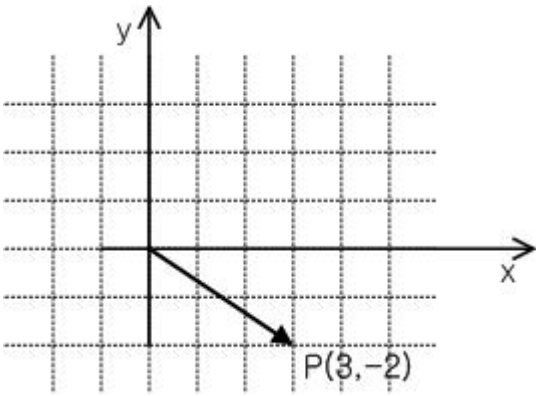
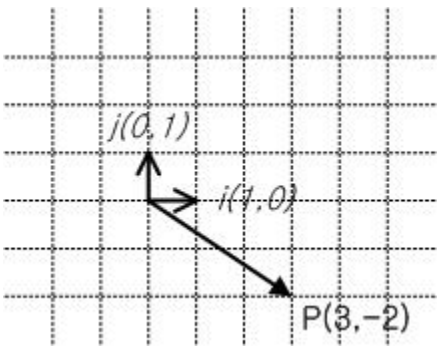
Basis, Linear Combination

1 베이시스(Basis)

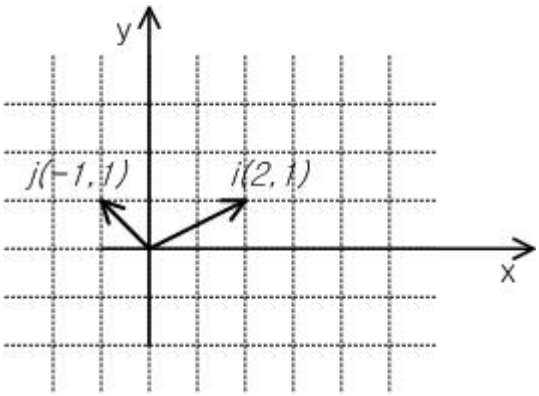


[그림] 벡터 P(3,-2)

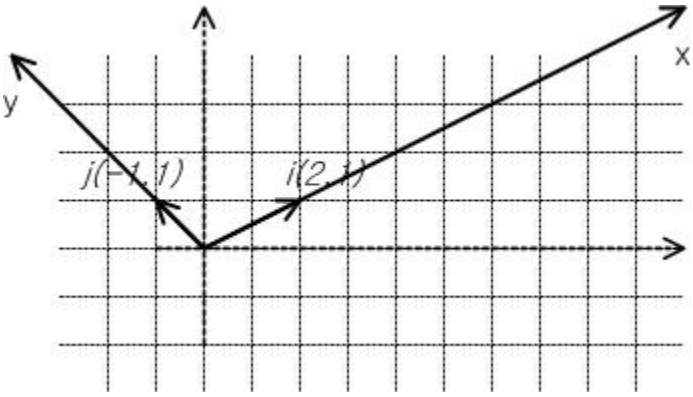


[그림] 베이시스 벡터 i(1,0)와 j(0,1)

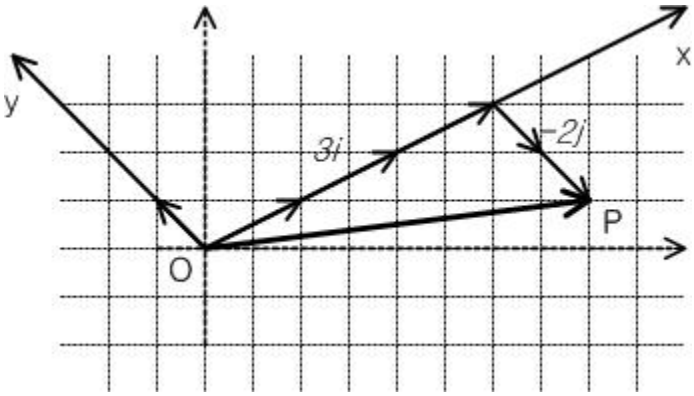
$$3i - 2j = 3\begin{bmatrix}1\\0\end{bmatrix} - 2\begin{bmatrix}0\\1\end{bmatrix}$$



[그림] 베이시스 벡터 i(2,1)와 j(-1,1)



[그림] 베이시스 벡터 i(2,1)와 j(-1,0)가 이루는 좌표계



［그림］ 새로운 축에서의 (3,-2)의 의미

$$3i - 2j = 3\begin{bmatrix}2\\1\end{bmatrix} - 2\begin{bmatrix}-1\\1\end{bmatrix} = \begin{bmatrix}8\\1\end{bmatrix}$$

2 선형 조합(Linear Combination)

벡터 w가 다음과 같이 표현된다고 합시다.

$$w = k_1 v_1 + k_2 v_2 + \ldots + k_r v_r$$

이 경우(단, k_1, k_2, ... ,k_r은 스칼라 값), w를 v_1, v_2, ... , v_r의 **선형 조합(linear combination)**이라고 합니다. 선형 조합의 의미는 w가 다른 벡터의 조합에 의해 표현될 수 있다는 의미입니다. **벡터 공간(vector space)** - 벡터로 표현되는 모든 점들의 집합, 2차원에서 벡터 공간은 평면을 의미하고, 3차원에서 벡터 공간은 3차원 공간 자체를 의미합니다 - 에서 임의의 벡터 v_1, v_2, ... , v_r에 대해, v_1, v_2, ... , v_r의 선형 조합이 모든 벡터 공간상의 벡터를 표현할 수 있다면, 이 벡터들이 벡터 공간을 **스팬(span)**한다고 합니다. 2차원 평면의 예를 들면, (0, 1), (1, 0)벡터의 선형 조합 k_1(0,1) + k_2(1,0)는 임의의 k값들을 대입함에 따라 모든 평면상의 점들을 표현할 수 있으므로, 2차원 평면 벡터 공간을 스팬합니다. 우리는 2차원 평면을 스팬하는 무수히 많은 벡터들을 선택할 수 있습니다. 하지만, k_1(1,0) + k2(-1,0)은 어떤 k값을 대입하더라도 평면상의 모든 점들을 표현할 수 없으므로 2차원 벡터 공간을 스팬하지 않습니다. 짐작하듯이 우리가 2차원 평면상에서 점의 위치를 표현하기 위해 좌표축을 고를 때에는 2차원 평면을 스팬하는 좌표축

벡터를 선택해야 합니다. 다시 정리하면 우리는 2차원 평면을 위해 (1,0), (0,2)축을 선택하는 것은 타당하지만, (1,0), (-1,0) 축을 선택하는 것은 타당하지 않습니다.

S = {v_1, v_2, ... , v_r}가 벡터들의 집합일때, 아래의 벡터 방정식을 고려해 봅시다.

$$k_1v_1 + k_2v_2 + \ldots + k_rv_r = 0$$

위 식은 최소한 $k_1 = 0$, $k_2 = 0$, ..., $k_r = 0$의 해를 가집니다. 이 해의 의미는 벡터의 종류에 상관없이 모든 k가 0이라면 방정식이 성립한다는 의미입니다. 만약 이 해가 유일한 해라면 S를 **선형 독립(linearly independent)** 집합이라고 합니다. 만약 다른 해가 존재한다면 **선형 종속(linearly dependent)** 집합이라고 합니다. 선형 독립의 의미는 S에 속한 임의의 벡터가 다른 벡터의 선형 조합에 의해 표현 불가능함을 의미합니다. 예를 들면, S = { (0,1), (1,0), (0,2) }을 고려해 봅시다.

$$-2(0,1) + 0(1,0) + 1(0,2) = 0$$

위 식은 k1 = 0, k2 = 0, k3 = 0 이외에도 k1 = -2, k2 = 0, k3 = 1의 해를 가지므로, 선형 종속입니다.

$$(0,2) = 2(0,1) + 0(1,0)$$

위 식에서 보듯이 (0,2) 벡터는 다른 벡터의 선형 조합(linear combination)으로 표현가능합니다. 즉, 선형 독립인 벡터의 집합은 군더더기 벡터가 없습니다. 2차원 평면의 예를 들면, 우리는 2차원 벡터 공간을 표현하기 위해 적절한 2개의 축만을 사용하면 됩니다. 만약 3개의 축을 사용한다면 필요없는 여분의 축을 사용하는 결과 즉 선형 종속이 되는 것입니다.

이제 차원(dimension)과 차원상의 위치를 표현하기 위해 사용해야 하는 축 벡터(axis vector)를 정의할 준비가 거의 되었습니다.

벡터 공간 V에 대해 S = {v_1, v_2, ... , v_r}가 V의 유한 집합(finite set)이고 아래의 두 조건을 만족하면 S를 **베이시스(basis)**라고 합니다.

1) S는 선형 독립이다.

2) S는 V를 스팬(span)한다.

베이시스의 의미는 벡터 공간을 모두 표현할 수 있는 최소한의 벡터 집합이라는 의미입니다. 우리는 어떤한 벡터 공간에 대해서도 쉽게 한 베이시스를 다음과 같이 선택할 수 있습니다.

$$v_1 = (1,0, \ldots, 0),\ v_2 = (0,1,0, \ldots, 0),\ \ldots,\ v_r = (0, \ldots, 0,1)$$

우리는 이러한 베이시스를 **표준 베이시스(standard basis)**라고 합니다. 그리고 베이시스의 벡터의 갯수를 **차원(dimension)**이라고 합니다.

$$S = \{(1,0,0),\ (0,1,0),\ (0,0,1)\}$$

위 벡터 집합은 3차원에 대한 표준 베이시스입니다.

벡터 집합의 모든 벡터 쌍이 서로 직각(orthogonal)이라면, 우리는 그 집합을 **올소고널 집합(orthogonal set)**이라고 합니다. 올소고널 집합의 모든 벡터의 길이가 1이라면 우리는 그 집합을 **올소노멀(orthonormal)**이라고 합니다. 예를 들면 아래의 벡터들은 orthonormal입니다.

$$v_1 = (0,\ 1,\ 0),\ v_2 = (\frac{1}{\sqrt{2}},\ 0,\ \frac{1}{\sqrt{2}}),\ v_3 = (\frac{1}{\sqrt{2}},\ 0,\ -\frac{1}{\sqrt{2}})$$

우리는 모든 차원에 대해 베이시스를 선택할 때 여러가지 이점 때문에(실습문제 8) **올소노멀 베이시스(orthonormal basis)**를 선택합니다. 표준 베이시스(standard basis)는 항상 올소노멀입니다. 그래서 우리는 2차원 평면과 3차원 공간에 대해서 표준 올소노멀 베이시스를 선택합니다. 3차원 공간의 경우 그것은 다음과 같습니다.

$$(1,0,0),\ (0,1,0),\ (0,0,1)$$

그리고 각각을 x축, y축, z축 단위 벡터(unit vector)라 부릅니다. 우리는 x축, y축, z축 단위 벡터의 선형 조합식을 아래와 같이 구성할 수 있습니다.

$$k_1(1,0,0) + k_2(0,1,0) + k_3(0,0,1)$$

이 식에 의해서 3차원 공간상의 모든 점을 표현할 수 있습니다. 그래서 우리는 3차원 물체를 나타내기 위해서 서로 직각인 길이가 1인 세 개의 축을 사용합니다.

3 베이시스 클래스의 구현

```
class KBasis2
{
public:
    KVector2 basis0;
    KVector2 basis1;

public:
    KBasis2() { basis0 = KVector2(1, 0); basis1 = KVector2(0, 1); }
    void SetInfo(const KVector2& tbasis0, const KVector2& tbasis1 )
    {
        basis0 = tbasis0;
        basis1 = tbasis1;
    }
    KVector2 Transform(const KVector2& input)
    {
        KVector2 t0 = input.x*basis0;
        KVector2 t1 = input.y*basis1;
        KVector2 temp(t0.x + t1.x,  t0.y + t1.y);
        return temp;
    }
};
```

```
class KScreenCoordinate
{
public:
    KVector2 axis0;
    KVector2 axis1;
    KVector2 origin;

public:
    KScreenCoordinate() { axis0 = KVector2(1, 0); axis1 = KVector2(0,
-1); origin = KVector2(0, 0); }
```

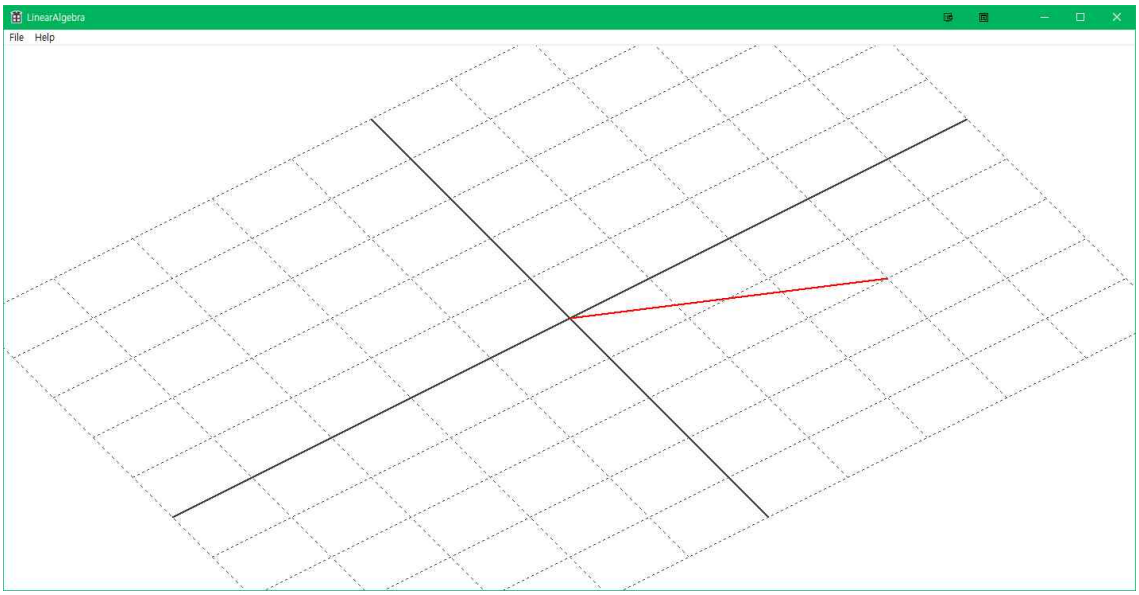
```
    void SetInfo(const KVector2& taxis0, const KVector2& taxis1, const
KVector2& torigin)
    {
        axis0 = taxis0;
        axis1 = taxis1;
        origin = torigin;
    }
    void SetOrigin(const KVector2& origin_) { origin = origin_;  }
    KVector2 Transform(const KVector2& input)
    {
        KVector2 t0 = input.x*axis0;
        KVector2 t1 = input.y*axis1;
        KVector2 temp(t0.x + t1.x + origin.x, t0.y + t1.y + origin.y);
        return temp;
    }
};
```

```
void OnPaint(HDC hdc)
{
    RECT rect;
    GetClientRect(g_hwnd, &rect);
    KVector2 origin;
    origin.x = rect.left + (rect.right - rect.left) / 2.0f;
    origin.y = rect.top + (rect.bottom - rect.top) / 2.0f;
    KVectorUtil::g_screenCoordinate.SetInfo(KVector2(                0),
KVector2(0, -50), origin);

    KBasis2     basis2;
    basis2.SetInfo(KVector2(2, 1), KVector2(-1, 1) );
    KVectorUtil::SetBasis2( basis2 );

    KVectorUtil::DrawGrid(hdc, 10, 10 );
    KVectorUtil::DrawAxis(hdc, 10, 10 );
    KVectorUtil::DrawLine(hdc,  KVector2(0,  0),  KVector2(3,  -2),  2,
PS_SOLID, RGB(255,0,0) );

}
```

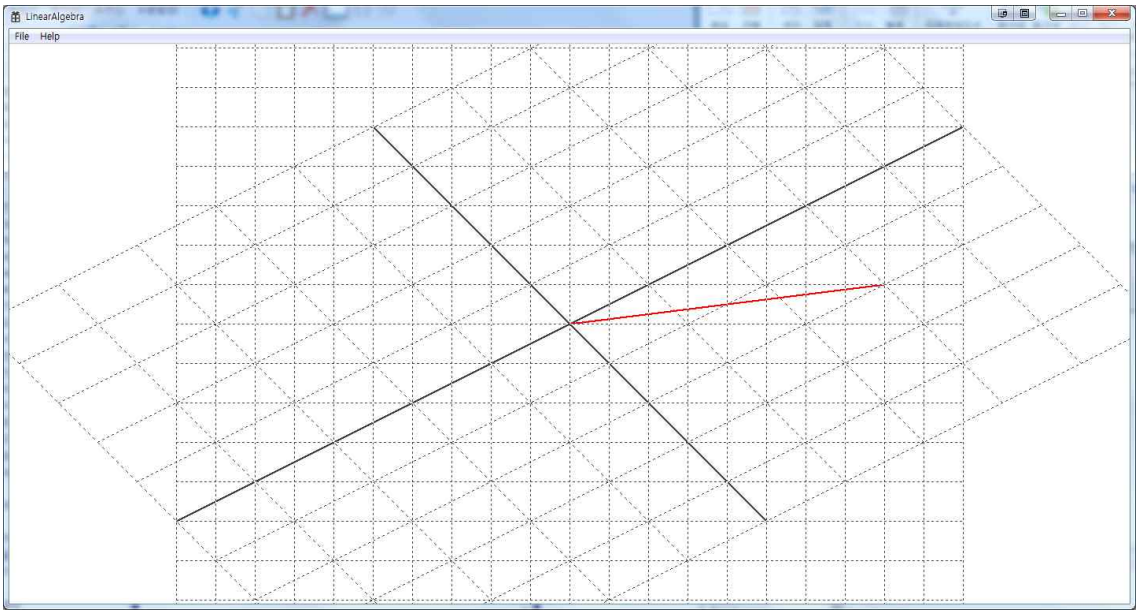


[그림] i(2,1)와 j(-1,1)를 베이시스로 가지는 축에서의 벡터 (3,-2)의 의미

$$3i - 2j = 3\begin{bmatrix}2\\1\end{bmatrix} - 2\begin{bmatrix}-1\\1\end{bmatrix} = \begin{bmatrix}8\\1\end{bmatrix}$$

```
    basis2.SetInfo(KVector2(1, 0), KVector2(0, 1) );
    KVectorUtil::SetBasis2( basis2 );

    KVectorUtil::DrawGrid(hdc, 20, 20 );
```



[그림] $3i-2j=3\begin{bmatrix}2\\1\end{bmatrix}-2\begin{bmatrix}-1\\1\end{bmatrix}$은 원래 좌표계의 $\begin{bmatrix}8\\1\end{bmatrix}$을 나타냅니다.

4 보다 간단한 표기법

$$3i-2j=3\begin{bmatrix}2\\1\end{bmatrix}-2\begin{bmatrix}-1\\1\end{bmatrix}=\begin{bmatrix}8\\1\end{bmatrix}$$

$$\begin{bmatrix}i & j\end{bmatrix}\begin{bmatrix}3\\-2\end{bmatrix}=i3+j(-2)$$

$$\begin{bmatrix}2 & -1\\1 & 1\end{bmatrix}\begin{bmatrix}3\\-2\end{bmatrix}=\begin{bmatrix}2\\1\end{bmatrix}3+\begin{bmatrix}-1\\1\end{bmatrix}(-2)=\begin{bmatrix}2\times3+(-1)\times(-2)\\1\times3+1\times(-2)\end{bmatrix}=\begin{bmatrix}8\\1\end{bmatrix}$$

5 애니메이션을 위한 준비

```
// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
```

```
WCHAR szWindowClass[MAX_LOADSTRING];                // the main window
class name
// _20180519_jintaeks
HWND    g_hwnd = NULL;
HDC     g_hdc = 0;
HBITMAP g_hBitmap = 0;
RECT    g_clientRect;
```

```
void                Initialize();
void                Finalize();
void                OnSize();
void                OnIdle(float fElapsedTime_);
```

```
    Initialize();

    DWORD dwOldTime = ::timeGetTime();

    MSG msg;

    // Main message loop:
    while (true)
    {
        ::PeekMessage(&msg, NULL, 0, 0, PM_REMOVE);
        const DWORD dwNewTime = ::timeGetTime();
        const BOOL bIsTranslateMessage = TranslateAccelerator(msg.hwnd,
hAccelTable, &msg);
        if (!bIsTranslateMessage)
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }//if

        OnIdle(float(dwNewTime - dwOldTime) / 1000.f);
        Sleep(10);

        dwOldTime = dwNewTime;
```

```
        if (msg.message == WM_QUIT)
        {
            break;
        }//if
    }//while

    Finalize();
```

```
void OnPaint(HDC hdc)
{
}
```

```
    case WM_SIZE:
        OnSize();
        break;
```

```
void Initialize()
{
}//Initialize()

void Finalize()
{
    if (g_hdc != 0) {
        DeleteDC(g_hdc);
        g_hdc = 0;
    }//if
    if (g_hBitmap != 0) {
        DeleteObject(g_hBitmap);
        g_hBitmap = 0;
    }//if
}//Finalize()
```

```
void OnSize()
```

```
{
    Finalize();

    ::GetClientRect(g_hwnd, &g_clientRect);
    const int iWidth = g_clientRect.right - g_clientRect.left + 1;
    const int iHeight = g_clientRect.bottom - g_clientRect.top + 1;

    KVector2 origin;
    origin.x = iWidth / 2.0f;
    origin.y = iHeight / 2.0f;
    KVectorUtil::g_screenCoordinate.SetInfo(KVector2(!              0),
KVector2(0, -50), origin);

    HDC hdc = ::GetDC(g_hwnd);
    g_hdc = CreateCompatibleDC(hdc);
    g_hBitmap = CreateCompatibleBitmap(hdc, iWidth, iHeight);
    SelectObject(g_hdc, g_hBitmap);
}//OnSize()
```

```
void OnIdle(float fElapsedTime_)
{
    const int iWidth = g_clientRect.right - g_clientRect.left + 1;
    const int iHeight = g_clientRect.bottom - g_clientRect.top + 1;

    HDC hdc = ::GetDC(g_hwnd);

    HBRUSH brush;
    brush = CreateSolidBrush(RGB(255, 255, 255));
    SelectObject(g_hdc, brush);
    Rectangle(g_hdc, 0, 0, iWidth, iHeight);

    {
        KBasis2     basis2;
        basis2.SetInfo(KVector2(2, 1), KVector2(-1, 1));
        KVectorUtil::SetBasis2(basis2);

        KVectorUtil::DrawGrid(g_hdc, 10, 10);
        KVectorUtil::DrawAxis(g_hdc, 10, 10);
        KVectorUtil::DrawLine(g_hdc,  KVector2(0,  0),  KVector2(3,  -2),
2, PS_SOLID, RGB(255, 0, 0));
```

```
    }

    BitBlt(hdc, 0, 0, iWidth, iHeight, g_hdc, 0, 0, SRCCOPY);
    DeleteObject(brush);

    ::ReleaseDC(g_hwnd, hdc);
}//OnIdle()
```

@