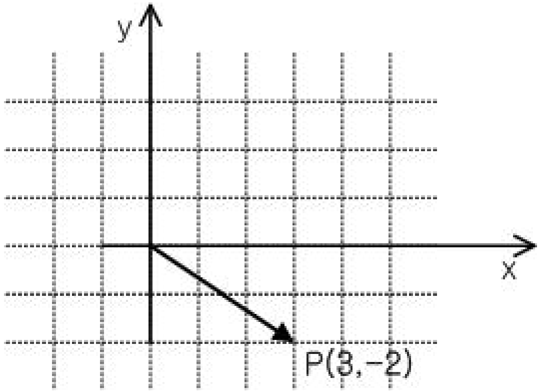
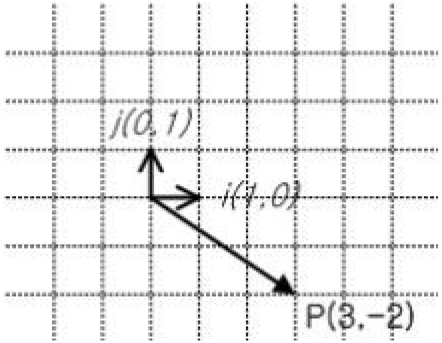
Basis, Linear Combination

Basis

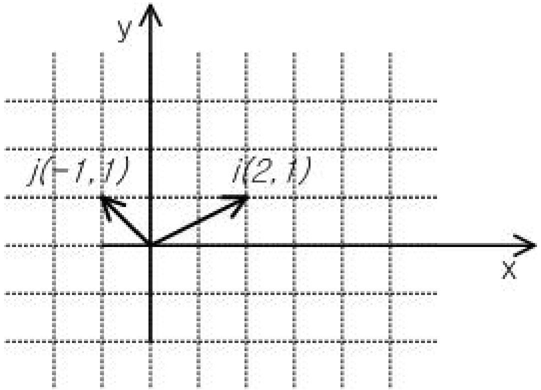


[Fig] Vector P(3,-2)

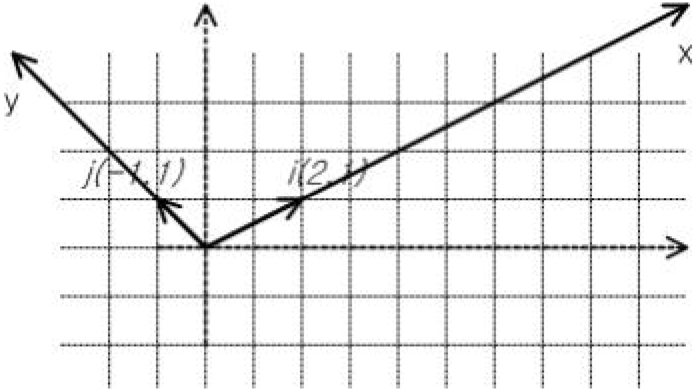


[Fig] Basis Vectors i(1,0) and j(0,1)

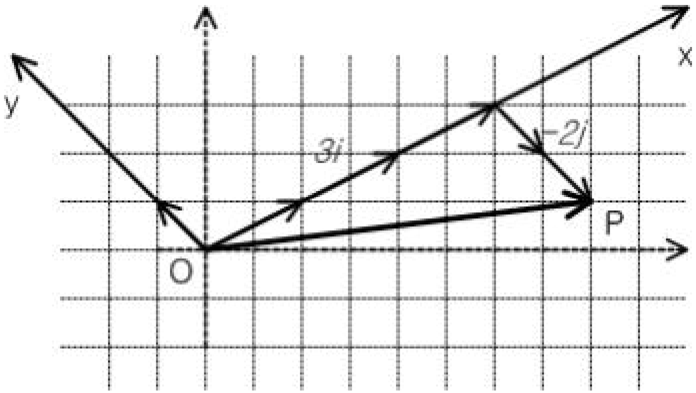
$$3i - 2j = 3\begin{bmatrix}1\\0\end{bmatrix} - 2\begin{bmatrix}0\\1\end{bmatrix}$$



[Fig] Basis Vectors i(2,1) and j(-1,1)



[Fig] Coordinate system by Basis Vector i(2,1) and j(-1,0)



[Fig] Meaning of (3,-2) in new Coordinate System

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

Linear Combination

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

such that k_1, k_2, … ,k_r are scalar values

w is a **linear combination** of v_1, v_2, … , v_r

vector space

span

For S = {v_1, v_2, … , v_r}

Consider Vector equation

$$k_1v_1 + k_2v_2 + ... + k_rv_r = 0$$

Trivial solution at k_1 = 0, k_2 = 0, …, k_r = 0

linearly independent

linearly dependent

Example)

S = { (0,1), (1,0), (0,2) }

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

k1 = 0, k2 = 0, k3 = 0

k1 = -2, k2 = 0, k3 = 1

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

Basis

for a vector apace V, S = {v_1, v_2, … , v_r} is a finite set of V
S is basis, when it satisfies two conditions below

1) S is linearly independent
2) S spans V.

Standard Basis and Dimension

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

Ex) Standard Basis for 3 Dimension

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

orthogonal set

orthonormal

Ex) Orthonormal

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

Ex) Orthonormal Basis for 3 Dimension

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

Unit Vector for each Axis

x-axis

y-axis

z-axis

Ex) All points in 3-dimensional space

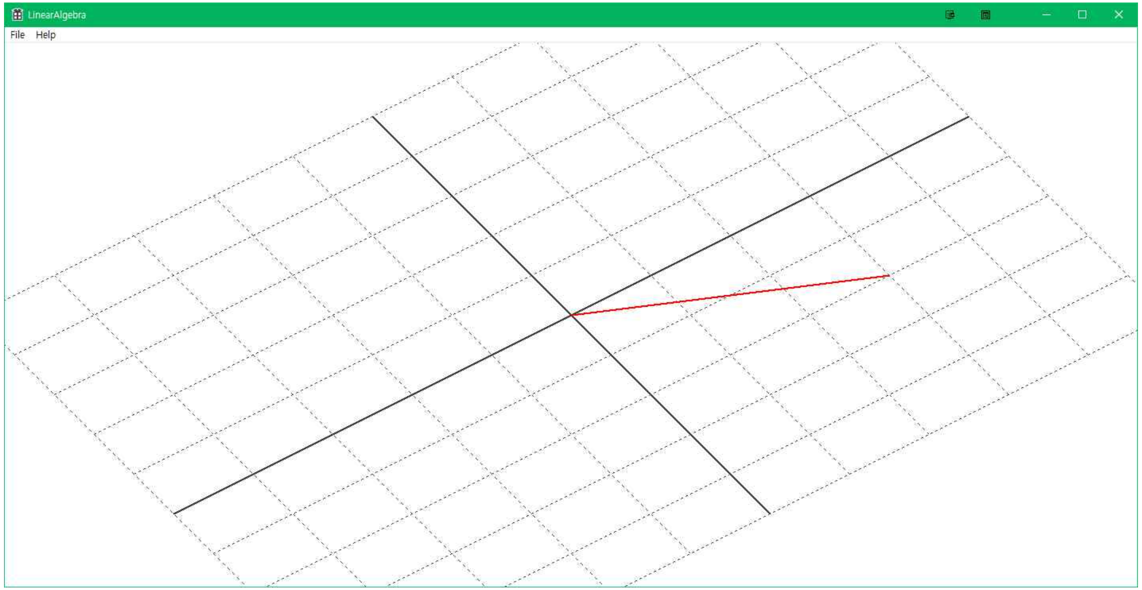
$$k_1(1,0,0)\ +\ k_2(0,1,0)\ +\ k_3(0,0,1)$$

Implementing Basis Class

```
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;
    }
};
```

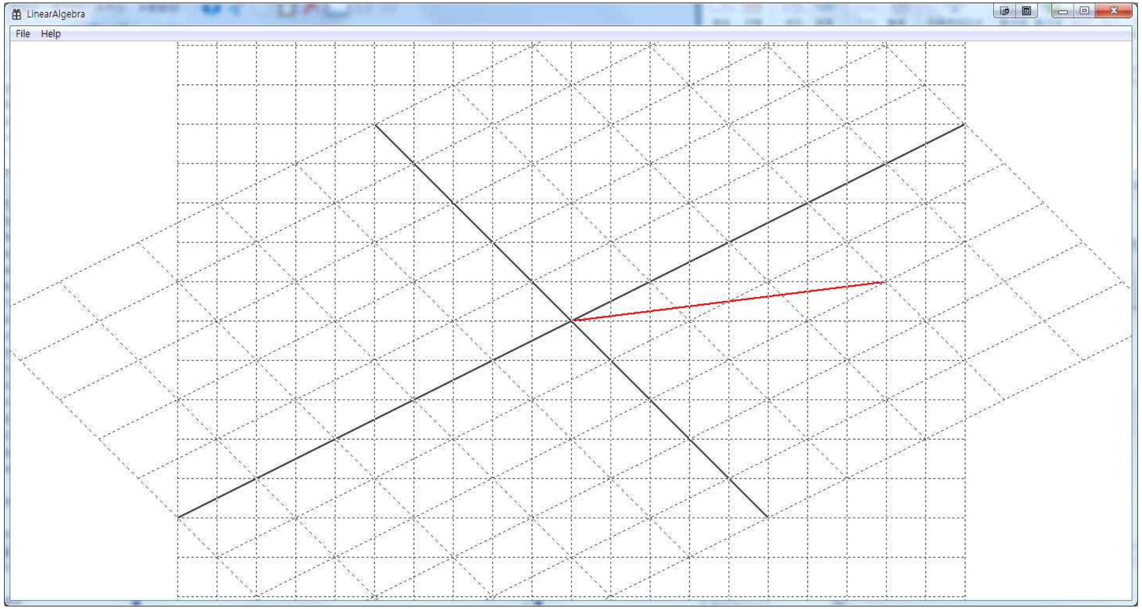
```
void OnPaint(HDC hdc)
{
    RECT rect;
    GetClientRect(g_hwnd, &rect);
    KVector2 origin;
    origin.x = rect.left + (rect.right - rect.left) / 2.0f;
    origin.y = rect.top + (rect.bottom - rect.top) / 2.0f;
    KVectorUtil::g_screenCoordinate.SetInfo(KVector2(!    0), KVector2(0, -50), origin);

    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) );

}
```



[Fig] Vector (3,-2) in basis i(2,1) and j(-1,1)

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

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

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



[Fig] $3i-2j=3\begin{bmatrix}2\\1\end{bmatrix}-2\begin{bmatrix}-1\\1\end{bmatrix}$ is $\begin{bmatrix}8\\1\end{bmatrix}$ in standard basis

Idea: Simpler Notation

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

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

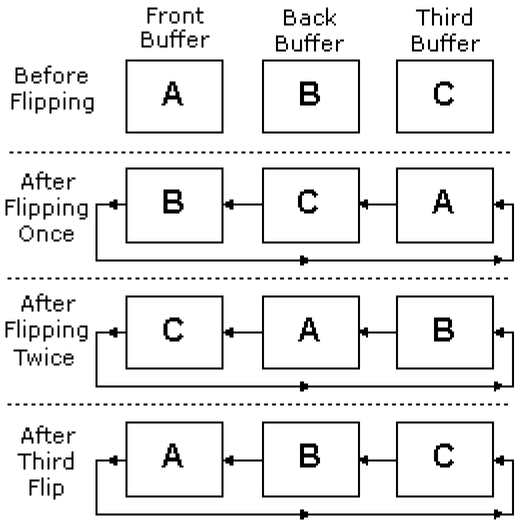
$$\begin{bmatrix}2&-1\\1&1\end{bmatrix}\begin{bmatrix}3\\-2\end{bmatrix}=\begin{bmatrix}2\\1\end{bmatrix}3+\begin{bmatrix}-1\\1\end{bmatrix}(-2)=\begin{bmatrix}2\times3+(-1)\times(-2)\\1\times3+1\times(-2)\end{bmatrix}=\begin{bmatrix}8\\1\end{bmatrix}$$

Preparing for Animation

Flickering in Animation

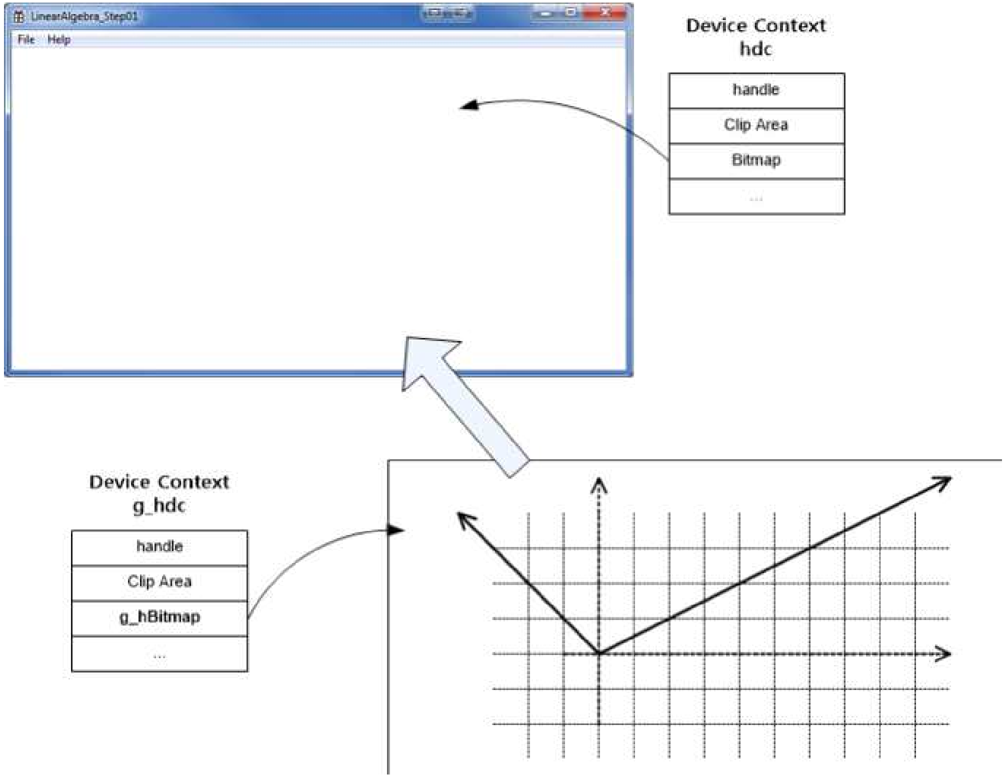
(Shows flickering of LinearAlgebra_Step04 Matrix2 project when g_hdc is not used)

Flipping Surfaces in Direct3D



[Fig] Swap Chain of DirectX: if you call Present(), next buffer in Swap Chain will be selected as back-buffer.

Hidden Buffer in GDI



[Fig] Hidden buffer

Preparing hidden buffer

```
// 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;
```

Adding OnSize() and OnIdle()

OnSize()

OnIdle()

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

GetMessage() vs. PeekMessage()

```
    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();
```

OnPaint()

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

WM_SIZE message

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

Initialize() and Finalize()

```
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()
```

Implementing OnSize()

```
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()
```

Implementing OnIdle()

Front Buffer

Back Buffer

Flickering

```
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()
```

Exercise

1. Define basis that is rotated by theta counterclockwise.
2. Draw a vector (3,-2) on that coordinate system.
3. Write a animation program that rotates (3,-2) once per a second.

@