3D게임프로그래밍
-CHAPTER1-

SOULSEEK

목차

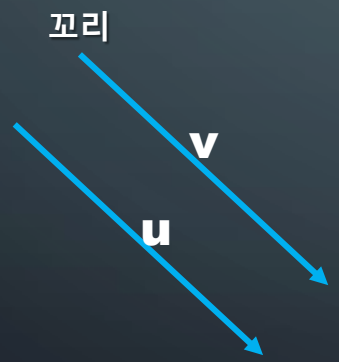
3D에 필요한 수학

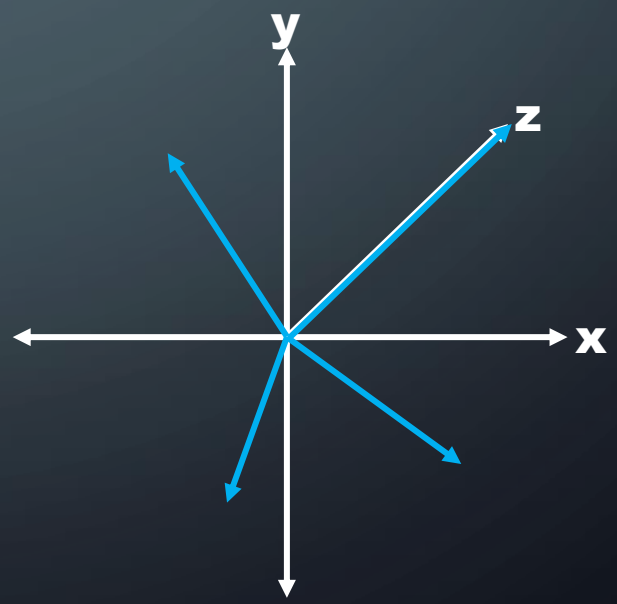
1. 3D에 필요한 수학

벡터(Vector)

- 길이와 가리키는 방향의 속성을 가진 선분
- 위치에 대한 속성은 가지지 않기 때문에 위치가 다르지만 동일한 길이와 방향(평행)을 가진 벡터는 서로 동일한 것이다.
- 좌표 시스템의 원점에 꼬리가 일치 하도록 벡터들을 평행하게 배치 할 수 있으며 벡터의 꼬리가 원점과 일치하면 벡터가 표준점에 위치한 것이 된다.

u = (u_x, u_y) – 2차원 벡터**, N = (N_x, N_y, N_z) - 3**차원 배열....





1. 3D에 필요한 수학

영 벡터

- 모든 성분이 **0**인 벡터 – **(0, 0, 0)**

단위 벡터

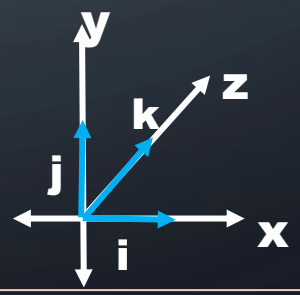
- 크기가 **1**인 벡터를 말한다**.**

기저 벡터

- **3**개의 선분이 직교하는 벡터 – 비표준 기저**,** 비정규 직교 기저
- 각 축**(x, y, z)**에 대한 단위벡터의 집합 – 표준 기저**,** 정규 직교 기저
- 절대 좌표계는 표준 기저를 사용하는 좌표계
- 상대 좌표계는 비표준 기저를 사용하는 좌표계

기저벡터

i = (1, 0, 0)
j = (0, 1, 0)
k = (0, 0, 1)



1. 3D에 필요한 수학

D3DX 라이브러리 Vector클래스

D3DXVECTOR3 Class

```
typedef struct D3DXVECTOR3 : public D3DVECTOR
{
    public:
        D3DXVECTOR3() {};
        D3DXVECTOR3( CONST FLOAT* );
        D3DXVECTOR3( CONST D3DVECTOR&);
        D3DXVECTOR3( FLOAT x, FLOAT y, FLOAT z);

        //형 변환
        operator FLOAT* ();
        operator CONST FLOAT* () const;

        //할당 연산자
        D3DXVECTOR3& operator   +=
                                ( CONST D3DXVECTOR3&);
        D3DXVECTOR3& operator   -=
                                ( CONST D3DXVECTOR3&);
        D3DXVECTOR3& operator   *=      ( FLOAT );
        D3DXVECTOR3& operator   /=      ( FLOAT );

        //단항 연산자
        D3DXVECTOR3 operator + () const;
        D3DXVECTOR3 operator - () const;

        //이항 연산자
        D3DXVECTOR3& operator
                + ( CONST D3DXVECTOR3& ) const;
        D3DXVECTOR3& operator
                - ( CONST D3DXVECTOR3& ) const;
        D3DXVECTOR3& operator * ( FLOAT ) const;
        D3DXVECTOR3& operator / ( FLOAT ) const;

        friend D3DXVECTOR3 operator
                        * ( FLOAT, CONST struct D3DVECTOR3& );

        BOOL operator == ( CONST D3DXVECTOR3& ) const;
        BOOL operator != ( CONST D3DXVECTOR3& ) const;

} D3DXVECTOR3, *LPD3DXVECTOR3;
```

1. 3D에 필요한 수학

벡터 상등

- 두 개의 벡터의 x, y, z값이 일치할 때를 말한다.
- 기하학적으로는 같은 방향과 같은 길이를 가지는 두 벡터는 동일하다고 하지만, 대응성분이 동일할 것은 아니다.
- $u_x = v_x, u_y = v_y, u_z = v_z$ 라면 $(u_x, u_y, u_z) = (v_x, v_y, v_z)$ 이다.

```
D3DXVECTOR3     u(1.0f, 0.0f, 1.0f);
D3DXVECTOR3     v(0.0f, 1.0f, 0.0f);

If(u == v)          // 같은지 체크
      return true;

If(u != v)          // 다른지 체크
      return true;
```

1. 3D에 필요한 수학

벡터의 크기 계산

U라는 벡터의 크기를 구할 때 ➔ $\|u\| = \sqrt{u_x^2 + u_y^2 + u_z^2}$ ➔ **u**의 **두 수직 막대는 크기를 표시하기 위한** 것이기 때문**.**

문제**1. U = (1, 2, 3)**과 **V = (1, 1)**의 크기를 구하라**.**

D3DX 라이브러리를 이용한 벡터의 크기

FLOAT D3DXVec3Length(CONST D3DXVECTOR* pV); ➔길이를 계산하려는 벡터의 크기를 리턴한다**.**

```
D3DXVECTOR3 v(1.0f, 2.0f, 3.0f);

Float magnitude = D3DXVec3Length(&v);       // = sqrt(14);
```

1. 3D에 필요한 수학

벡터의 정규화

- 벡터의 크기를 1로 만들어 단위 벡터가 되도록 하는 것.
- 각 성분을 벡터의 크기로 나누면 구할 수 있다.
- 공식 ➔ $\hat{u} = \frac{u}{\|u\|} = (\frac{u_x}{\|u\|}, \frac{u_y}{\|u\|}, \frac{u_z}{\|u\|})$

문제 **2 : u = (1, 2, 3)**과 **v = (1, 1)**벡터를 정규화하라**.**

D3DX 라이브러리를 이용한 정규화

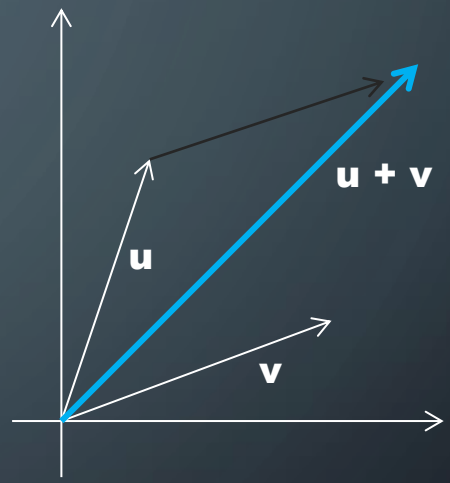
D3DXVECTOR3* D3DXVec3Normalize(D3DXVECTOR3* pOut, CONST D3DXVECTOR3* pV);

1. 3D에 필요한 수학

벡터 더하기

- 대응되는 성분을 더하면 두 개의 벡터를 더할 수 있다.
- 더하고자 하는 벡터는 반드시 **동일한 차원**을 가져야 한다.

공식 ➔ **u + v =** $(u_x + v_x, u_y + v_y, u_z + v_z)$



D3DX 라이브러리를 이용한 벡터 더하기

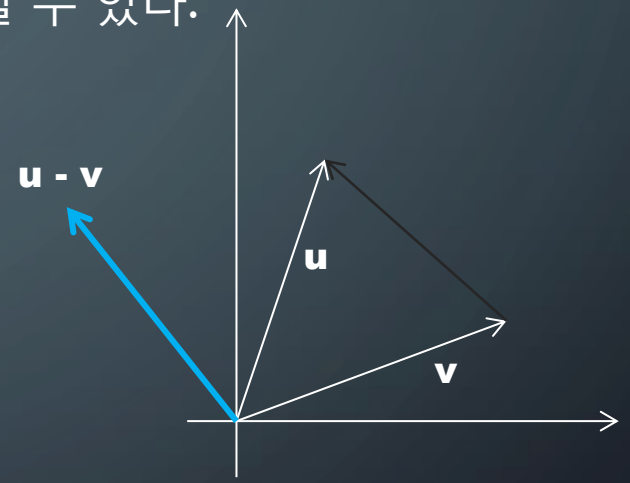
```
D3DVECTOR3 u(2.0f, 0.0f, 1.0f);
D3DVECTOR3 v(0.0f, -1.0f, 5.0f);

D3DVECTOR3 sum = u + v; //오버로드된 덧셈 연산자를 이용한다.
```

1. 3D에 필요한 수학

벡터 빼기

- 벡터의 대응되는 성분을 빼는 방법으로 벡터 빼기를 수행할 수 있다.

공식 ➔ $u - v = u + (-v) = (u_x - v_x, u_y - v_y, u_z - v_z)$



D3DX 라이브러리를 이용한 벡터의 빼기

```
D3DXVECTOR3 u(2.0f, 0.0f, 1.0f);
D3DXVECTOR3 v(0.0f, -1.0f, 5.0f);

D3DXVECTOR3 defference = u – v;
```

1. 3D에 필요한 수학

벡터의 스칼라 곱

- 스칼라와 의 곱을 통해 벡터의 배율이 변경된다.
- 음수를 이용해 벡터의 방향을 반전 시킬 수 있다.

공식 ➔ ku = (ku_x, ku_y, ku_z)

D3DX 라이브러리를 이용한 벡터의 스칼라 곱

```
D3DXVECTOR3 u(1.0f, 1.0f, -1.0f);
D3DXVECTOR3 scaledVec = u * 10.0f;
```

1. 3D에 필요한 수학

벡터의 내적

- 방향을 가진 두 선분이 이루는 각을 말한다.
- 내적의 특성을 이용해서 사물이 있는지 없는 지 판단하는데 이용한다.

공식 ➔ **u • v =** $u_x v_x + u_y v_y + u_z v_z$ **= s**

u • v = $\|u\|\|v\|$ ➔ **코사인 법칙에 의거**

➔ **두 벡터간의 내적이 벡터 크기배율을 가진 벡터 간 각도의 코사인임을 알 수 있다.**

➔ **u와 v가 모두 단위 벡터일 경우 u • v 는 두 벡터 간 각도의 코사인이 된다.**

u • v = 0 라면, u와 v는 수직이다.
u • v > 0 라면, 두 벡터 간의 각도 θ**는 90도 보다 작다.**
u • v < 0 라면, 두 벡터 간의 각도 θ**는 90도 보다 크다.**

D3DX 라이브러리를 이용한 벡터의 내적

```
FLOAT D3DVec3Dot(CONST D3DXVECTOR3* pV1, CONST D3DXVECTOR3* pV2);

D3DXVECTOR3 u(1.0f, -1.0f, 0.0f);
D3DXVECTOR3 v(3.0f, 2.0f, 1.0f);

Float dot = D3DXVec3Dot(&u, &v);
```

1. 3D에 필요한 수학

벡터의 외적

- 두 벡터와 직각을 이루는 벡터를 구하는 것.
- 일반적인 곱의 교환법칙은 성립되지 않는다.
- 평면의 법선 벡터(바라보는 방향)와 적 캐릭터가 어느 쪽에 있는지, 캐릭터가 벽에 부딪혀서 미끄러지는 방향이 어디인지 알아 볼 때 사용한다.

공식 ➔ **p = u * v =** $[(u_y v_z - u_z v_y), (u_z v_x - u_x v_z), (u_x v_y - u_y v_x)]$

➔ $P_x = (u_y v_z - u_z v_y)$, $P_y = (u_z v_x - u_x v_z)$, $P_z = (u_x v_y - u_y v_x)$

ex) j = k + l = (0, 0, 1) * (1, 0, 0)을 찾고 **j**가 **k**와 **i**에 직각임을 확인해보자.

➔ j_x **= (0(0) – 1(0)) = 0,** j_y **= (1(1) – 0(0)) = 1,** j_z **= (0(0) – 0(1)) = 0**

내적을 활용하면 **j • k = 0, j • l = 0**이므로 **j**는 **k**와 **i**에 모두 직각임을 알 수 있다.

D3DX 라이브러리를 이용한 벡터의 외적

```
D3DXVECTOR3* D3DXVec3Cross(D3DXVECTOR3* pOut,
                           CONST D3DXVECTOR3* pV1, CONST D3DXVECTOR3* pV2);
```

1. 3D에 필요한 수학

변환

D3DX에서는 기본적으로 4x4행렬을 이용한다. ➔ 모든 변환에 대응할 수 있는 행렬 형태
이동할 벡터 또한 1x4행렬로 표시되어야 한다.

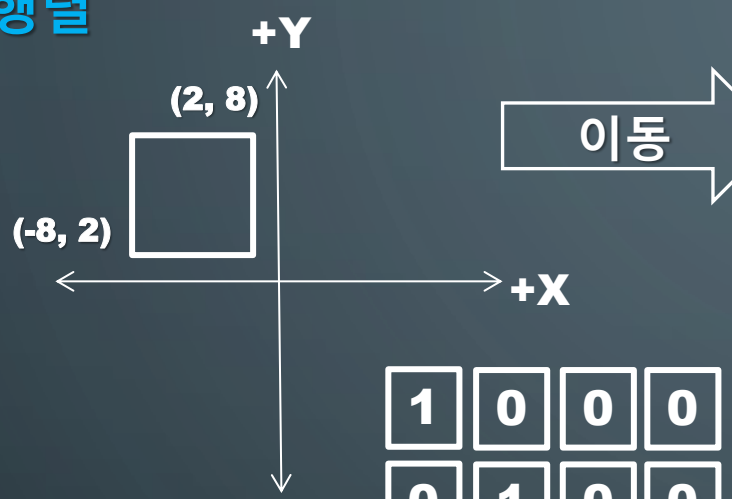
벡터가 이동하려면 네 번째 성분이 1이 되어야 하며 0이 되면 움직이지 않는다.

➔ $\mathrm{p} = [p_1 \quad p_2 \quad p_3 \quad 1]$ ➔ 이동
➔ $\mathrm{v} = [v_1 \quad v_2 \quad v_3 \quad 1]$ ➔ 이동불가

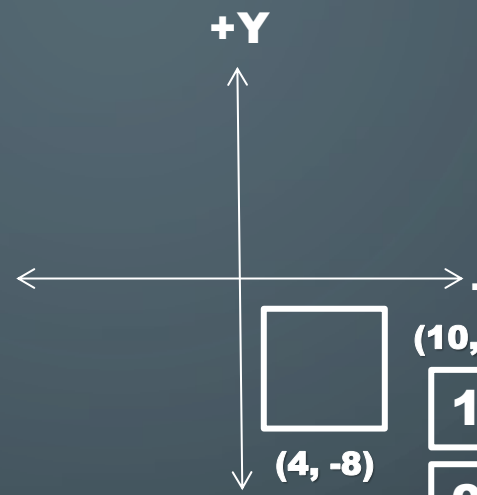
4행으로 보충된 벡터를 **동치벡터**라고 한다.

1. 3D에 필요한 수학

이동행렬



이동



(10, -2)

1	0	0	0
0	1	0	0
0	0	1	0
X	Y	Z	1

x, y, z가 이동에 관련된 부분이다.

1	0	0	0
0	1	0	0
0	0	1	0
-X	-Y	-Z	1

부호만 바꿔서 이동행렬의 역행렬을 만들 수 있다.

D3DX 라이브러리에서 이동행렬

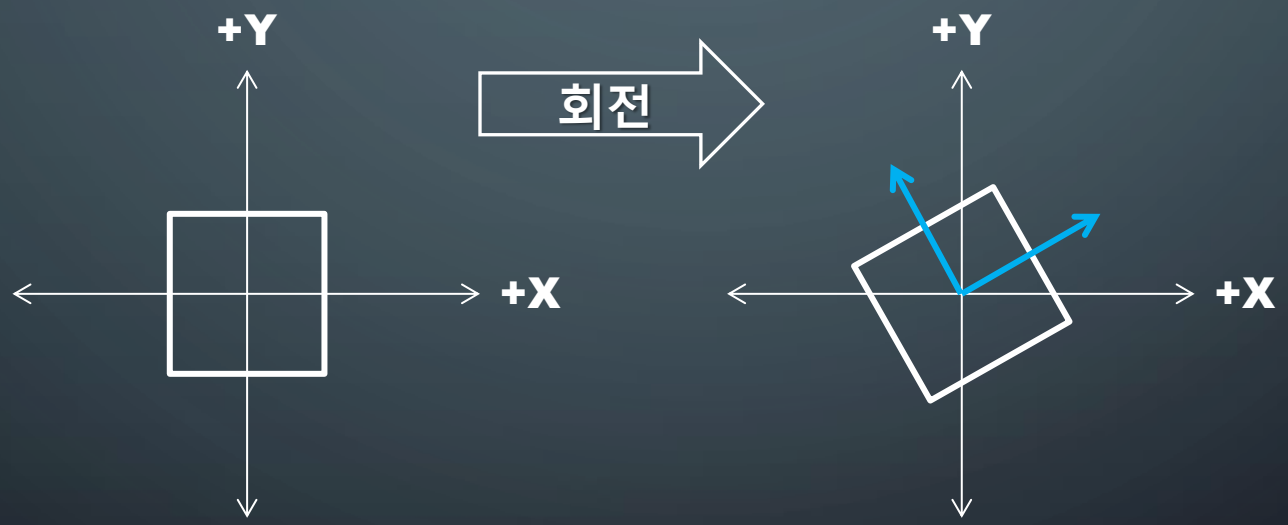
D3DXMATRIX* D3DXMatrixTranslation(D3DXMATRIX* pOut, FLOAT x, FLOAT y, FLOAT z);

- 첫 번째 행렬로 결과값을 전달해 준다.

1. 3D에 필요한 수학

회전행렬

- 행렬을 이용해 **x, y, z**에서 벡터 라디안을 회전 시킬 수 있다.
- 각도는 회전축을 내려다 볼 때 시계 방향으로 측정한다.
- 어떤 행렬의 역은 전치이기 때문에 직각인 행렬이 된다.



D3DX 라이브러리에서는 각 축에 대한 회전을 하는 함수를 제공한다.

1. 3D에 필요한 수학

축 행렬

$X(\theta)$ =

1	0	0	0
0	$\cos\theta$	$\sin\theta$	0
0	$-\sin\theta$	$\cos\theta$	0
0	0	0	1

D3DXMATRIX* D3DXMatrixRotationX(D3DXMATRIX* pOut, FLOAT Angle);
➔ 첫 번째 인자는 결과 , 두 번째 인자는 라디안으로 측정한 회전각

$Y(\theta)$ =

$\cos\theta$	0	$-\sin\theta$	0
0	1	0	0
$\sin\theta$	0	$\cos\theta$	0
0	0	0	1

D3DXMATRIX* D3DXMatrixRotationY(D3DXMATRIX* pOut, FLOAT Angle);
➔ 첫 번째 인자는 결과 , 두 번째 인자는 라디안으로 측정한 회전각

$Z(\theta)$ =

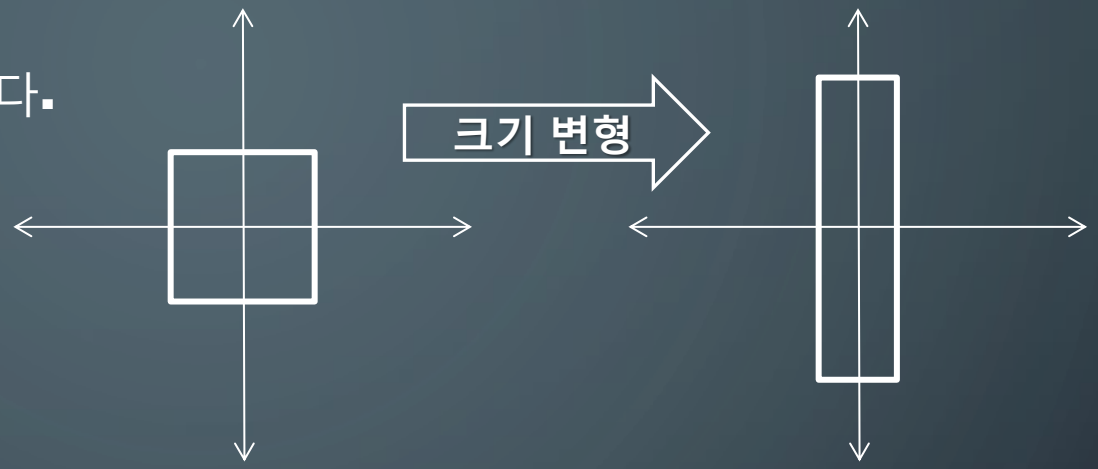
$\cos\theta$ 1	$\sin\theta$	0	0
$-\sin\theta$	$\cos\theta$	0	0
0	0	1	0
0	0	0	1

D3DXMATRIX* D3DXMatrixRotationZ(D3DXMATRIX* pOut, FLOAT Angle);
➔ 첫 번째 인자는 결과 , 두 번째 인자는 라디안으로 측정한 회전각

1. 3D에 필요한 수학

크기 변형 행렬

- 벡터와 의 곱으로 크기를 변경한다.
- 역행렬은 인자의 역으로 구할 수 있다.



$$S(q) = \begin{matrix} q_x & 0 & 0 & 0 \\ 0 & q_y & 0 & 0 \\ 0 & 0 & q_z & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

역행렬...

$$S(q) = \begin{matrix} \frac{1}{q_x} & 0 & 0 & 0 \\ 0 & \frac{1}{q_x} & 0 & 0 \\ 0 & 0 & \frac{1}{q_x} & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

D3DX 라이브러리에서 크기 변형 행렬

D3DXMATRIX* D3DXMatrixScaling(D3DXMATRIX* pOut, FLOAT sx, FLOAT sy, FLOAT sz);
첫 번째 인자는 결과..

1. 3D에 필요한 수학

변환 조합하기

- 여러 가지 변환이 적용 되었을 때 변환 되는 순서가 있다.
- 이 모든 변환은 4x4행렬 안에서 이루어진다, 변환 순서에 맞게 행렬을 곱해서 한번에 적용해 주면 원한은 변환이 된다.
- 스케일링➔자전➔이동➔공전➔부모(계층구조로 연결 되어 있다면)

ex)벡터 $p = [5,0,0,1]$을 모든 축으로 1/5의 크기로 배율을 변경한 후, y축으로 $\pi/4$라디안 만큼 회전 시킨 다음, x축으로 1단위, y축으로 2단위, z축으로 -3단위 만큼 이동하라.

➔크기 변형과 y축 회전, 그리고 이동을 수행해야 한다.

➔ $ps = [5,0,0,1]\begin{matrix}\frac{1}{5} & 0 & 0 & 0\\ 0 & \frac{1}{5} & 0 & 0\\ 0 & 0 & \frac{1}{5} & 0\\ 0 & 0 & 0 & 1\end{matrix} = [1,0,0,1]$➔ p' ➔ $pR_y = [1,0,0,1]\begin{matrix}.707 & 0 & -.707 & 0\\ 0 & 1 & 0 & 0\\ .707 & 0 & .707 & 0\\ 0 & 0 & 0 & 1\end{matrix}$

➔ $[.707, 0, -.707, 1]$ ➔ $p''T = [1.707, 2, -3.707, 1]$

➔ $SR_yT =$ $\begin{matrix}\frac{1}{5} & 0 & 0 & 0\\ 0 & \frac{1}{5} & 0 & 0\\ 0 & 0 & \frac{1}{5} & 0\\ 0 & 0 & 0 & 1\end{matrix}$ x $\begin{matrix}.707 & 0 & -.707 & 0\\ 0 & 1 & 0 & 0\\ .707 & 0 & .707 & 0\\ 0 & 0 & 0 & 1\end{matrix}$ x $\begin{matrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 1 & 2 & -3 & 1\end{matrix}$ = $\begin{matrix}.1414 & 0 & -.1414 & 0\\ 0 & 1 & 0 & 0\\ .1414 & 0 & .1414 & 0\\ 1 & 2 & -3 & 1\end{matrix}$

1. 3D에 필요한 수학

D3DX 라이브러리에서 지원하는 변환 함수

변환이 적용된 행렬을 이용해 벡터의 위치를 변환할 때 사용

```
D3DXVECTOR3* D3DXVec3TransformCoord(D3DXVECTOR3* pOut,      // 결과
                                    CONST D3DXVECTOR3* pV,  // 변환할 포인터
                                    CONST D3DXMATRIX* pM);  // 변환 행렬
```

```
D3DXMATRIX T(....);
D3DXVECTOR3 p(...);
D3DXVectorTransformCoord(&p, &p, &T); // 포인트를 변환한다.
```

벡터의 위치는 변환하지 않고 벡터의 변환만 적용할 때 사용.

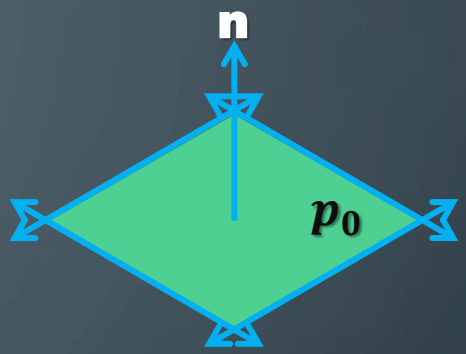
```
D3DXVECTOR3* WINAPI D3DXVec3TransformNormal( D3DXVECTOR3* pOut,   // 결과
                                             CONST D3DVECTOR3* pV,          // 변환할 벡터
                                             CONST D3DMATRIX* pM);          // 변환 행렬
```

```
D3DXMATRIX T(....);
D3DXVECTOR3 v(...);
D3DXVectorTransformNormal (&v, &v, &T); // 벡터를 변환한다.
```

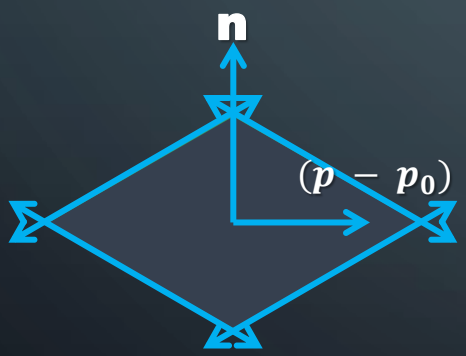
1. 3D에 필요한 수학

평면

- 노말 벡터의 방향이 같은 점들의 모임
- 하나의 벡터 **n**과 평면상의 포인트 p_0 으로 표현한다.
- 벡터 **n**은 법선 벡터라고 부르며 평명과 수직을 이룬다.



➔ $\mathrm{n} \cdot (\mathrm{p} - p_0) = 0$ ➔$(\mathrm{p} - p_0)$으로 구성한 벡터가 **평면의 법선 벡터와 직각일 경우 벡터 p역시 평면 위에 존재**한다.



D3DX 라이브러리에서는 **D3DXPLANE**이라는 구조체로 평면을 표현한다.

1. 3D에 필요한 수학

포인트와 평면 공간 관계

ex)평면**(n, d)**가 있다고 할 때**,** 포인트 **p**가 평면과 어떤 관계에 있는지 확인 할 수 있다**.**

만약 $n \cdot p + d = 0$ 라면, p는 평면과 **공면**에 있다.
만약 $n \cdot p + d > 0$ 라면, p는 평면의 **앞쪽 양의 절반(평면의 안쪽)** 공간에 있다.
만약 $n \cdot p + d < 0$ 라면, p는 평면의 **뒤쪽 음의 절반(평면의 바깥쪽)** 공간에 있다.
만약 평면의 법선 벡터 n이 단위 길이라면, $n \cdot p + d$로 평면에서 포인트 p까지의 부호를 가진 가장 짧은 거리를 얻을 수 있다.

D3DX 라이브러리에서 포인트와 평면의 관계

```
FLOAT D3DXPlaneDotCoord(CONST D3DXPLANE* pP,        // 평면
                        CONST D3DXVECTOR3* pP);     // 포인트

D3DXPLANE p(0.0f, 1.0f, 0.0f, 0.0f);
D3DXVECTOR3 v(3.0f, 5.0f, 2.0f);

float x = D3DXPlaneDotCoord(&P, &V);

if(x approximately equals 0.0f)    // v는 평면과 공면 이다.
if(x > 0)                          // v는 양의 절반 공간에 있다.
if(x < 0)                          // v는 음의 절반 공간에 있다.
```

1. 3D에 필요한 수학

평면 구축

법선 **n**과 평면 상의 알려진 포인트 p_0이 있다고 하면 **d**성분의 값은

공식 ➔ $n \cdot p_0 + d = 0$ ➔ $n \cdot p_0 = -d$ ➔ $-n \cdot p_0 = d$

D3DX 라이브러리에서..

```
D3DXPLANE* D3DXPlaneFromPointNormal
                    (D3DXPLANE* pOut,                   // 결과
                    CONST D3DXVECTOR3* pPoint           // 평면 상의 포인트
                    CONST D3DXVECTOR3* pNormal);        // 평면의 법선
```

세개의 포인트 p_0, p_1, p_2가 있다면, 평면상의 두 개의 벡터를 구성 할 수 있다.

u = $p_1 - p_0$**, v =** $p_2 - p_0$ ➔ **n = u * v ➔ -(**$n \cdot p_0$**) = d** 가 된다.

D3DX 라이브러리에서..

```
D3DXPLANE* D3DXPlaneFromPoints
                    (D3DXPLANE* pOut,                   // 결과
                    CONST D3DXVECTOR3* pV1,             // 평면상 포인트 1
                    CONST D3DXVECTOR3* pV2,             // 평면상 포인트 2
                    CONST D3DXVECTOR3* pV3);            // 평면상 포인트 3
```

1. 3D에 필요한 수학

D3DX 라이브러리에서 평면을 지원하는 함수

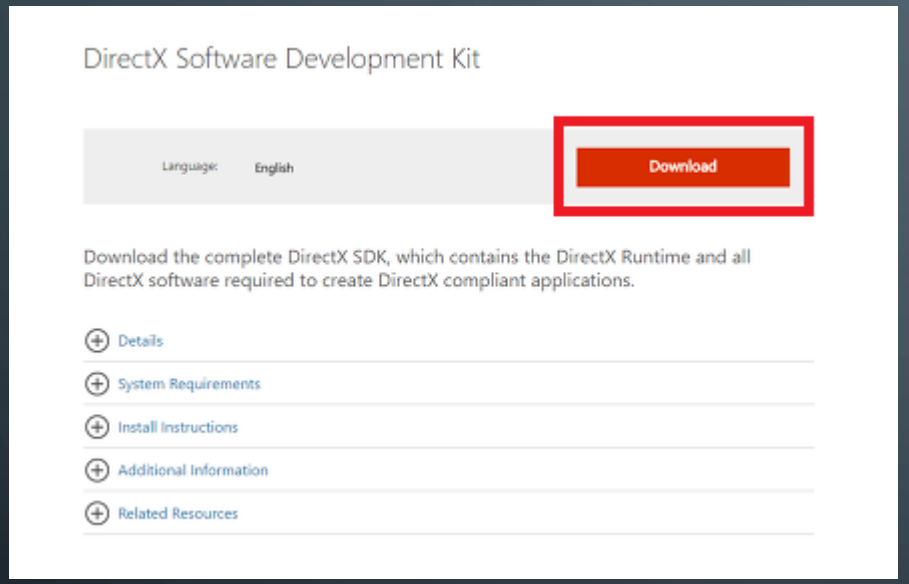
```
//평면의 법선 벡터 정규화
D3DXPLANE* D3DXPlaneNormalize
                        (D3DXPLANE* pOut,          // 정규화된 결과 평면
                        CONST D3DXPLANE* pP);      // 입력 평면

//평면의 변환
D3DXPLANE* D3DXPlaneTransform
                        (D3DXPLANE* pOut,          // 결과
                        CONST D3DPLANE* pP,        // 입력 평면
                        CONST D3DXMATRIX* pM);     // 변환 행렬
```

D3DX INIT

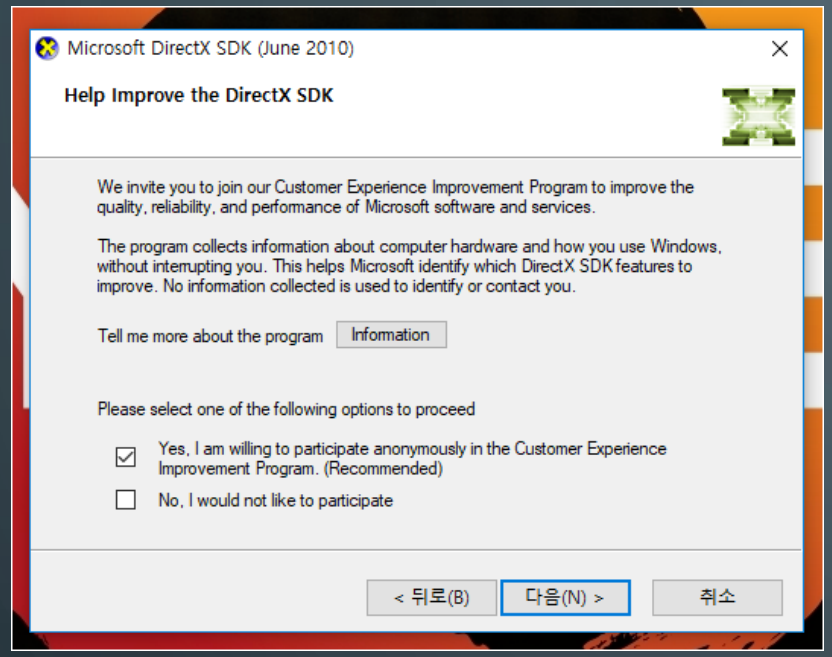
2. D3DX INIT

환경설정

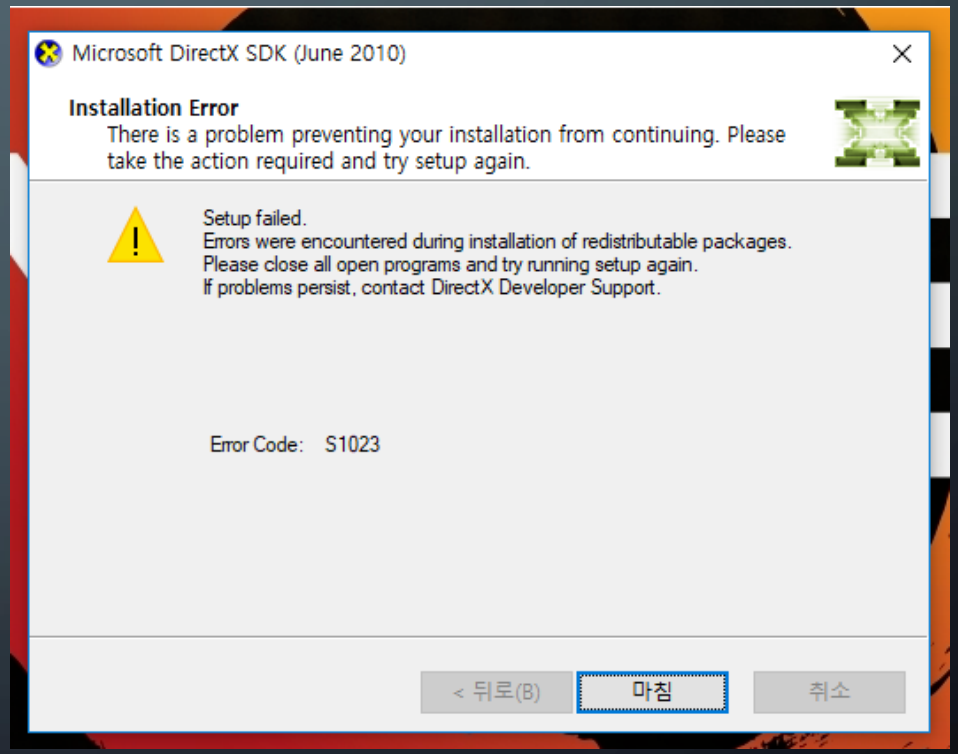
- **DirectX SDK** 설치
- **Download : https://www.microsoft.com/en-us/download/details.aspx?id=6812**



2. D3DX INIT



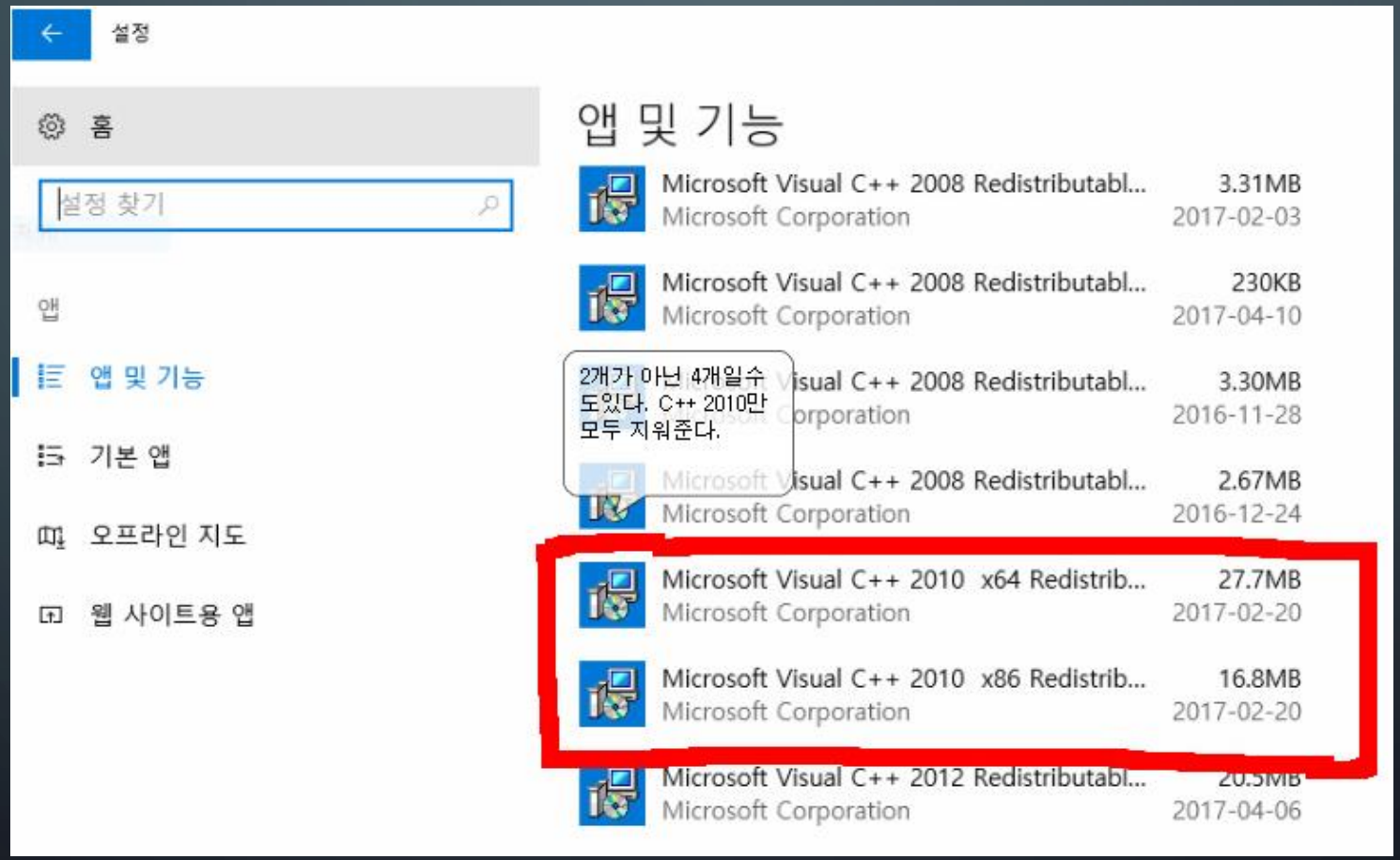
설치 중 이런 오류가 발생한다면..



2. D3DX INIT

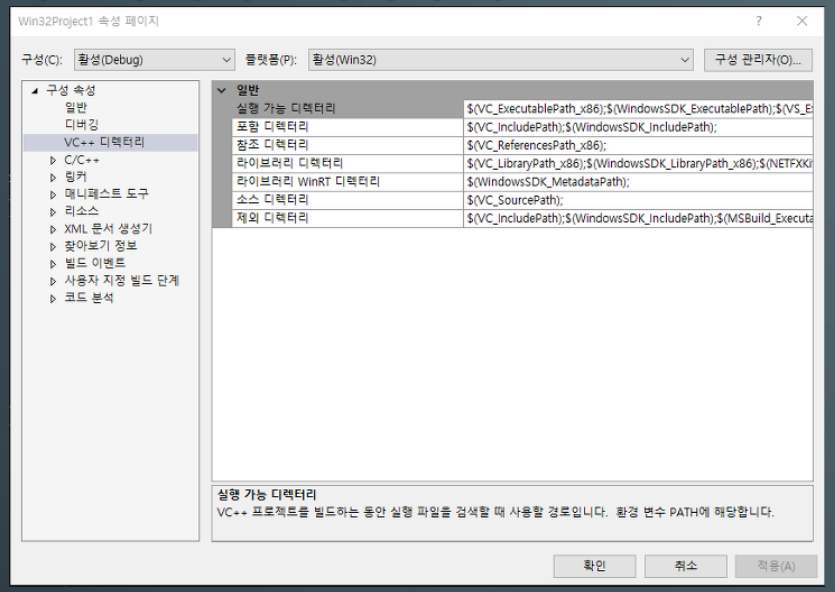
Win10 기준..

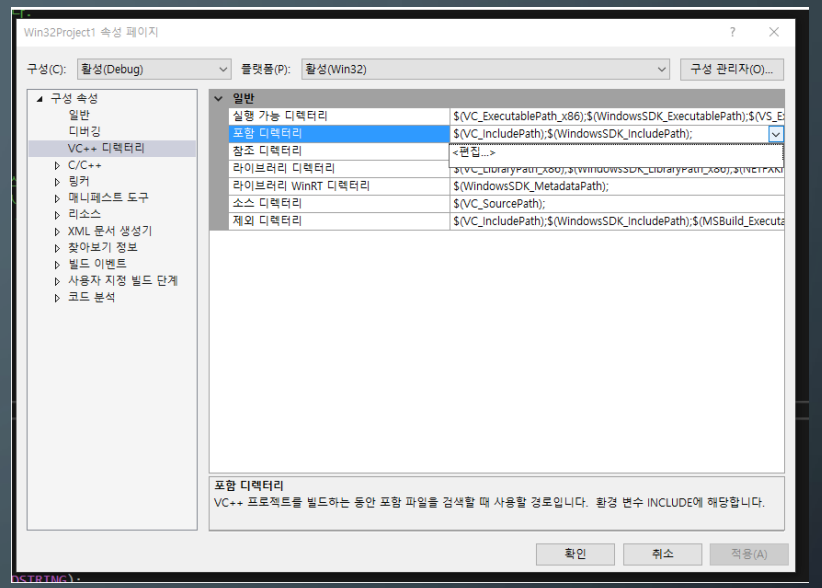
- 해당 파일을 삭제한 후 다시 SDK를 설치한다.

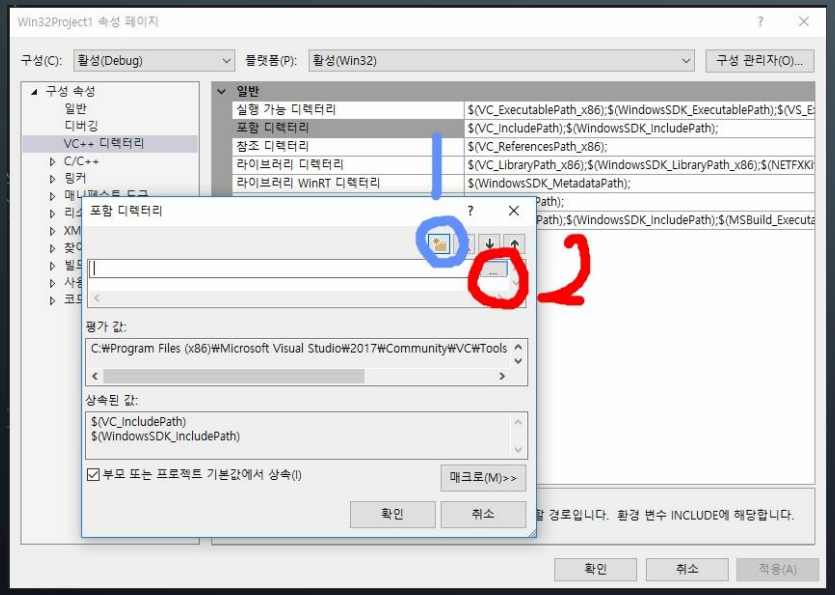


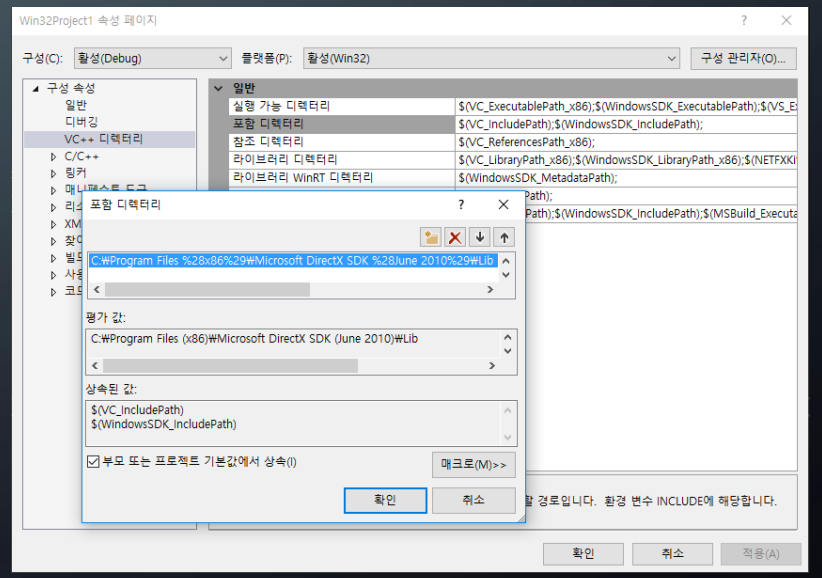
2. D3DX INIT

프로젝트에 라이브러리 추가

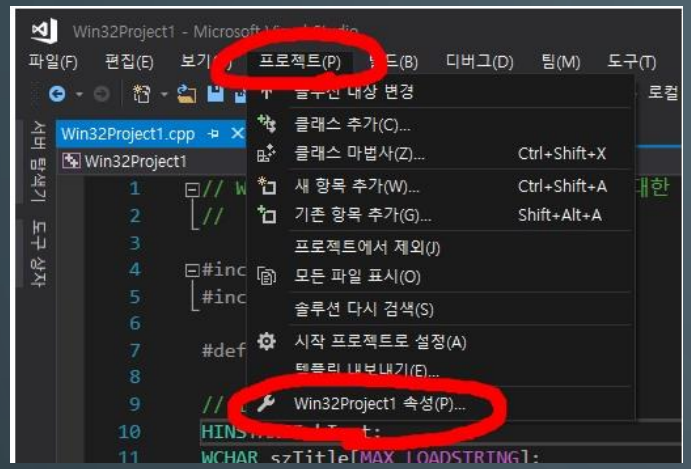


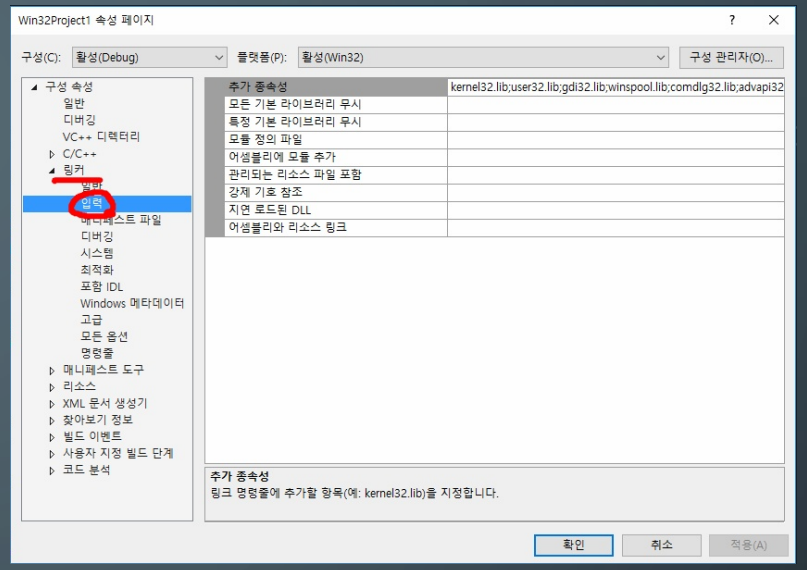


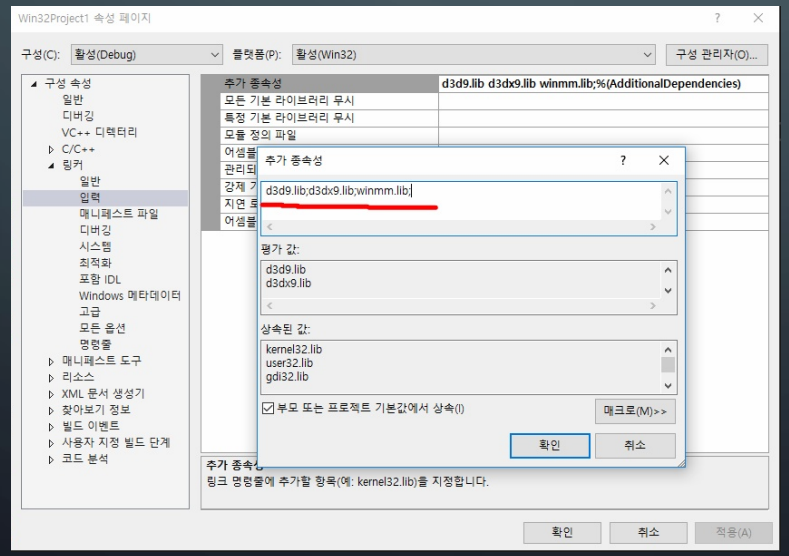




2. D3DX INIT


Win32Project1 - Microsoft
파일(F) 편집(E) 보기 프로젝트(P) 디버그(D) 팀(M) 도구(T)
Win32Project1.cpp
Win32Project1
클래스 추가(C)...
클래스 마법사(Z)... Ctrl+Shift+X
새 항목 추가(W)... Ctrl+Shift+A
기존 항목 추가(G)... Shift+Alt+A
프로젝트에서 제외(J)
모든 파일 표시(O)
솔루션 다시 검색(S)
시작 프로젝트로 설정(A)
Win32Project1 속성(P)...


Win32Project1 속성 페이지
구성(C): 활성(Debug)
플랫폼(P): 활성(Win32)
구성 관리자(O)...
구성 속성
일반
디버깅
VC++ 디렉터리
C/C++
링커
입력
매니페스트 파일
디버깅
시스템
최적화
포함 IDL
Windows 메타데이터
고급
모든 옵션
명령줄
매니페스트 도구
리소스
XML 문서 생성기
찾아보기 정보
빌드 이벤트
사용자 지정 빌드 단계
코드 분석
추가 종속성 kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi32
모든 기본 라이브러리 무시
특정 기본 라이브러리 무시
모듈 정의 파일
어셈블리에 모듈 추가
관리되는 리소스 파일 포함
강제 기호 참조
지연 로드된 DLL
어셈블리와 리소스 링크
추가 종속성
링크 명령줄에 추가할 항목(예: kernel32.lib)을 지정합니다.
확인
취소
적용(A)


Win32Project1 속성 페이지
구성(C): 활성(Debug)
플랫폼(P): 활성(Win32)
구성 관리자(O)...
추가 종속성 d3d9.lib d3dx9.lib winmm.lib;%(AdditionalDependencies)
추가 종속성
d3d9.lib;d3dx9.lib;winmm.lib;
평가 값:
d3d9.lib
d3dx9.lib
상속된 값:
kernel32.lib
user32.lib
gdi32.lib
부모 또는 프로젝트 기본값에서 상속(I)
매크로(M)>>
확인
취소
링크 명령줄에 추가할 항목(예: kernel32.lib)을 지정합니다.
확인
취소
적용(A)

2. D3DX INIT

- **WinMain, WinProc**은 여전히 활용**, D3DInit**과 **Render, Cleanup**함수가 추가된다.

WinMain

```
INT WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR, INT)
{
	//윈도우 클래스 등록
	WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS_CLASSDC, MsgProc, 0L, 0L,
			  GetModuleHandle(NULL), NULL, NULL, NULL, NULL,
			  L"D3D Init", NULL };

	//winclass 레지스터에 등록
	RegisterClassEx(&wc);

	HWND hWnd = CreateWindow(L"D3D Init", L"D3D Init", WS_OVERLAPPEDWINDOW, 100, 100, 1024, 768,
				 GetDesktopWindow(), NULL, NULL, wc.hInstance, NULL);

	if (SUCCEEDED(InitD3D(hWnd)))
	{
		ShowWindow(hWnd, SW_SHOWDEFAULT);
		UpdateWindow(hWnd);

		MSG msg;

		while (GetMessage(&msg, NULL, 0, 0))
		{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
		}
	}

	//등록된 레지스트 winclass 릴리즈.
	UnregisterClass(L"D3D Init", wc.hInstance);
	return 0;
}
```

1. **RegisterClassEx()**로 생성하고자 하는 윈도우의 클래스를 등록한다.
2. **CreateWindow()**로 윈도우를 생성한다.
3. **InitD3D() 함수에서 Direct3D를 초기화 한다.**
4. **ShowWindow(), UpdateWindow()**로 윈도우를 화면에 표시 한다.
5. **GetMessage(), TranslateMessage(), DispatchMessage()**로 이루어진 메시지 루프를 수행 한다.
6. **메시지 루프를 빠져나올 경우 초기화한 Direct3D를 메모리에서 해제한다.**
7. 프로그램을 종료한다.

2. D3DX INIT

InitD3D

```
HRESULT InitD3D(HWND hWnd)
{
	// 디바이스를 생성하기 위한 D3D 객체 생성
	if (NULL == (g_pD3D = Direct3DCreate9(D3D_SDK_VERSION)))
		return E_FAIL;

	D3DPRESENT_PARAMETERS d3dpp;   // 디바이스 생성을 위한 구조체
	ZeroMemory(&d3dpp, sizeof(d3dpp)); // 반드시 ZeroMemory() 함수로 미리 구조체를 깨끗이 지워야 한다.

	d3dpp.Windowed = TRUE;    // 창 모드로 생성
	d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;    // 가장 효율적인 SWAP 효과
	d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;     // 현재 바탕화면 모드에 맞춰서 후면 버퍼를 생성.

	// 디바이스를 설정해서 생성
	// 디폴트 비디오카드를 사용하고, HAL 디바이스를 생성한다.
	// 정점 처리는 모든 카드에서 지원하는 SW처리로 생성한다.
	if (FAILED(g_pD3D->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
	D3DCREATE_SOFTWARE_VERTEXPROCESSING, &d3dpp, &g_pd3dDevice)))
	{
		return E_FAIL;
	}

	// 디바이스 상태 정보를 처리할 경우 여기에서 한다.

	return S_OK;
}
```

2. D3DX INIT

HRESULT CreateDevice(UINT Adater, D3DDEVTYPE DeviceType, HWND hFocusWindow, DWORD BehaviorFlags, D3DPRESENT_PARAMETERS* pPresentationParameters, IDirect3DDevice9* ppReturnedDeviceInterface);

Adapter
디바이스를 생성할 화면의 순서 번호이다. **D3DADAPTER_DEFAULT**는 기본 화면을 나타낸다. 모니터가 두 대 이상인 경우에만 해당된다. 모니터가 한대라면 신경 쓰지 말자.

DeviceType
출력 디바이스의 종류를 결정한다. **D3DDEVTYPE_HAL, D3DDEVTYPE_SW, D3DDEVTYPE_REF**가 있는데, 이 중 **HAL**이 하드웨어 가속을 지원하는 디바이스다.

hForcusWindow
디바이스가 출력할 윈도우의 핸들. 전체 화면 모드일 경우 최상위 윈도우(부모가 없는 윈도우를 말함)

BehaviorFlags
D3DCREATE_HARDWARE_VERTEXPROCESSING과 **D3DCREATE_SOFTWARE_VERTEXPROCESSING**가 중요한 옵션이다. 정점 셰이더를 지원할 때 하드웨어 가속을 사용할 것인지, 소프트웨어적으로 지원할 것인지를 결정하는 것이다. **D3DCREATE_MIXED_VERTEX_PROCESSING**을 사용할 경우에는 하드웨어 가속과 소프트웨어 가속을 실행 중에 변활 할 수 있다.

pPresentationParameters
D3DPresent_PARAMETERS의 포인터

ppReturnedDeviceInterface
우리가 얻고자 했던 바로 그 값이다. **IDirect3DDevice9**의 인터페이스를 갖고 있는 포인터가 담겨서 돌아온다.

2. D3DX INIT

Render – 화면그리기

```
void Render()
{
	if (NULL == g_pd3dDevice)
		return;

	// 후면 버퍼를 파란색(0, 0, 255)으로 지운다.
	g_pd3dDevice->Clear(0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0, 0, 255), 1.0f, 0);

	// 랜더링 시작
	if (SUCCEEDED(g_pd3dDevice->BeginScene()))
	{
		//실제 렌더링 명령들이 나열될 곳

		//렌더링 종료
		g_pd3dDevice->EndScene();
	}


	// 후면 버퍼를 보이는 화면으로 전환.
	g_pd3dDevice->Present(NULL, NULL, NULL, NULL);
}
```

1. 화면을 깨끗이 지운다(**Clear** – 후면 버퍼)
2. 이제부터 폴리곤을 그리겠다고 **D3D**에게 알린다.(**BeginScene** - 후면버퍼)
3. 실제 랜더링 명령들이 나열될 곳
4. 폴리곤을 다 그렸다고 **D3D**에게 알린다(**EndScene** - 후면버퍼)
5. 화면에 나타나게 한다.(**Present** – 전면버퍼로 전환)

WinAPI로 그릴 때 화면이 깜빡 거리는 걸 **DubleBuffer**로 해결했다.
D3DX에서는 이 문제를 후면버퍼에 그리는 행위를 하고 전면버퍼로 플립하는 **페이지 플리핑 방식**으로 해결하였다.