, 8개의 평면을 절단이나 선별을 위해 사용할 수 있는 것이다.

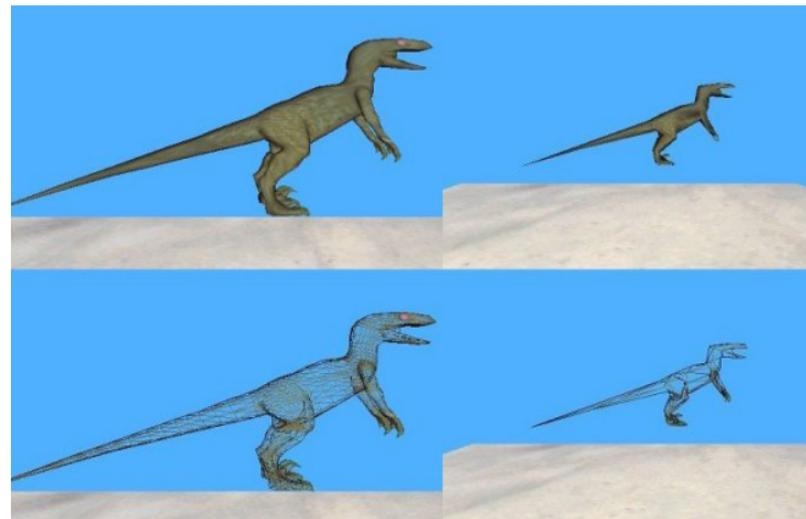
한 파이프라인 구성에서 절단, 선별 의미소 특성들을 최대 두 개까지 사용할 수 있다 (`SV_ClipDistance`를 두 개 사용하거나, `SV_CullDistance`를 두 개 사용하거나, 각각 하나씩 사용하는 등). 그리고 특성당 최대 네 개의 성분들을 포함할 수 있다. 결과적으로 최대 여덟 개의 평면을 절단이나 선별을 위해 사용할 수 있는 셈이다(절단의 경우 `float4` 형식의 `SV_ClipDistance` 두 개, 선별의 경우 `float4` 형식의 `SV_CullDistance` 두 개).

테셀레이션 단계 개요

▼ 테셀레이터 활용 예시

▼ 동적 LOD (Level Of Detail)

LOD란 Level of Detail로 말 그대로 단계에 따라 디테일을 달리 한다는 말이다.
지형이나 오브젝트를 렌더링 할 때 폴리곤의 수를 조정하여 화질의 저하 없이 높은 퀄리티를 내기 위해 사용된다.



<모델>

위 그림은 거리에 따라 디테일을 다르게 준 모델이다.

LOD : 단계에 따라 디테일을 달리한다.

같은 물체를 더 많은 Mesh로 그리면 정교하고, (Low Level)
적은 Mesh로 그리면 덜 정교해질 것이다 (High Level)
카메라의 위치에 가까우면 더 많은 Mesh을 사용하여 그리는 Level 선택
멀리 있으면 높은 레벨 선택

보통은 Mesh 를 여러개 준비해서 바꿔치는 기법인데
그 외에도 동적으로 셰이더 단계에서
삼각형의 개수를 줄였다가 늘렸다가 할 수 있다면 좋을 것 같다.

동적 !

이라는 것은, 렌더링 과정 속에서
변경하겠다는 의미이다.

▼ Control Point, Patch 정의

테셀레이션 단계가 들어가는 순간부터 우리는
이제 PRIMITIVE_TOPOLOGY가 아니라
“_1~32_CONTROL_POINT_PATCHLIST 라는
기본도형 위상구조 변수를 넘겨주어야 한다.

삼각형이 있다면
각 꼭짓점이 도형을 결정
그 꼭짓점들을 Control Patch 라고 부르고 (단, 꼭짓점이 반드시 Control Point가 아
니다)
그러한 Control Point들의 집합을 Patch 라고 부른다.

▼ Control Point 와 Vertext의 차이

Control Point는 제어점의 개념이 강하다.

즉, 얘네들이 그려질 수도 있지만

Control Point 사이에 도형을 그릴 때

그리는 기준점이 될 수도 있다는 의미이다

Hull Shader 는 두 가지의 작업을 동시에 수행합니다.

그것은 제어점(Control Point) 를 생성하는 작업과 Patch Constant Data 를 계산하는 작업입니다.

이들 작업은 병렬적으로 수행되게 됩니다.

HLSL 코드는 실제로 드라이버 수준의 하드웨어 명령어를 생성하게 되는데,

이 때, 병렬처리가 가능한 형태로 변환되게 됩니다.

이는 Hull Shader 가 빠르게 동작할 수 있는 중요한 이유이기도 합니다.

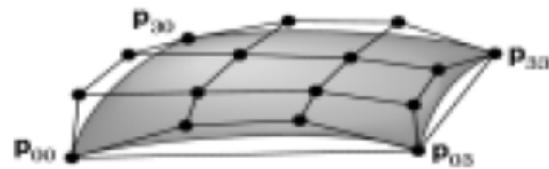
Hull Shader 의 입력으로 들어오는 제어점(Control Point)들은

낮은 차수의 면을 표현하는 정점들입니다.

이를 높은 차수의 면을 표현하는 제어점들로 만들어 내게 됩니다.

이 때 생성된 제어점들은 Tessellator 스테이지에서 사용되는 것이 아니라,

그 다음 스테이지인 Domain Shader 에서 사용됩니다.



위의 그림은 베지어(Bezier) 제어점들을 이용해서 베지어 곡면을 표현한 것입니다.

근본적으로 테셀레이션은 평면을 곡면으로 생성시키는 개념과 매우 비슷합니다.

(굳이 평면을 많은 갯수의 폴리곤으로 표현할 필요는 없기 때문이겠죠.)

그렇기 때문에, 분할 방법으로 사용되는 알고리즘들은 베지어처럼 게임 프로그래머들에게 친숙한 개념들이 사용됩니다.

3.5 덮개 셰이더 단계

▼ 대략적 역할 2개

입력된 기하 구조들을 이후의 처리를 위해 **가공하는 역할**을 수행한다.

즉, 입력 기하구조를 얼마나 잘게 쪼갤 것인지를 테셀레이터 단계에 알려주고 +
영역 셰이더 단계를 위해서는 제어 패치 기본도형들을 처리해서 넘겨준다.

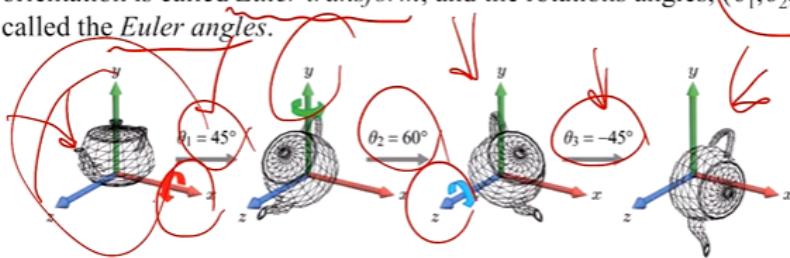
▼ 그래픽 원리 (추가)

▼ 오일러 변환

▼ 정의 , 오일러 각, 특성

Euler Transforms

- When we successively rotate an object about the principal axes, the object acquires an arbitrary orientation. This method of determining an object's orientation is called *Euler transform*, and the rotations angles, $(\theta_1, \theta_2, \theta_3)$, are called the *Euler angles*.



- Concatenating three matrices produces a single matrix defining an arbitrary orientation.

$$R_z(-45^\circ)R_y(60^\circ)R_x(45^\circ) = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \\ 0 & 1 & 0 \\ -\frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix}$$
$$= \begin{pmatrix} \frac{\sqrt{2}}{4} & \frac{2+\sqrt{3}}{4} & \frac{-2+\sqrt{3}}{4} \\ -\frac{\sqrt{2}}{4} & \frac{2-\sqrt{3}}{4} & \frac{-2-\sqrt{3}}{4} \\ -\frac{\sqrt{3}}{2} & \frac{4}{4} & \frac{4}{4} \end{pmatrix}$$

위의 그림과 같아

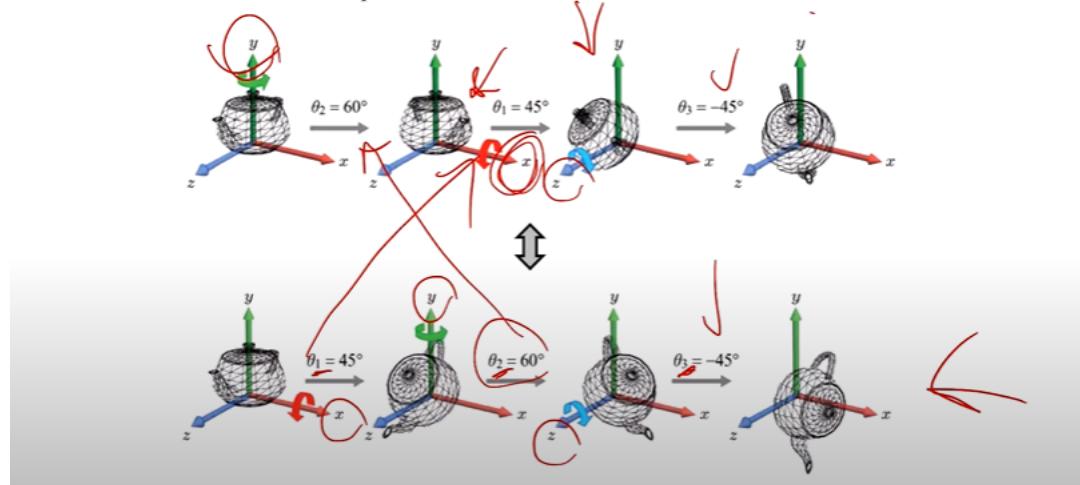
x,y,z 라는 주축 중심의 회전을 결합해서

임의의 방향을 제공하는 것을
오일러 변환. 이라고 한다.

3개 축을 쓰는 것이므로
각도 3개가 주어질 것이고
이러한 각도를
오일러각. 이라고 한다.

Euler Transforms (cont'd)

- The rotation axes are not necessarily taken in the order of x , y , and z . Shown below is the order of y , x , and z . Observe that the teapot has a different orientation from the previous one.



회전 축 순서를 바꾸면
동일한 오일러 각에 대해서도
다른 회전 결과가 나오게 된다.

▼ 세 축의 회전이 종속적인 이유

(즉, z 축이 도는데 x,y 축이 도는 이유)

이 세축이 회전에서 종속적인 이유는 바로 오일러 각에서 회전 자체를 이 세 축으로 나눠서 계산하기 때문이다.

수학적으로 표현된 강체를 돌리기 위해서 우리는 세 축으로 회전 방향을 디지털화 시켰다.

x에 대해서 회전하고 y에 대해서 회전하고 z에 대해서 회전시켰다고 생각해보자.

a는 x에 대해서 회전시키면 이미 x로 회전된 a'의 상태이고 우리는 a에 대해서 y 축으로 회전하는 것이 아니라 a'에 대해서 y축으로 회전하는 것이기 때문에 a''가 된다. 그리고 마지막으로 z에 대해서 회전시킬 때에는 이미 우리가 알던 a가 아니라 a''에 대해서 회전시키는 것이다.

이미 y축으로 돌릴 차례에는 x로 돌아간 상태이기 때문에 두 축에 대한 계산이 독립적일 수가 없다.

회전 변환 행렬을 곱할 때에 x에 대한 회전행렬 Rx가 있고 y에 대한 회전행렬 Ry 가 있고 z에 대한 회전행렬 Rz가 있을 때에

X라는 물체에 대해 회전을 시킨다면

RzRyRxX 와 같이 계산을 할 것이다.

(행렬의 곱셈 참고)

그런데 여기서 각 행렬은 서로 곱해지며 의존적으로 변하기 때문에 아무리 세축에 독립적으로 강체를 돌려보려 해도 그렇게 할 수가 없다.

출처:

<https://handhp1.tistory.com/3>

[망고 먹는 개발자]

▼ 짐벌락 현상

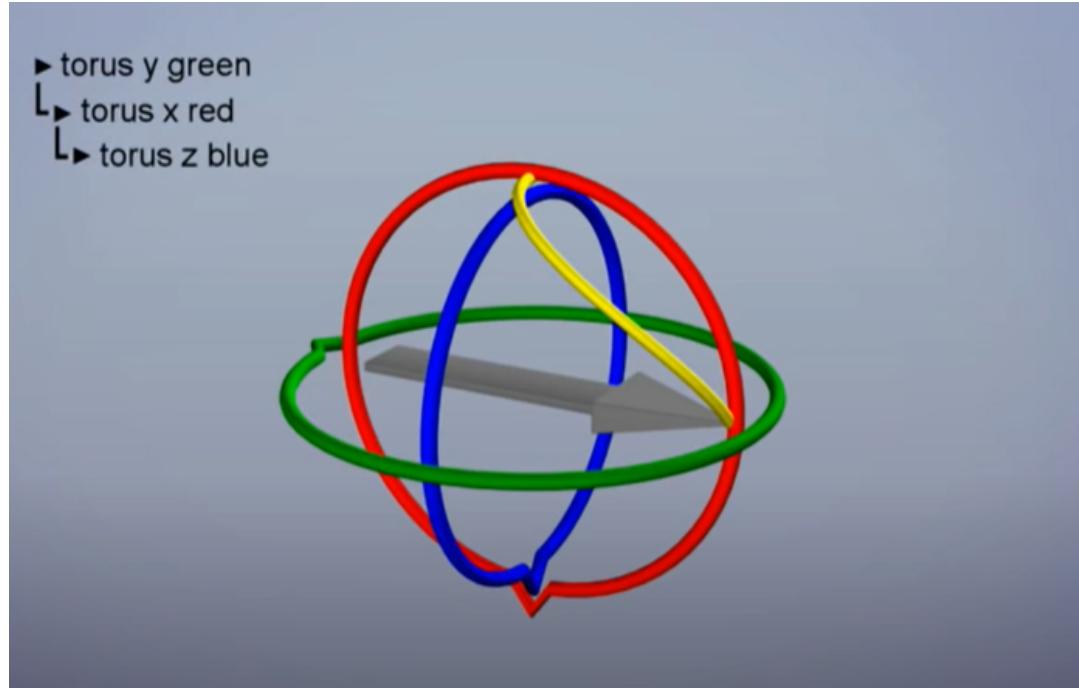
3개 축 중에서 하나의 축이 먹통이 되는 현상

두 번째 고리가 도는 것 때문에 첫 번째 고리와 세 번째 고리가 겹쳐지면, 첫 번째 고리와 세 번째 고리의 회전이 겹쳐져 버리기 때문에 한 축에 대한 자유도를 상실하게 된다. 그래서 두 번째 고리를 움직일 때에 조심스러울 수밖에 없는데 이것이 바로 짐벌락 현상이다.

▼ 짐벌락 현상이 문제가 되는 이유

keyframe 들 사이에서 예상치 못하는 방식으로

물체가 이동할 수 있기 때문이다.



▼ key frame ?, in_between_frame, in_between_frame 그리는 방법

Keyframe Animation in 2D

- In the traditional hand-drawn cartoon animation, the senior key artist would draw the keyframes, and the junior artist would fill the in-between frames.
- For a 30-fps computer animation, for example, much fewer than 30 frames are defined per second. They are the keyframes. In real-time computer animation, the in-between frames are automatically filled at run time.
- The key data are assigned to the keyframes, and they are interpolated to generate the in-between frames.
- In the example, the center position p and orientation angle θ are interpolated.

$$p(t) = (1 - t)p_0 + tp_1$$

$$\theta(t) = (1 - t)\theta_0 + t\theta_1$$

$t = 0$. $t = 1$

keyframe 0

keyframe 1

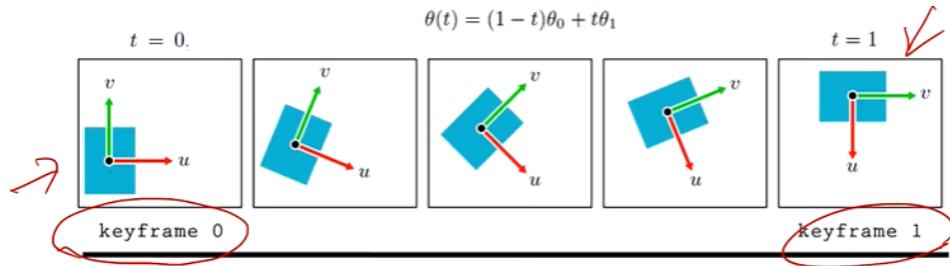
24 frame 의 화면을 그린다고 할때

1, 13, 24 frame 에 해당하는 그림을 key frame

그 사이를

in-between-frame 이라고 한다.

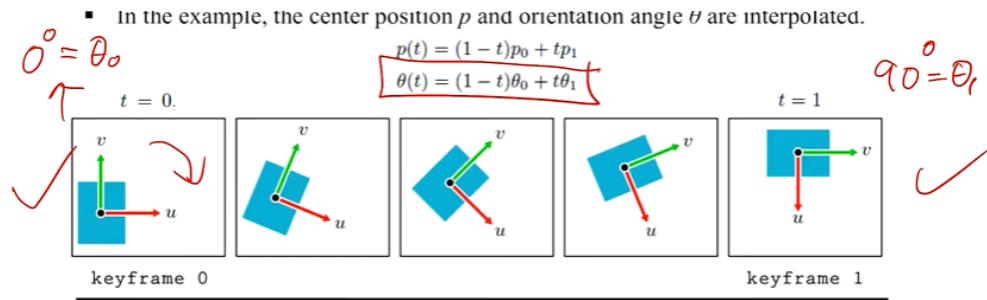
interpolation을 통해 중간 in - between frame을 그려주게 된다.



key frame 0 이 초기값이고

key frame 이 1 지났을 때 변화된 결과를 얻어낼 수 있다.

(즉, 1초가 지났을 때)



중심점은 p_0 에서 p_1 으로 가게 될 것인데

그 중간 과정에서의 중심점을

linear interpolation으로 그려낼 수 있게 된다.

또한 각도도 0에서 90으로 가게 된다.

이러한 각도 또한 선형 보간을 통해서

중간 과정에서의 각도들도 구할 수 있게 될 것이다.

▼ 쿼터니언

▼ 삼각함수 덧셈 공식 (3가지)

“[이코노미+교과수업의 끝나는 끝나는](#)”

$\sin(\alpha + \beta)$	=	$\sin \alpha \cdot \cos \beta + \cos \alpha \cdot \sin \beta$	싸 코 씌 플 코 씌
$\cos(\alpha + \beta)$	=	$\cos \alpha \cdot \cos \beta - \sin \alpha \cdot \sin \beta$	코 코 마 씌 씌
$\tan(\alpha + \beta)$	=	$\frac{\tan \alpha + \tan \beta}{1 - \tan \alpha \cdot \tan \beta}$	일 마 탄 탄 분의 탄 플 탄

“[이코노미+교과수업의 끝나는 끝나는](#)”

▼ 1번째, 2번째 증명까지 마무리

<https://m.blog.naver.com/galaxyenergy/221259791722>

▼ 복소수

▼ (키워드) 표현, 거리 및 좌표로 생각하기

복소수.

① 표현

$$z = \underline{a} + \underline{b}i$$

정수부 허수부.

② 차이 및 곱셈개념

$$|z| = |a+bi| = \sqrt{a^2+b^2}$$

$$(r^2 = 1 \Rightarrow |z|)$$



$$\text{켤레복소수: } z^* = a - bi$$

$$zz^* = a^2 + b^2 = |z|^2$$

<복소평면>

여기

(1)

↑

$$z = r \cos \theta +$$

$$r \sin \theta i$$

$$r \cos \theta \cdot a$$

$$r \sin \theta \cdot b$$

(2)

↑

실수

▼ (키워드) 나눗셈 연산

③ 연산

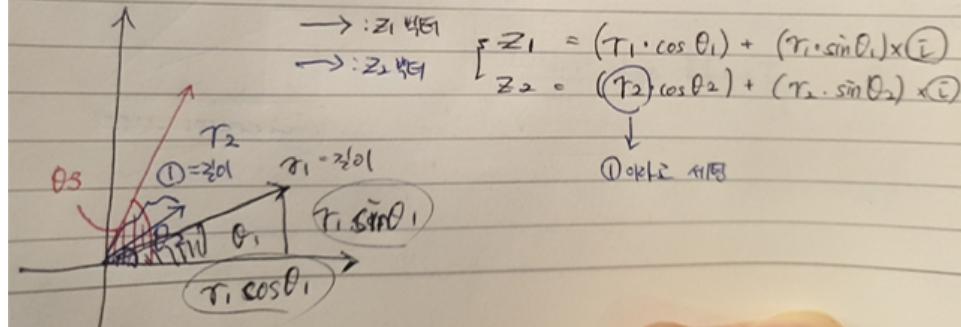
• 나눗셈 방식,

$$\left[\frac{a+bi}{c+di} \right] \times \left(\frac{c-di}{c-di} \right) = \frac{(ac+bd)}{c^2+d^2} + \frac{i(bc-ad)}{c^2+d^2}$$

▼ (키워드) 극좌표 개념으로 2개의 내적하기

④ 극 좌표에서 하위를 나타내는 방식

극 좌표 \rightarrow 기준 x, y 좌표가 아니라, 원단의 크기, 방향, 각도로 표현하는 것



$$Z_1 + Z_2 = r_1 \cdot r_2 \left((\cos \theta_1 \cdot \cos \theta_2 - \sin \theta_1 \cdot \sin \theta_2) + i(\cos \theta_1 \cdot \sin \theta_2 + \sin \theta_1 \cdot \cos \theta_2) \right)$$

* 상수항은 뒷설

$$\begin{aligned} \sin(\alpha + \beta) &\Rightarrow \sin \alpha \cdot \cos \beta + \cos \alpha \cdot \sin \beta \\ \cos(\alpha + \beta) &\Rightarrow \cos \alpha \cdot \cos \beta - \sin \alpha \cdot \sin \beta \\ \tan(\alpha + \beta) &\Rightarrow \frac{\tan \alpha + \tan \beta}{1 - \tan \alpha \cdot \tan \beta} \end{aligned}$$

Windows 정품 인증

①
장점과 단점

$$\sin(\alpha+\beta) \Rightarrow \sin\alpha \cdot \cos\beta + \cos\alpha \cdot \sin\beta$$

$$\cos(\alpha+\beta) \Rightarrow \cos\alpha \cdot \cos\beta - \sin\alpha \cdot \sin\beta$$

$$\tan(\alpha+\beta) \Rightarrow \frac{\tan\alpha + \tan\beta}{1 - \tan\alpha \cdot \tan\beta}$$

$$\Rightarrow r_1 \times \left(\cos(\theta_1 + \theta_2) + i \sin(\theta_1 + \theta_2) \right)$$

\Rightarrow

즉, r_1 벡터를, ① 각도의 합이 ②인 단위벡터와
내적을 계산해보니,
 r_1 벡터는 ② 대진(对进)사인 결과가 나온 것

즉, 쿼터니언 간의 곱

혹은 쿼터니언 벡터 간의 내적으로

회전을 표현할 수 있다!

▼ (키워드) 위의 과정 행렬로 표현하기

회전은 표현

$$z = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \Rightarrow \begin{bmatrix} a & -b \\ -b & a \end{bmatrix}$$

(좌우방위 회전위)

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$I = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

즉, 대칭임,

직접
부록을 (현전)의 관점에서
생각하는 편이
훨씬,

$$T = T_1 \cdot \cos\theta + (T_1 \cdot \sin\theta) i$$

$$\Rightarrow \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

▼ 쿼터니언 기본 개념 및 제2코사인 법칙

2차원 회전 $\Rightarrow a+bi$ 를 이용하여 표현,,

3차원 $\Rightarrow a+b\hat{i}+c\hat{j}+d\hat{k} \Rightarrow$ 총 4개 원소 사용,,

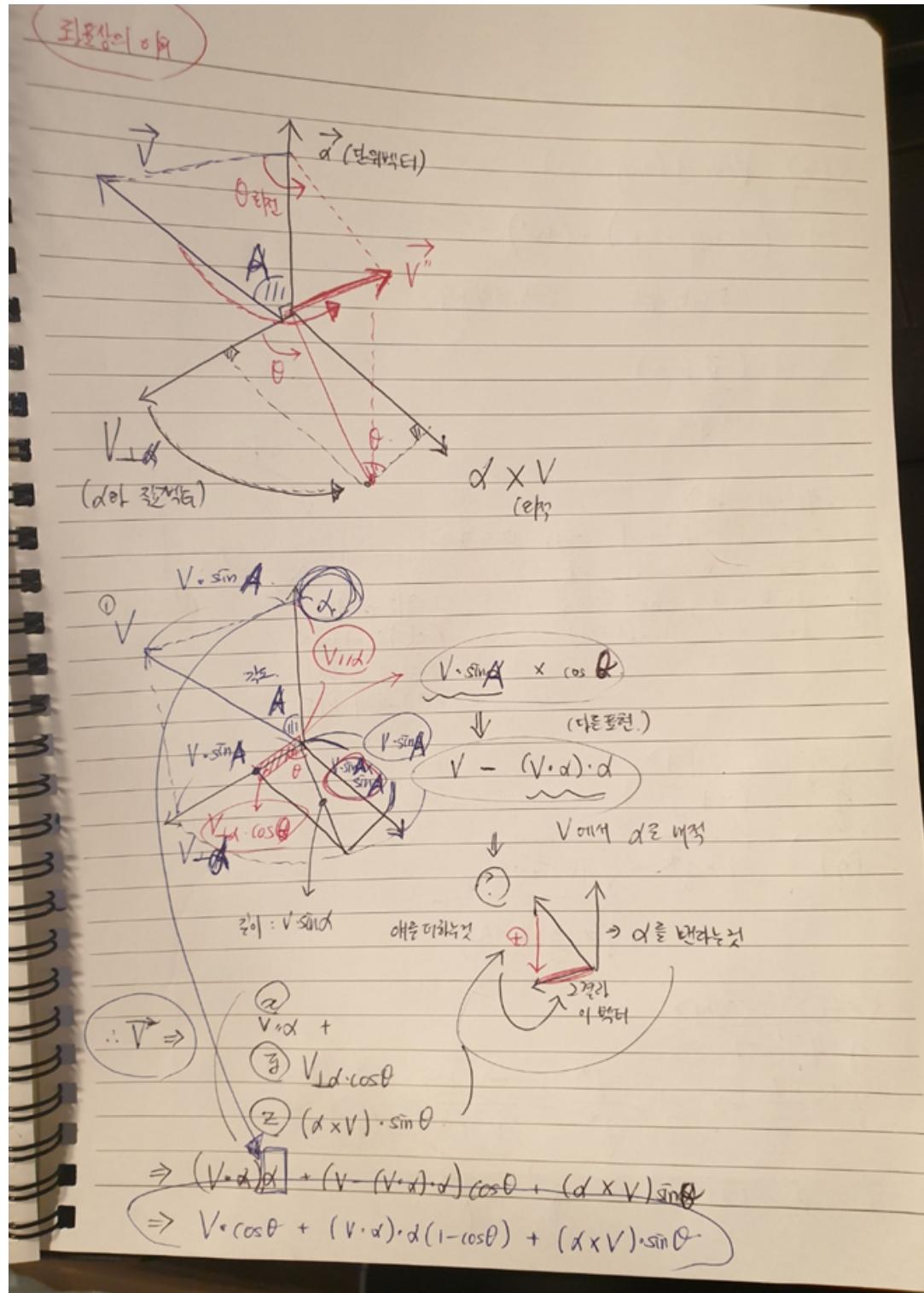
* (별도: 물체의 멀티가지 속수 알고 싶어하기)

- 코사인 제2법칙

$$a^2 = b^2 + c^2 - 2bc \cos A.$$

$$\begin{aligned} a^2 &= (c - b \cos d)^2 + b^2 \sin^2 d \\ &= c^2 + b^2 \cos^2 d + b^2 \sin^2 d - 2bc \cos d \\ &= c^2 + b^2 (\sin^2 d + \cos^2 d) - 2bc \cos d \\ &= c^2 + b^2 - 2bc \cos d \quad \text{①} \end{aligned}$$

▼ (키워드) 3차원 좌표상 회전의 관점으로 이해하기



▼ 쿼터니언 그외 성질

▼ 쿼터니언의 구성

그리 성질

<구성> Vector $\vec{v} (x, y, z, w)$

$$\vec{v} = (x\hat{i} + y\hat{j} + z\hat{k}) + (w)$$

벡터부, 학부

실수부, 스칼라부

$$= (\underline{\underline{\vec{v}}}, s)$$

x, y, z .

▼ 각 원소간의 상호 관계

$$= (\underline{\underline{\vec{v}}}, s)$$

x, y, z .

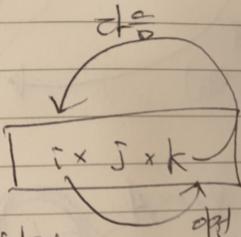
<서로간의 상호>

• $\hat{i}^2 = \hat{j}^2 = \hat{k}^2 = -1$, 이라는 성질도 양쪽.

• $\hat{i} \times \hat{j} = \hat{k}$, $\hat{j} \times \hat{i} = -\hat{k}$,

• $\hat{j} \times \hat{k} = \hat{i}$, $\hat{k} \times \hat{j} = -\hat{i}$

• $\hat{k} \times \hat{i} = \hat{j}$, $\hat{i} \times \hat{k} = -\hat{j}$



뒤로 돌아가면,

(양) 뒤에 네석

앞으로 돌아가면

(음) 앞의 네석,

▼ 컬레와 거리

$$\begin{aligned}
 & \langle \text{켤레} \rangle \cup \langle \text{거리} \rangle \\
 f^* &= (-V, S) \\
 |f| &= \sqrt{f \cdot f^*} = \sqrt{|V|^2 + S^2} \\
 &= \sqrt{x^2 + y^2 + z^2 + w^2}
 \end{aligned}$$

▼ 켤레와 역수

$$\begin{aligned}
 & \langle \text{역수} \rangle, \langle \text{켤레} \rangle \\
 f \cdot f^{-1} &= 1 \\
 f^{-1} = \frac{1}{f} &\Rightarrow \frac{1}{f} \times \frac{f^*}{f^*} \Rightarrow \frac{-V + S}{|V|^2 + S^2} \quad \text{if } |f| = 1 \quad (\text{여기!}) \\
 &= \textcircled{1} \text{이 된다} \\
 \therefore \text{if } (|f| = 1) &\Rightarrow f^{-1} = f^*
 \end{aligned}$$

▼ 켤레와 곱

$$\begin{aligned}
 & \langle \text{켤레} \rangle \langle \text{곱} \rangle \\
 (f_1 \circ f_2)^* &= f_2^* \cdot f_1^*
 \end{aligned}$$

▼ 거리와 곱(절대값)

〈거리, 각〉

$$|q_1 \cdot q_2| = |q_1| |q_2|$$

▼ 벡터간 외적

(추가정식)

$$\begin{aligned} \vec{a} &= (a_x, a_y, a_z) \\ (\vec{b} &= (b_x, b_y, b_z)) \quad \rightarrow \text{벡터간 } \boxed{\text{외적}} \quad \vec{a} \times \vec{b} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix} \end{aligned}$$

▼ 벡터 삼중곱

(벡터 삼중곱) (여기)

$$\vec{a} \times \vec{b} \times \vec{c} = \vec{b} (\vec{a} \cdot \vec{c}) - \vec{c} (\vec{a} \cdot \vec{b})$$

| |
 스칼라 스칼라.

▼ 2개 쿼터니언 내적 공식

< 쿼터니언 곱 다른 표현 >

$q_1 q_2 ?$

$$q_1 = x_i + y_j + z_k + w_i \Rightarrow (\vec{v}_1, s_1)$$

$$q_2 = x_2 i + y_{2j} + z_{2k} + w_2 \Rightarrow (\vec{v}_2, s_2)$$



$$q_1 \cdot q_2 = \left(\vec{v}_1 \times \vec{v}_2 + s_1 \vec{v}_2 + s_2 \vec{v}_1, s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2 \right)$$

Vector

Scalar.

(부록)

▼ 쿼터니언 회전 표현 공식

< 쿼터니언 회전 표현 방법 >

$$q = x_i + y_j + z_k + w$$

$$V'' = q \cdot V \cdot \underline{\underline{q^{-1}}}$$

$\hookrightarrow |q| = 1$ 이라면 $\underline{\underline{q^*}}$ 로도 표현할 수 있다.
(켤레복소수)

$$= q \cdot V \cdot q^*$$

▼ Pure Quaternion

< Pure Quaternion ? >

$$V_f \rightarrow (\vec{J}, \frac{o}{\downarrow})$$

$$\text{公} = \textcircled{0}$$

▼ 쿼터니언 회전 표현 공식 증명

Pure Quaternion,,

$$q \cdot v = q^* \quad (\text{if } |q| = 1)$$

$\Rightarrow (\vec{u}, w) (\vec{v}, 0) (-\vec{w}, w)$

$* q_1 q_2 \Rightarrow (V_1 \times V_2) + S_1 V_2 + S_2 V_1, \quad S_1 S_2 - V_1 \cdot V_2$

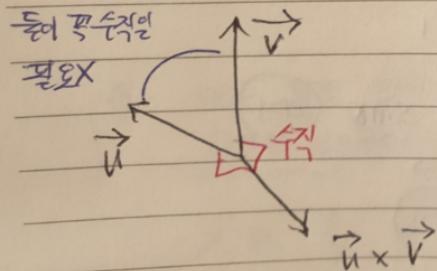
$\begin{cases} q_1: (\vec{v}_1, S_1) \\ q_2: (\vec{v}_2, S_2) \end{cases}$

$$\Rightarrow (\vec{u}, w) (-\vec{v} \times \vec{u} + w \cdot \vec{v}, \vec{v} \cdot \vec{u})$$

(설명) 원자,

$$w \cdot \vec{v} \vec{u} - \vec{u} \cdot (-\vec{v} \otimes \vec{u} + w \cdot \vec{v})$$

$$\Rightarrow \vec{u} \cdot (\vec{v} \otimes \vec{u}) \Rightarrow \text{수직인 벡터끼리 내적} \Rightarrow \textcircled{0}$$



즉, 설명 - \textcircled{0}

즉, 그 곱도, 순수 쿼터니언이 나온다는 것이다.

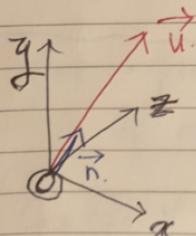
(설명)

$$\Rightarrow \underbrace{w^2 \cdot \vec{v}}_{\text{기하학적으로 생각}} + 2w \cdot (\vec{u} \times \vec{v}) + 2 \cdot \vec{u} (\vec{u} \cdot \vec{v}) - \vec{v} (\vec{u} \cdot \vec{u})$$

기하학적으로 생각

$$q = (\underline{\underline{u}}, w)$$

해설 $\rightarrow x, y, z$ 은 폐부



이때, \vec{v} 는 이와 같이 3차원상에서 표현 가능.

그리고 해당 방향으로 단위벡터를 \vec{n} 이라고 하면,

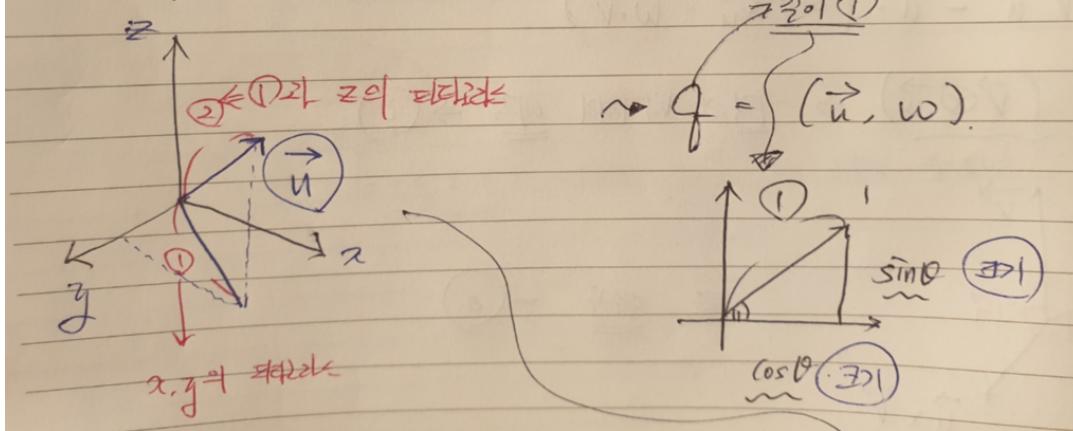
$$\vec{u} = \underset{(크기)}{S} \times \vec{n}$$

으로 표현할 수 있다.

(회동 속도)

각. 우리는 현재 $|\vec{q}| = 1$ 인 벡터인을 다루고 있다.

길이는 구할때, 어떤식으로 진행했나?



x, y 의 대비관계

즉, $|\vec{q}|$ 는 \vec{u} 와 \vec{w} 을 피타고레스에서
연습해보니!

그리면 \vec{u} 에서 한 바퀴도 빙글면서 그 \vec{q} 가. $\sqrt{u^2 + w^2}$

$w = \cos \theta$ 라 할수있다 \Rightarrow 해당 벡터의 크기이므로
 $u = \sqrt{\sin^2 \theta}$ 라고도 표기할 수 (이전 방향을 생각)
있지 않을까??

$$\therefore \sqrt{u^2 + w^2} = \left((w^2 - \vec{u} \cdot \vec{u}) \cdot \vec{v} + 2(\vec{u} \cdot \vec{v}) \cdot \vec{u} + 2 \cdot w \cdot (\vec{u} \times \vec{v}) \right)^{(0)}$$

↓

$$\left(\cos^2 \theta + \sin^2 \theta = 1 \right)$$

$$\left(u^2 = \sin^2 \theta = s^2 \quad \text{길이 } 1 \text{인 단위벡터} \right)$$

$w = \cos\theta$ $\cancel{\text{가능}}$ \checkmark $\because (\vec{n} \in \sqrt{\sin\theta})$ 라고도 표기할 수 (이유는?)
 w 는 $\frac{\sin\theta}{\cos\theta}$ 일지 않을까?
 $\therefore \vec{V}'' = ((w^2 - \vec{U} \cdot \vec{U}) \cdot \vec{V} + 2(\vec{U} \cdot \vec{V}) \cdot \vec{U} + 2w \cdot (\vec{U} \times \vec{V})) \circlearrowleft$
 \downarrow
 $(\cos^2\theta + \sin^2\theta = 1)$
 $(\vec{U}^2 = \sin^2\theta = 1)$ 길이 1인 단위벡터
 $\Rightarrow ((\cos 2\theta) \cdot \vec{V} + (1 - \cos(2\theta))(\vec{U} \cdot \vec{V}) \cdot \vec{U} + \sin(2\theta)(\vec{U} \otimes \vec{V})$
 $(\cos^2\theta - \sin^2\theta = \cos(2\theta))$
 $(2\cos\theta\sin\theta = \sin(2\theta))$
 $\cos(2\theta) = 1 - 2\sin^2\theta.$

우리가 알아서 보면,
 $\vec{V}'' = \vec{V} \cdot \cos\theta + (\vec{V} \cdot \vec{U}) \cdot \vec{U} \cdot (1 - \cos\theta) + (\vec{U} \otimes \vec{V}) \cos\theta$
 (회전된 벡터)
 이 같은 형태.

 주. 내가 특정 벡터를, 컴퓨터언을 이용해서
만큼 회전시키자 한다면
 그 반 % 를 각도로 해서.

즉. 내가 특정 벡터를, 컴퓨터언을 이용해서
θ만큼 회전시키자 한다면,

그 뒤 $\frac{\theta}{2}$ 를 각도로 해서.

$$f = \left(\sin\left(\frac{\theta}{2}\right) \cdot \vec{v}, \cos\left(\frac{\theta}{2}\right) \right)$$

↓
 짚어 1인
 단위벡터,

↓
 치수.

$$V'' = q \cdot V \cdot f^* 형태로 변환해주면 된다.$$

$$V'' = q \cdot V \cdot f^* 형태로 변환해주면 된다.$$

만약. 여기번의 변환을 거치고 싶다면,

즉. 여러분의 컴퓨터언을 곱하고 싶다면.

$$(q_2(q_1 \cdot V \cdot f_1^*) \cdot f_2^*) \rightsquigarrow (q_2 q_1)^* = q_2^* q_1^*$$

↓
 을 이동

$$(q_2 q_1) \cdot V \cdot (q_2 q_1)^*$$

로 표현할 수 있을 것이다.

즉. 회전의 컴퓨터언을 만들었을 때
 $(q_2 q_1)$ (여러번 변환시에)
 적용해주면 된다.

▼ 직교 투영

▼ 원리

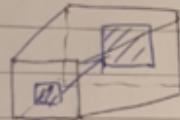
직교투영?

"원근도형"과 함께 "Projection Transformation"의 한 종류이다.

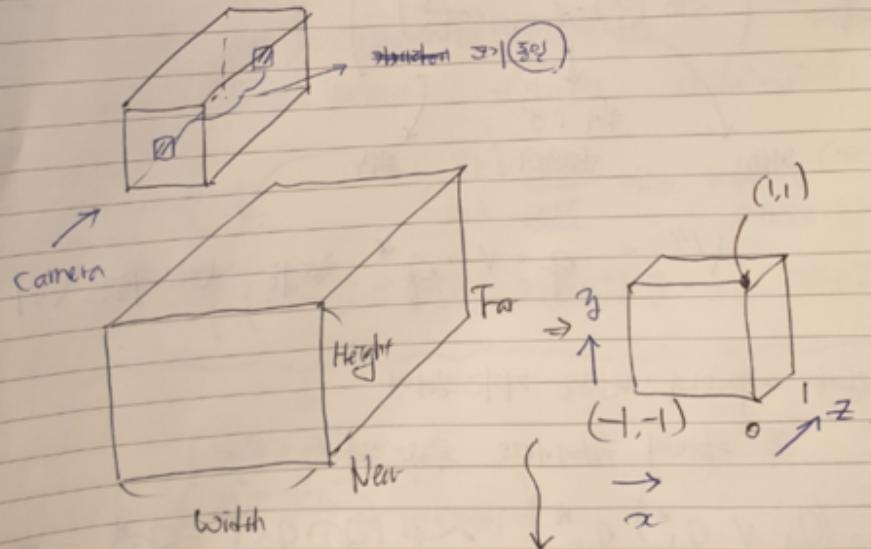
원근도형에서는 근간(깊이값)에 따라

실제 대상의 크기가 넓면에 비례하는

크기와 약수 다른다.

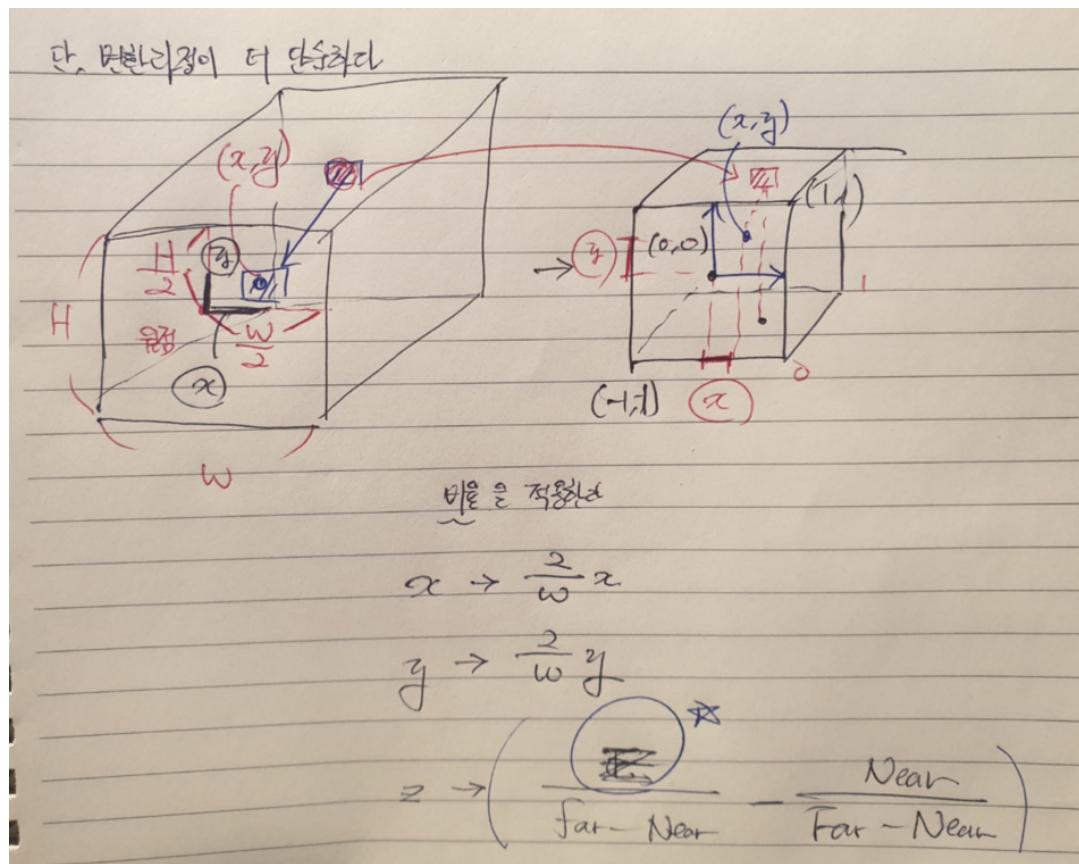


(직교) 투영에서는 근간(깊이값)이 드시가 된다.
질문해가 표현방법, 형태가 되기 때문이다.



이제는 형태론의 변환을 해야한다는 점은
증명하라

$$\begin{cases} x : -1 \sim 1 \\ y : -1 \sim 1 \\ z : 0 \sim 1 \end{cases}$$

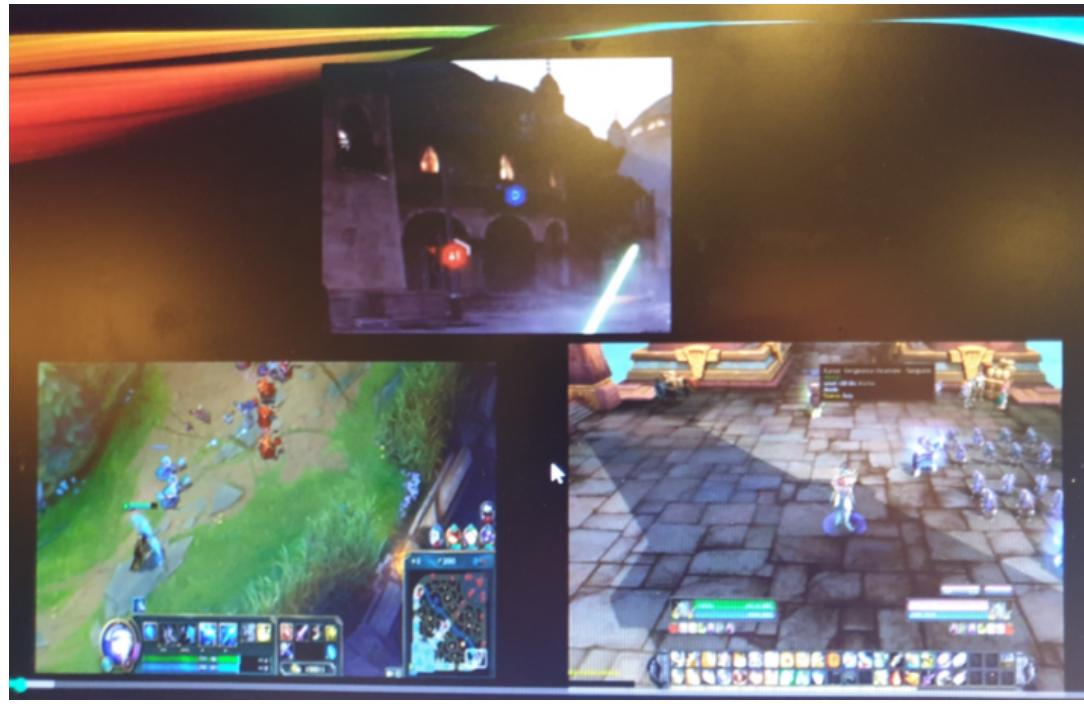


$$\begin{aligned} & \text{if } z = \text{Near} \rightarrow ① \\ & \quad = \text{Far} \rightarrow ② \end{aligned}$$

! 멀티리지 \Rightarrow

$$\begin{bmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 0 \\ 0 & 0 & \frac{-n}{f-n} & 1 \end{bmatrix}$$

▼ 적용 상황



이런식으로 화면 상에는 물체에 붙은 UI는 원근감을 줘서
원근 투영을 하지만

화면상에 크기가 고정되어야 하는 UI 들의 경우
직교 투영을 적용해주어야 한다.

▼ 시스템 프로그래밍

알림가능한 상태, APC

▼ 문제

```

int _tmain(int argc, TCHAR* argv[])
{
    TCHAR fileName[] = _T("data.txt");

    HANDLE hFile = CreateFile (
        fileName, GENERIC_WRITE, FILE_SHARE_WRITE, 0, CREATE_ALWAYS, FILE_FLAG_OVERLAPPED, 0
    );
    if(hFile == INVALID_HANDLE_VALUE)
    {
        _tprintf(_T("File creation fault!\n"));
        return -1;
    }

    OVERLAPPED overlappedInst;
    memset(&overlappedInst, 0, sizeof(overlappedInst));
    overlappedInst.hEvent= (HANDLE)1234;           // 추가로 데이터 전송 가능한 경로.
    WriteFileEx(hFile, strData, sizeof(strData), &overlappedInst, FileIOCompletionRoutine);

    SleepEx(INFINITE, TRUE);
    CloseHandle(hFile);

    return 1;
}

```

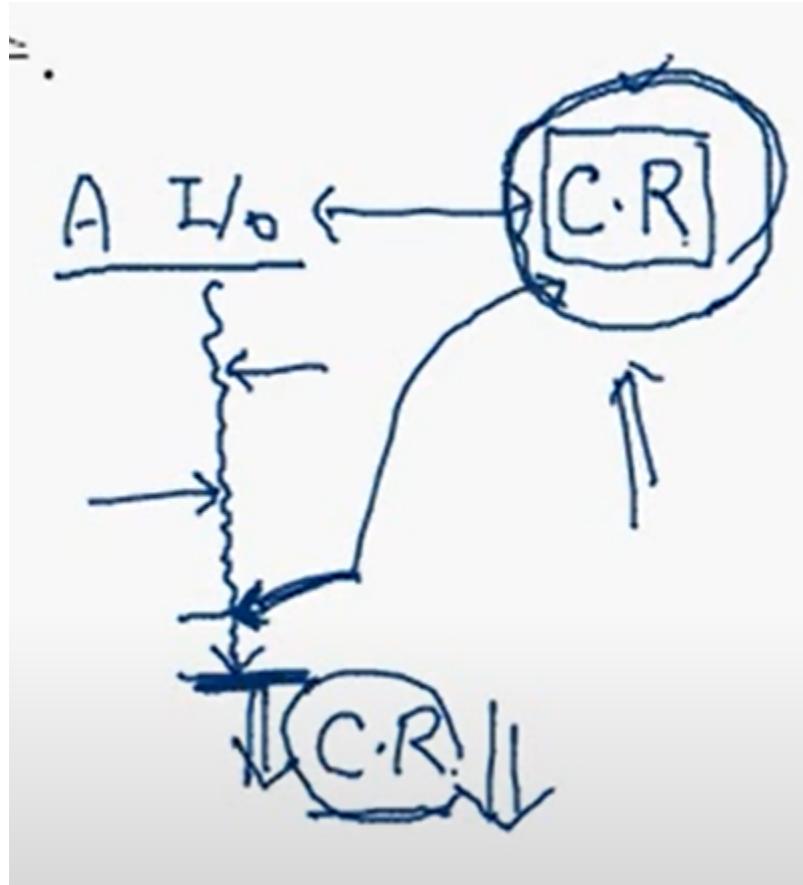
▼ 다음 함수의 의미

일반적인 Sleep 함수.

단, 2번째 인자에 True를 넣어주면

해당 함수를 호출한 쓰레드 (프로세스가 아니라)는 Alertable State 가 된다라는 의미이다.

▼ 알람 가능한 상태 ?



IO 연산을 실행했다고 해보자

나는 특정 지점까지 IO연산이 진행되기를 기대했는데

중간에 끝날 수 있다.

그러면 그 녀석은 끝날 때 바로 Complete routine 함수를

실행하려 가버릴 텐데

이 말은 즉슨 내가 다른 일이 아니라

이 일을 해야 한다는 것

즉, 일의 우선순위를 넘겨준다는 것

적어도 나는 특정 지점까지 IO연산을

진행하고나서야 비로소

Completion Routine을 실행할 꺼야 ! 라고 할 수도 있다.

전자의 경우는

Completion Routine의 우선순위를 높여주는 것이고

후자는 우선순위를 낮춰주는 것이다

알림 가능한 상태.란

나 이제 일 다했으니까

Completion Routine 을 실행해도 괜찮아!

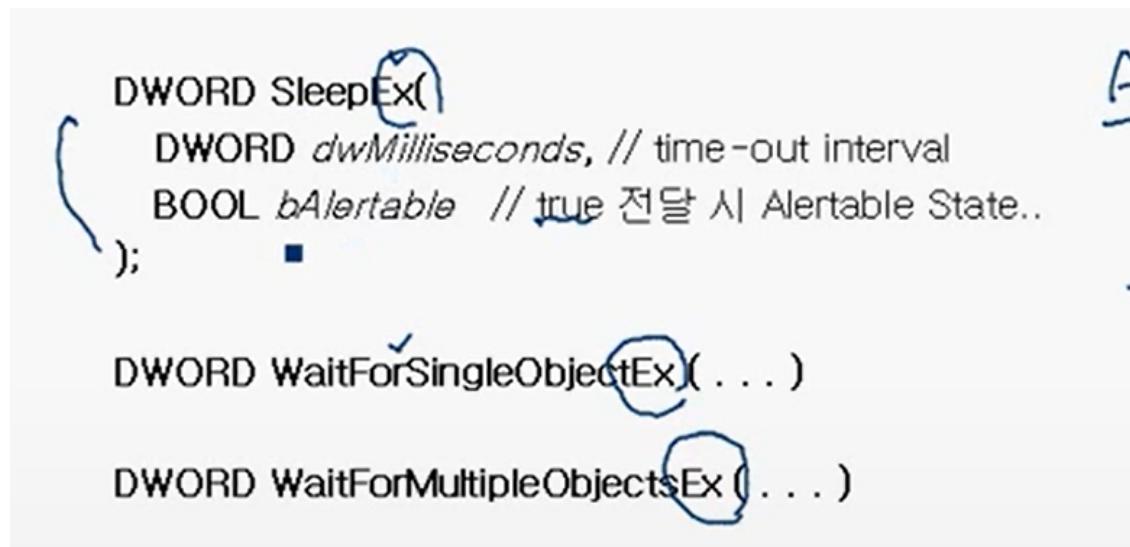
라고 하는 것

즉, C.R의 우선순위를 높여주겠다! 라는 것이

알림 가능한 상태. 라는 것이다.

그러면

▼ 실행함수 3가지



2번째 인자를 true로 세팅해줘야 한다.

▼ (보기)

```

FileIOCompletionRoutine ( DWORD, DWORD, LPOVERLAPPED);

int _tmain(int argc, TCHAR* argv[])
{
    TCHAR fileName[] = _T("data.txt");

    HANDLE hFile = CreateFile (
        fileName, GENERIC_WRITE, FILE_SHARE_WRITE, 0, CREATE_ALWAYS, FILE_FLAG_OVERLAPPED, 0
    );
    if(hFile == INVALID_HANDLE_VALUE)
    {
        _tprintf(_T("File creation fault! \n"));
        return -1;
    }

    OVERLAPPED overlappedInst;
    memset(&overlappedInst, 0, sizeof(overlappedInst));
    overlappedInst.hEvent= (HANDLE)1234;           // 추가로 데이터 전송 가능한 경로.
    WriteFileEx(hFile, strData, sizeof(strData), &overlappedInst, FileIOCompletionRoutine);

    SleepEx(INFINITE, TRUE);
    CloseHandle(hFile);

    return 1;
}

```

▼ 위 줄에 해당함수를 실행하지 않는다면 ? 저 줄의 의미

저 줄을 넣어주지 않는다면

해당 IO연산이 끝나더라도 Completion Routine이

실행되지 않을 수도 있다는 것을 의미한다.

따라서 내가 원하는 시점에

C.R를 호출하고 싶다면

저 함수를 세팅해줘야 한다.

Aleratable State로 바꿔주는 함수를

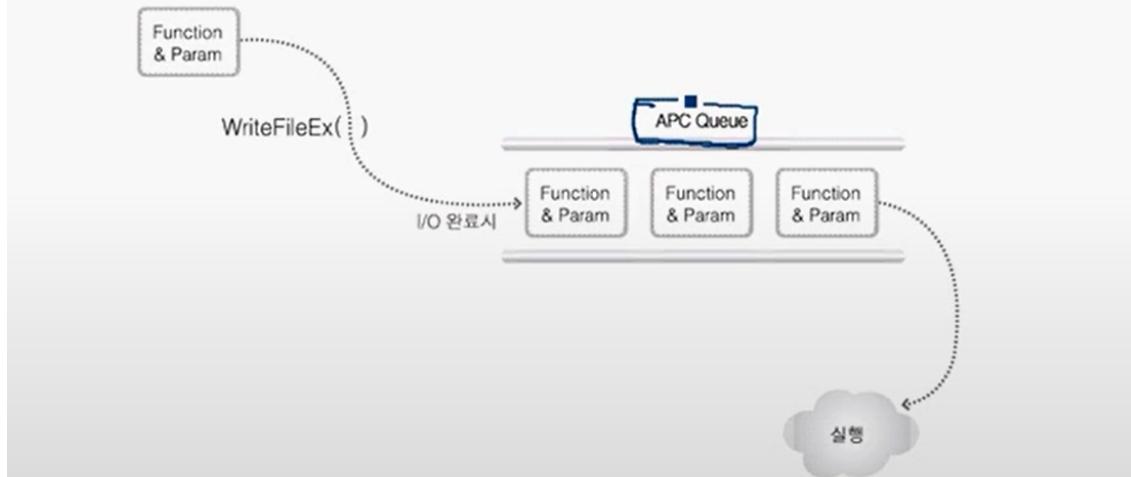
넣어줘야 한다.

▼ APC (비동기 함수 호출 매커니즘) Queue ? // 관련 함수

APC(비동기 함수 호출 메커니즘)



- 모든 쓰레드는 자신만의 APC Queue를 소유



각 쓰레드는 독립적으로 APC Queue
를 지니고 있다.

여기에 들어간 것들은
해당 쓰레드가 알림 가능한 상태로 들어갔을 때
호출할 콜백 함수들을 모아놓은 것

즉, 해당 쓰레드가 알림 가능한 상태가 되면
이 큐 안에 있는 **모든 함수들이 실행**되는 것이다.
그 다음 완전히 큐를 비워준다.

WriteFileEX 의 함수 ?

IO가 완료가 될 경우에
ACP Queue에다가 콜백함수 정보를 입력한다.

20장. 메모리 관리

▼ 가상 메모리 관리

▼ 페이지 개수 구하는 방법

총 공간 // 페이지 크기

ex) 4GB 를 메모리 할당 받고, 페이지 하나의 크기가 400mb 라면

페이지 개수는 10개가 될 것이다.

(페이지 : IN 가상 메모리)

▼ 페이지 상태 3가지

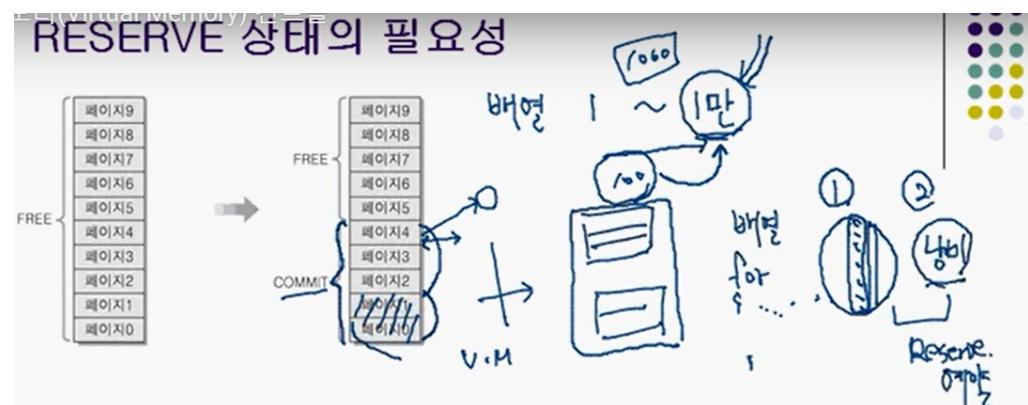
Reserve는 Window Only

Free, Commit

▼ 문제

Commit 상태와 Free 상태의 차이 ?

▼ 답변



처음 : free → 내가 할당 X → 물리 메모리 공간과 연결되지 않았다.

비어있는 메모리 공간

commit은 물리 메모리 공간이 연결된 공간

메모리 공간 할당이 이루어진 것

ex) malloc

하나의 페이지를 commit 한다라는 의미는

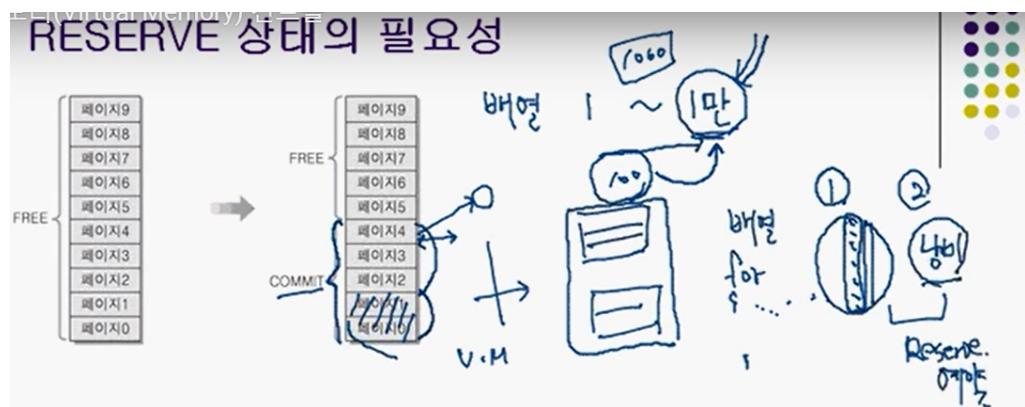
하나의 물리 메모리를 매핑 시키겠다는 의미이다.

해당 페이지를 free 공간으로 다시 만들어준다는 것은

해당 공간이 물리 메모리와의 연결이 끊어진다는 것이다.

▼ 문제 2

▼ Reserve 상태의 필요성



한 달에 한번만 10000 만큼의 메모리 공간을 쓰지만

평소에는 100 정도의 메모리 공간을 쓰는

프로그램이 존재한다고 했을 때

한달에 한번이라는 순간을 위해

미리 10000을 잡아놓는 것은 비효율적이다

이 말은 즉슨 10000 만큼의 페이지를
commit 상태로 둔다는 것
물리 메모리와 매핑 시켜둔다는 것
다른 메모리를 매핑 시키지 못한다는 것

그러면 우선 100을 잡아두고
나중에 10000 을 쓸 수 있게
reserve 상태의 페이지를 두자

또 여기서 발생가능한 문제는
메모리들이 순차적으로 연결된 형태가 되지 않을 수도 있다는 것이다.
연결된 메모리 공간을 할당하고 싶은 것이다.

아래 2개는 commit

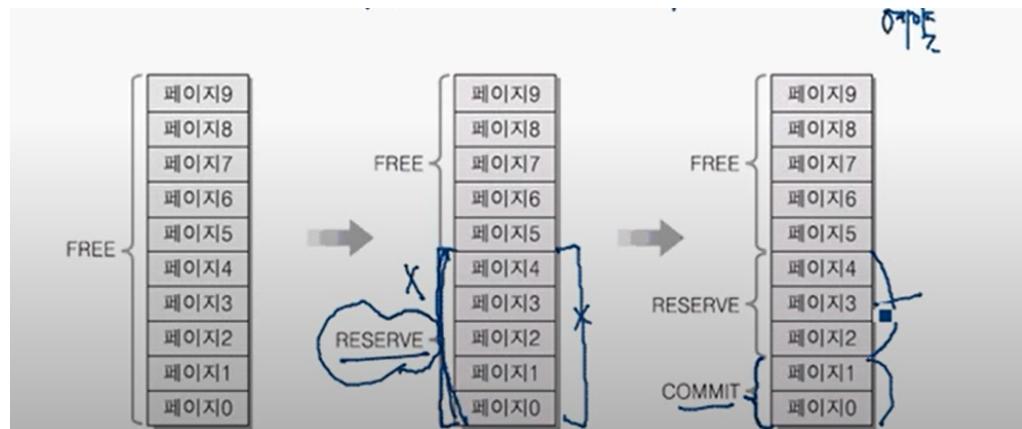
그 위 3개는 reserve

배열의 for 문 등이 가능하게 하기 위해
메모리들이 한번에 연결되어 잡히도록 해야 하는ㄷ
reserve 영역 따로, commit 영역 따로
이렇게 끊겨서 메모리가 잡힐 수도 있다는 것이다.
필요할 때마다 조금씩 메모리를 가져다 쓰는 것은
좋은 해결책이 되지 않는다는 것이다.

가상 메모리 페이지 상에서

- 1) 연결된 메모리 공간도 제공해주면서
 - 2) 메모리 낭비도 막는 방법은 없을까
- 이를 해결하는 것이 reserve 상태

▼ Reserve 상태의 의미



어떤 식으로 이루어져 있는가

미리 쓸 거니까 이 공간 건드리지 마 ! 라고 잡아두는 것

즉, 필요하기 전까지는 물리 메모리와의 매팅도 허용하지 않다가

나중에 필요할 때 commit 상태로 두면서

매팅 시키겠다는 의미이다.

▼ 메모리 할당의 시작점과 단위 (특정 단어) + 그 이유

특정 메모리 시작점을 기준으로

1페이지 단위로 할당 (최소 단위)

단, **Allocation Granularity Boundary** 기준으로

메모리가 할당된다.

(이유 = 메모리 단편화를 막기 위해서)

이게 무슨 말이냐면,

4개의 페이지가 있다고 할때

A,B,A,B 이렇게 따로 할당되는 것을 막기 위해서

이러면 비효율적이니까

미리 2개를 최소로 할당하는 것

즉, 내가 특정 시작점을 기준으로 A를 한개만

할당한다고 하더라도

실제는 2개 크기 == **Allocation Granularity Boundary**

가 할당된다는 것이다

예를 들어, 내가 4k byte를 할당하고 싶다고 한다면

실제로는 64k byte가 할당되고

또 다시 다른 종류의 메모리를 할당한다면

나머지 60kbyte는 그대로 두고

다른 영역에 또 다시 64k byte를 할당하고

그 안에 할당할 메모리를 채워넣는 개념

미리 넉넉하게 메모리 공간을 확보해두는 개념이다.

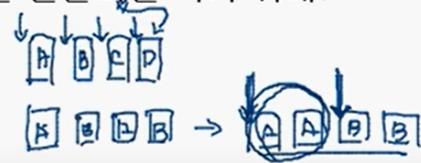
▼ 관련구조체 및 함수

- ▶ GetSystemInfo(&si) // SYSTEM_INFO
 - pageSize = si.dwPageSize
 - allocGranularity = si.dwAllocationGranularity

메모리 할당의 시작점과 단위확인



- “메모리 할당의 시작점”
 - Allocation Granularity Boundary, 기준
 - 페이지 크기의 몇 배수: 지나친 단편화를 막기 위해.



- 할당할 메모리의 크기
 - 최소 1페이지 이상
- GetSystemInfo(&si) // SYSTEM_INFO
 - pageSize = si.dwPageSize
 - allocGranularity = si.dwAllocationGranularity

▼ 가상 메모리 할당 및 해제 함수 // malloc, free와 다른 점

VirtualAlloc & VirtualFree

```
LPVOID VirtualAlloc (
    LPVOID lpAddress,           // 할당의 시작 주소.
    SIZE_T dwSize,              // 할당의 크기
    DWORD  fAllocationType,     // MEM_RESERVE or MEM_COMMIT
    DWORD  fProtect            // PAGE_NOACCESS or PAGE_READWRITE
);
```

반환: 할당이 이뤄진 메모리의 시작 번지!

```
BOOL VirtualFree (
    LPVOID lpAddress,
    SIZE_T dwSize,
    DWORD dwFreeType // MEM_DECOMMIT or MEM_RELEASE
);
```

위 함수들은 reserve 상태까지 설정할 수 있다.

malloc, free는

commit, free 상태만을 지정해줄 수 있다.

▼ 각 함수의 3번째 인자 의미

만약 virtual alloc에 mem_reserve를 둔다면, 해당 페이지를
reserve 상태로 두겠다는 의미이다.

그리고 virtual alloc에서 mem_release를 해주면
해당 페이지를 free 상태로 내리겠다는 의미이다

반면

mem_commit을 세팅하면
해당 페이지를 commit 상태로 두겠다는 의미이고

virtual free 함수에
mem_decommit, mem_release 을 세팅하면
각각 해당 commit 상태를
reserve, free 상태로 둔다는 의미이다.

▼ 해당 함수들의 메모리 할당 공간 ?

malloc, free 는 Heap

사실 Virtual Alloc, Free 함수도 Heap 이기는 하지만
가상 메모리 공간의 일부에 할당한다.
라고 생각하는 것이 더 좋다

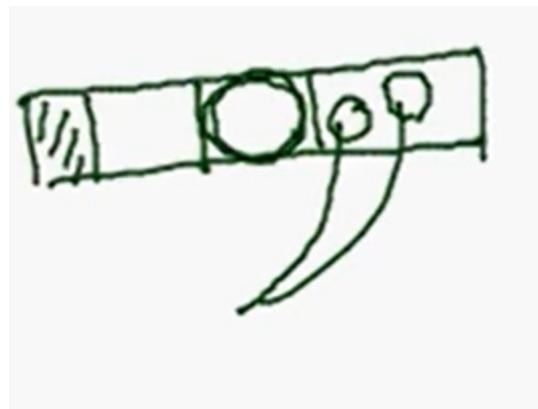
▼ 힙 메모리 관리

▼ 디폴트 힙 ?

프로세스에 대해 기본적으로 할당해주는 힙공간

초기 메모리 공간의 크기가 정해져 있다는 의미

물론 알아서 크기가 늘어난다.

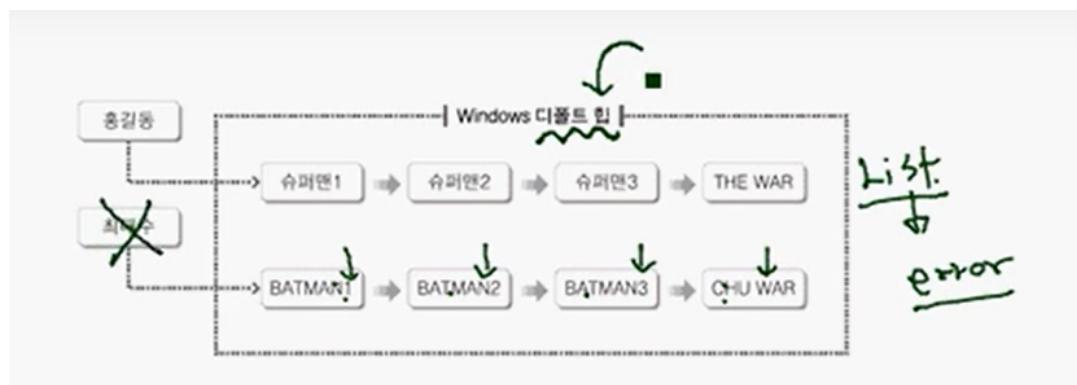


코드, 데이터, 스택, 디폴트 힙 + 여유 공간

이 여유공간을 가상 메모리를 할당할 수도 있고

Dynamic Heap을 할당할 수도 있다.

▼ 디폴트 힙의 단점



예를 들어, 하나의 디폴트 힙에

홍길동, 최대수가 빌린 책의 목록을 저장한다고 해보자.

보통 list 형태로 저장한다.

그러던 와중

최대수가 회원탈퇴를 하게 되면

해당 list를 다 돌면서 일일히 해제해주는 작업을 거쳐야 한다.

왜냐하면, 같은 공간 안에 서로 다른 사람들의 데이터를

저장해두었기 때문에

그 사람의 데이터를 선별해서 지워주는 과정을 거쳐야 하는 것이다.

이러한 과정속에서

에러가 발생할 확률이 높다.

▼ 힙컨트롤 ? 다이나믹 힙 ?

디폴트힙 외에

윈도우 시스템은 별도의 힙을 추가로 만들 수 있게 해준다.

이것이 바로 힙 컨트롤

Dynamic Heap 기법이다.



윈도우즈는 디폴트 힙 이외에

별도의 힙을 만들 수 있게 제공해준다.

이게 바로 힙 컨트롤이다.

Dynamic Heap 기법이라고도 한다.

홍길동, 최대수를 위한
별도의 힙을 만들어주고
각각의 고유한 데이터를 할당

만약 최대수와 관련된 데이터를 지워주려고 한다면
최대수 전용 힙
그냥 힙 자체를 삭제해주면 된다라는 의미이다.

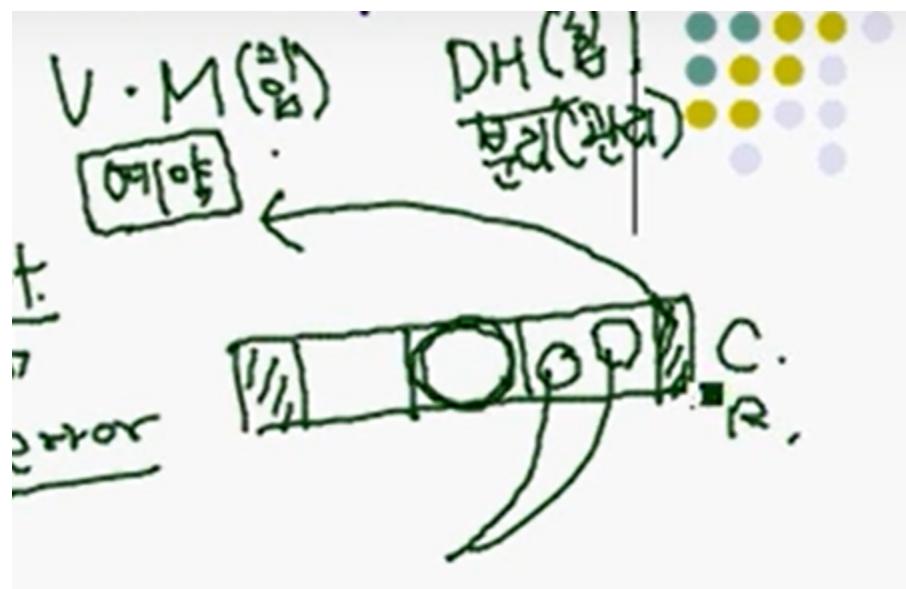
▼ 가상 메모리와 다이나믹 힙의 차이

가상 메모리는 메모리 예약

즉, 메모리를 아껴보겠다는 취지

Dynamic Heap은 보다
데이터를 분리 ! 관리 !의 측면이 강하다.

▼ 저자가 가상 메모리 또한 힙이라고 생각하는 이유



처음에 할당된 디플트 힙외에

그 여유 공간으로부터 메모리를 할당받아서

해당 메모리를 commit으로 둘지, free로 둘지, reserve로 둘지

를 결정하는데

이 과정은 런타임 과정에서 발생한다.

힙 또한 그 크기가 런타임에서 결정되기 때문에

이러한 맥락에서 그리 생각

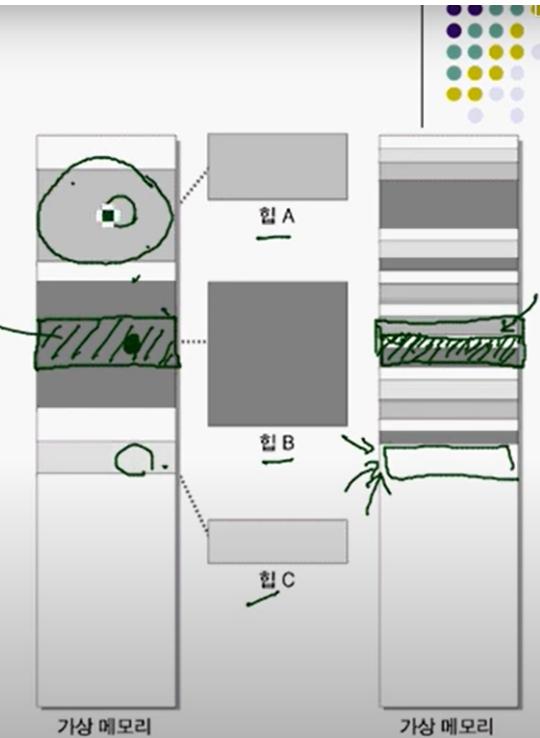
▼ 장점 2가지

Dynamic Heap의 이점

- 메모리 단편화 해소
 - 프로그램의 로컬리티가 낮아지는 문제점 발생
- 동기화 문제에서 자유
 - 쓰레드별로 힙을 생성할 수 있다.

할당
할당

OS
int a;



1) 메모리 단편화 해소 ?

자. 하나의 공통된 공간에 메모를 할당, 해제, 할당 (오른쪽 메모리 블록)

하는 과정을 거치다 보면 빈공간도 많이 생기고

그러다보면 충분히 큰 메모리 공간을 하나의 쓰레드에

할당하지 못할 수 있고

할당받은 메모리 공간이 크기 못하면

로컬리티도 낮아질 것

2) 동기화 (할당 동기화) 를 의미

이는 우리가 지금까지 다룬

int a라는 변수에 여러 쓰레드가 접근하는 이런 동기화 문제가 아니다

왼쪽 그림과 같이

쓰레드 A,B,C 에 각자의 전용 힙을 따로 할당한다.

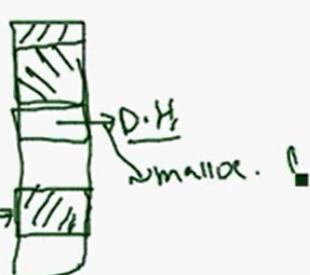
동기화 문제란, 같은 메모리 공간을 할당하려고 하는 것인데

이렇게 각자 별도의 공간을 분리해서 할당하는 것을
보장하게 된다면, 이러한 동기화 문제도 해결될 것이다.

▼ 관련함수 4개

Dynamic Heap 생성 및 할당

- 힙의 생성 및 소멸
 - HeapCreate: 힙의 생성
 - HeapDestory: 힙의 소멸
- 생성된 힙 내의 메모리 할당 및 해제
 - HeapAlloc: 힙 내에 메모리 할당
 - HeapFree: 힙 내에 메모리 반환



▼ malloc, free 함수와의 차이

이 둘은 디폴트 힙에 할당, 해제를 다룬다.

▼ MMF (Memory Mapped File)

▼ 개념

MMF 개념



메모리를 파일에 매핑 시키겠다

파일의 일부 메모리 공간을 프로세스의 가상 메모리 공간

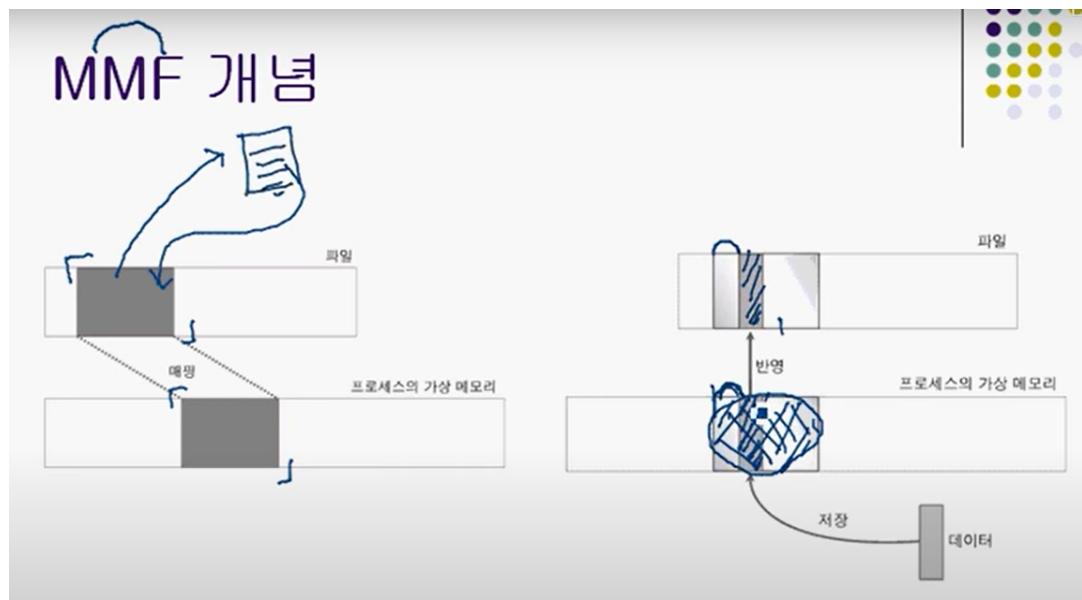
일부에 매핑을 시킨다.

해당 가상 메모리 공간 특정 위치에 변화가 생긴다면

파일에서도 해당 공간의 위치에 맞게 (offset에 맞게)

변화가 반영될 것이다.

▼ 장점 2



1) 메모리 write의 관점

우리가 파일에 있는 데이터들을 정렬하고 싶다면

MMF를 안쓴다면

파일에 있는 메모리를 다 불러와서

메인 메모리에 올리고

정렬하고

다시 저장하는 과정을 거쳐야 한다.

MMF에서는

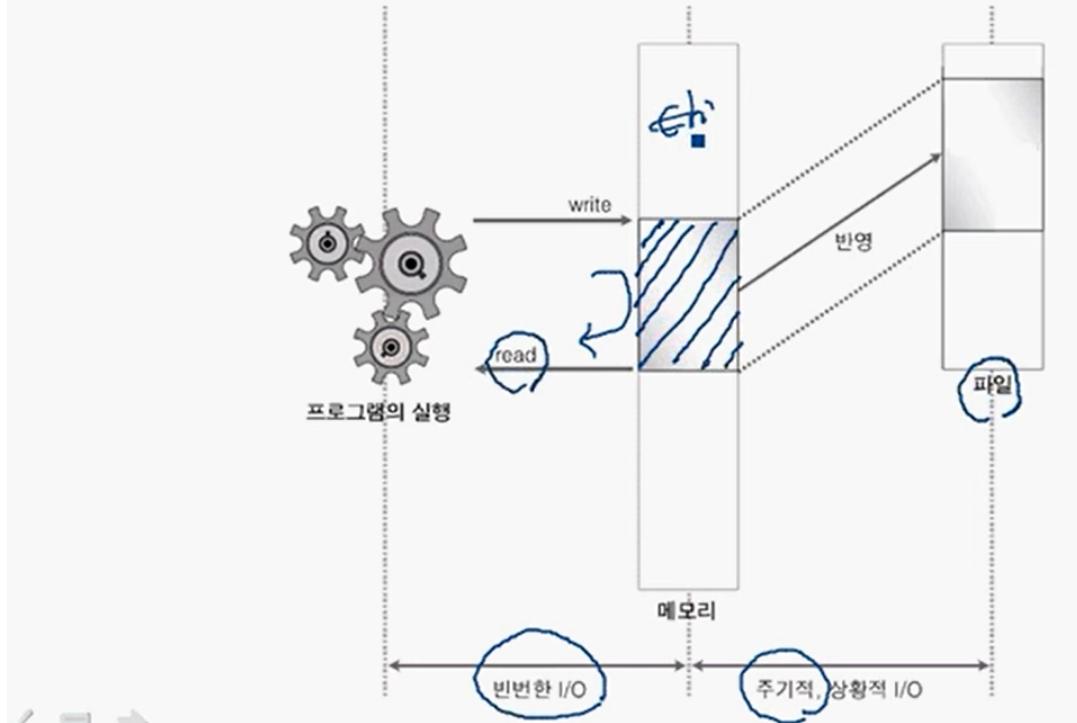
그냥 해당 프로세스 가상 메모리 상에서

정렬만 수행해주면

그 결과가 알아서 파일 메모리에 반영될 것이다.

2) 파일 데이터 캐싱

MMF 장점



그런데 생각해보면 매번 메인 메모리에 쓴 내용을 파일에 써줄 필요는 없다.

어차피 가장 최신의 데이터는 메모리에 유지하면서 빈번하게 write, read를 수행하게 될 것이다.

빈번한 I/O가 발생하는 곳에서는 (메인 메모리)
가장 최근의 데이터를 유지해주고

실질적으로 최신의 데이터가 저장이 된다라는
보장만 되는 상황에서는
해당 파일에 write 하는 과정을
가끔, 주기적으로 해줘도 된다.

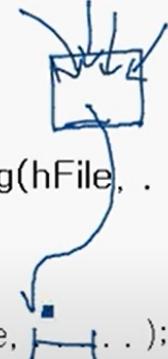
파일에서 데이터를 read, write 하려면
매우 오래 걸린다.

그저 메모리 공간에 캐시만 해주고
그것을 가끔 파일에 넣을 수 있게
반영만 해주면 된다는 것이다.

▼ 관련 3개 함수 및 특성

MMF 구현과정

- 1단계: 파일의 생성
`HANDLE hFile= CreateFile(. . . .);`
- 2단계: 파일 연결 오브젝트 생성 정보,
`HANDLE hMapFile=CreateFileMapping(hFile,);`
- 3단계: 가상 메모리에 파일 연결
`TCHAR * pWrite=MapViewOfFile(hMapFile,);`



먼저 파일을 만들고

파일의 정보를 정보를 담은 파일 연결 오브젝트를 만들고

해당 파일 연결 오브젝트를 인자로 전달해서
어디서부터 어디까지 가상 메모리에 매핑해주세요 ! 라고 요청하는 구조

▼ 개발

WidgetComponent

Window, Widget, Button Clone