Matrix

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

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

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

$$\begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}a\\c\end{bmatrix}x+\begin{bmatrix}b\\d\end{bmatrix}y=\begin{bmatrix}ax+by\\cx+dy\end{bmatrix}$$

Simpler New Notation: Matrix

Matrix

Element

$$\begin{pmatrix}1 & 2 & 3 & 4\end{pmatrix}, \begin{pmatrix}1\\2\\3\end{pmatrix}, \begin{pmatrix}\sin(x) & 2 & 3\\ \cos(x) & 4 & 5\end{pmatrix}$$

update class KVector2

Adding static variables(zero, one, right and up vectors)

Adding Linear Interpolation method

```
class KVector2
{
public:
    static KVector2 zero;
```

```
    static KVector2 one;
    static KVector2 right;
    static KVector2 up;

    static KVector2 Lerp(const KVector2& begin, const KVector2& end, float ratio);
```

Lerp() linearly interpolates vector 'begin' and vector 'end'.

If ratio_ equals to 0, Lerp() returns vector 'begin', if ratio_ equals to 1, Lerp() returns vector 'end'.

```
KVector2 KVector2::zero = KVector2(0, 0);
KVector2 KVector2::one = KVector2(1, 1);
KVector2 KVector2::right = KVector2(1, 0);
KVector2 KVector2::up = KVector2(0, 1);

KVector2 KVector2::Lerp(const KVector2& begin, const KVector2& end, float ratio_)
{
    float ratio = __min(1, __max(0, ratio_));
    KVector2 temp;
    temp.x = begin.x + (end.x - begin.x) * ratio;
    temp.y = begin.y + (end.y - begin.y) * ratio;
    return temp;
}
```

class KMatrix2

```
class KMatrix2
{
public:
    static KMatrix2 zero;
    static KMatrix2 identity;
public:
    float   _11, _12;
    float   _21, _22;

public:
    KMatrix2(float e11 = 1.0f, float e12 = 0.0f, float e21 = 0.0f, float e22 = 1.0f)
    {
        _11 = e11;
        _12 = e12;
        _21 = e21;
        _22 = e22;
    }
    ~KMatrix2() {}
    void Set(float e11, float e12, float e21, float e22)
    {
        _11 = e11;
        _12 = e12;
```

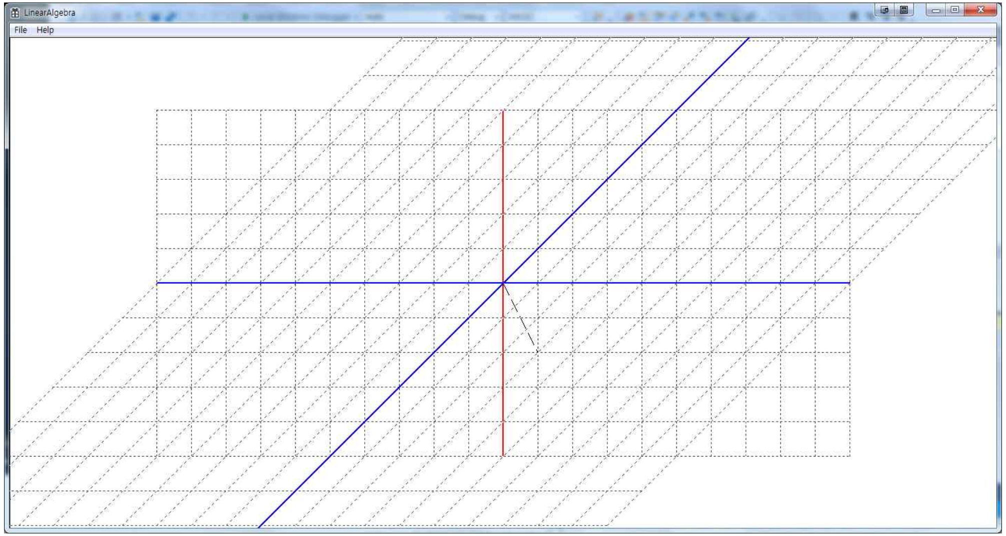
```
        _21 = e21;
        _22 = e22;
    }
};

inline KVector2 operator*(const KMatrix2& m, const KVector2& v)
{
    KVector2 temp;
    temp.x = m._11*v.x + m._12*v.y;
    temp.y = m._21*v.x + m._22*v.y;
    return temp;
}

inline KMatrix2 operator*(float scalar, const KMatrix2& m)
{
    KMatrix2 temp;
    temp._11 = scalar*m._11;
    temp._12 = scalar*m._12;
    temp._21 = scalar*m._21;
    temp._22 = scalar*m._22;
    return temp;
}
```

Shear Transform

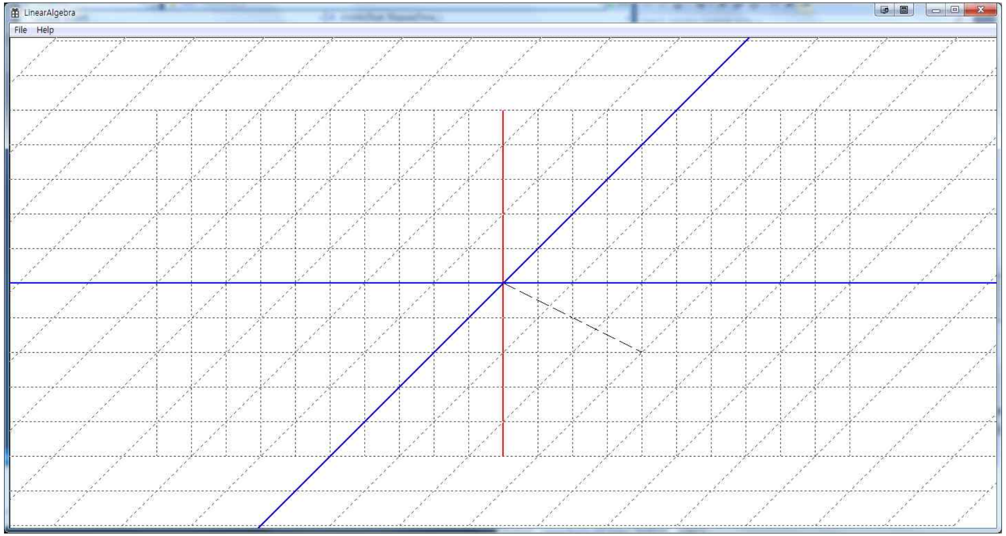
```
        KMatrix2    transform = KMatrix2(
            1, 1,
            0, 1);
```



[Fig] Shear Transform: i(1,0) and j(1,1) vectors are used basis.

Scale Transform

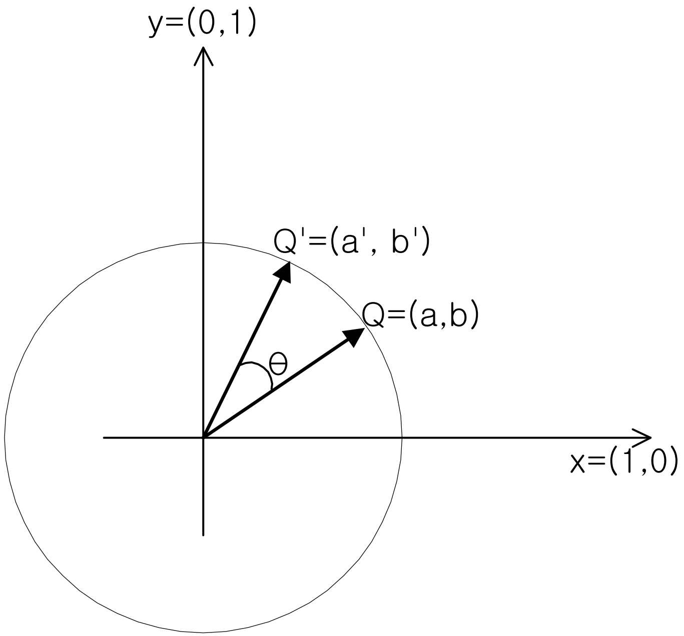
```
        KMatrix2    transform = KMatrix2(
            2, 1,
            0, 1);
```



[Fig] Scale Transform: when i(3,0) is used as first basis, x-values of the vector will be scaled by 3 along x-axis

Rotation Transform

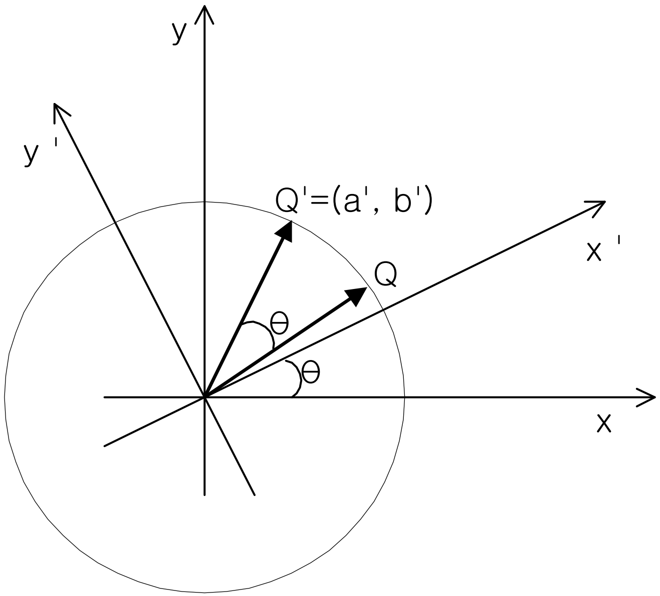
Get new point Q'=(a',b') that is rotated about θ with relative to x-axis of Q=(a,b)



[Fig]. Rotation Transform: It's just another kind of basis transform problem.

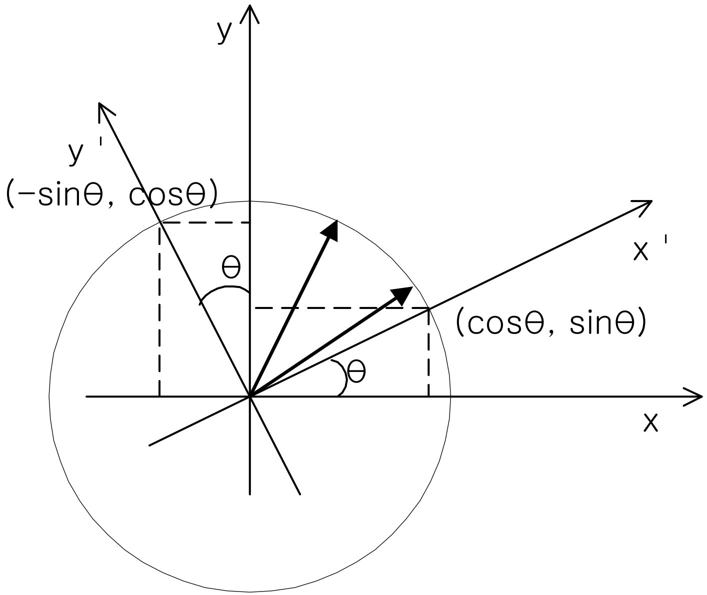
for new basis $x' = (x_1, y_1),\ \ y' = (x_2, y_2)$

$$Q' = \begin{vmatrix} a' \\ b' \end{vmatrix} = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} \begin{vmatrix} a \\ b \end{vmatrix}$$



[Fig]. axis transformation: new axis are rotated about θ.

With help of trigonometrix functions, we can calculate new basis.

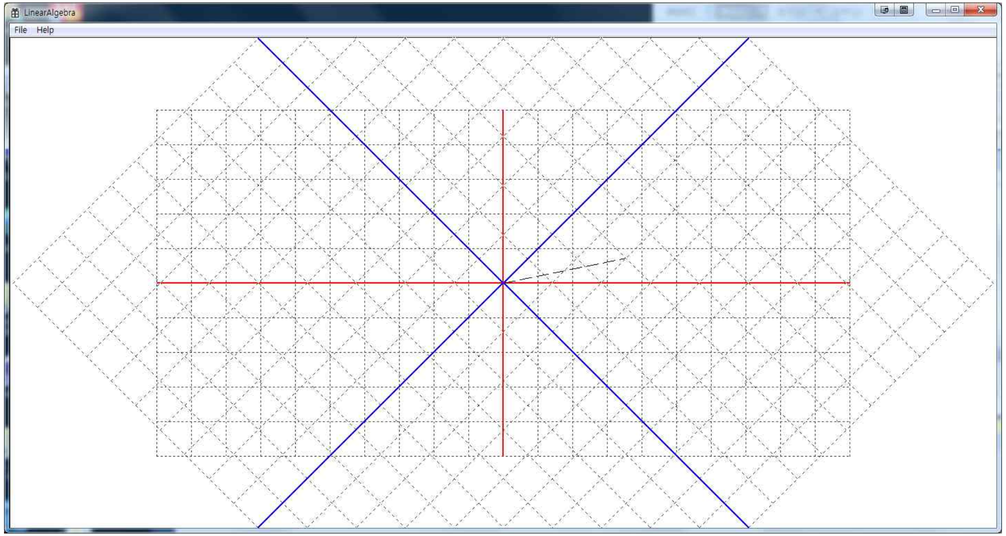


[Fig]. new axis rotated about θ

$$x' = (\cos\theta, \sin\theta),\ \ y' = (-\sin\theta, \cos\theta)$$

$$Q = \begin{vmatrix} a' \\ b' \end{vmatrix} = \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} \begin{vmatrix} a \\ b \end{vmatrix}$$

```
const float costheta = cos( M_PI_4 );
const float sintheta = sin( M_PI_4 );
KMatrix2    transform = KMatrix2(
    costheta, -sintheta,
    sintheta, costheta);
```



[Fig] Rotation Transform

Linear Transform

① A line must be a line after transformation.

② The position of the origin must not be modified.

It preserves the operations of addition and scalar multiplocation.

$$Q' = \begin{vmatrix} a' \\ b' \end{vmatrix} = \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} \begin{vmatrix} a \\ b \end{vmatrix}$$

linear transformation matrix

Transform more than one vectors

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

$$\begin{bmatrix}2 & -1\\1 & 1\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}=\begin{bmatrix}2\\1\end{bmatrix}2+\begin{bmatrix}-1\\1\end{bmatrix}2=\begin{bmatrix}2\times2+(-1)\times2\\1\times2+1\times2\end{bmatrix}=\begin{bmatrix}2\\4\end{bmatrix}$$

$$\begin{bmatrix}2 & -1\\1 & 1\end{bmatrix}\begin{bmatrix}3 & 2\\-2 & 2\end{bmatrix}=\begin{bmatrix}8 & 2\\1 & 4\end{bmatrix}$$

Matrix

Matrix is a rectangular array of numbers

element, entry

$$(1\ 2\ 3\ 4),\ \begin{pmatrix}1\\2\\3\end{pmatrix},\ \begin{pmatrix}\sin(x) & 2 & 3\\ \cos(x) & 4 & 5\end{pmatrix}$$

Row matrix

Column matrix

m by n matrix

Matrix Addition

$$A = \begin{vmatrix} 2 & 1 & 0 & 3 \\ -1 & 0 & 2 & 4 \\ 4 & -2 & 7 & 0 \end{vmatrix}, \quad B = \begin{vmatrix} -4 & 3 & 5 & 1 \\ 2 & 2 & 0 & -1 \\ 3 & 2 & -4 & 5 \end{vmatrix}, \quad C = \begin{vmatrix} 1 & 1 \\ 2 & 2 \end{vmatrix}$$

$$A + B = \begin{vmatrix} -2 & 4 & 5 & 4 \\ 1 & 2 & 2 & 3 \\ 7 & 0 & 3 & 5 \end{vmatrix}$$

A+C? B+C?

Matrix Multiplication

product of AB

$$A = \begin{vmatrix} 1 & 2 & 4 \\ 2 & 6 & 0 \end{vmatrix}, \quad B = \begin{vmatrix} 4 & 1 & 4 & 3 \\ 0 & -1 & 3 & 1 \\ 2 & 7 & 5 & 2 \end{vmatrix}$$

element (2,3) of AB will be 2*4 + 6*3 + 0*5 = 26

$$\begin{vmatrix} 1 & 2 & 4 \\ 2 & 6 & 0 \end{vmatrix} \begin{vmatrix} 4 & 1 & 4 & 3 \\ 0 & -1 & 3 & 1 \\ 2 & 7 & 5 & 2 \end{vmatrix} = \begin{vmatrix} \square & \square & \square & \square \\ \square & \square & 26 & \square \end{vmatrix}$$

Fig. Matrix multiplication AB

$$AB = \begin{vmatrix} 12 & 27 & 30 & 13 \\ 8 & -4 & 26 & 12 \end{vmatrix}$$

Commutative law will not be satisfied for matrix multiplication.

$$AB \neq BA$$

identity matrix

$$AI = IA = A$$

When $AB = BA = I$, then B is **inverse of A** and denoted as A^{-1}.

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

그림. 4×4 Identity matrix

Determinant

The determinant of a matrix A is denoted det(A) or |A|. Geometrically, it can be viewed as the scaling factor of the linear transformation described by the matrix.
In the case of 2×2 matrix:

$$\det(A) = |A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

For a 3×3 matrix:

$$\begin{aligned} |A| &= \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} \\ &= a\begin{vmatrix} \square & \square & \square \\ \square & e & f \\ \square & h & i \end{vmatrix} - b\begin{vmatrix} \square & \square & \square \\ d & \square & f \\ g & \square & i \end{vmatrix} + c\begin{vmatrix} \square & \square & \square \\ d & e & \square \\ g & h & \square \end{vmatrix} \\ &= a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ &= aei - afh - bdi + bfg + cdh - ceg \end{aligned}$$

(watch video)

Minor and Cofactor

Def: The determinant of some smaller square matrix, cut down from A by removing one or more of its rows or columns.

Minors obtained by removing just one row and one column from square matrices (first minors) are required for calculating matrix **cofactors.**

$$\begin{vmatrix} 1 & 4 & 7 \\ 3 & 0 & 5 \\ -1 & 9 & 11 \end{vmatrix}$$

$$M_{2,3} = \begin{vmatrix} 1 & 4 & \square \\ \square & \square & \square \\ -1 & 9 & \square \end{vmatrix} = \det \begin{vmatrix} 1 & 4 \\ -1 & 9 \end{vmatrix} = 9-(-4)) = 13$$

$$C = ((-1)^{i+j} M_{ij})_{1 \leqq i,j \leqq n}$$

$$C_{2,3} = (-1)^{2+3}(M_{2,3}) = -13$$

Adjugate Matrix

The transpose of its cofactor matrix

$$adj(A) = C^T$$

$$C = ((-1)^{i+j} M_{ij})_{1 \leqq i,j \leqq n}$$

$$adj(A) = C^T = ((-1)^{i+j} M_{ji})_{1 \leqq i,j \leqq n}$$

Invertible Matrix

For 2×2 matrix:

$$A^{-1} = \frac{1}{\det(A)} adj(A)$$

$$\det(A) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

$$C = \begin{vmatrix} d & -c \\ -b & a \end{vmatrix}$$

$$adj(A) = C^T = \begin{vmatrix} d & -b \\ -c & a \end{vmatrix}$$

$$A^{-1} = \frac{1}{ad-bc} \begin{vmatrix} d & -b \\ -c & a \end{vmatrix}$$

Exercise: Pentagon Defence Game

class KMatrix2

```
    float GetDeterminant() const
    {
        return _11 * _22 - _12 * _21;
    }
    KMatrix2 GetInverse() const;
```

KMatrix2.cpp

```
KMatrix2 KMatrix2::GetInverse() const
{
    KMatrix2 t;
    t.Set(_22, -_12, -_21, _11);
    float det = GetDeterminant();
    t = (1.0f / det) * t;
    return t;
```

```
}
```

class KVector2

```
inline KVector2 operator-(const KVector2& lhs, const KVector2& rhs)
{
    KVector2 temp(lhs.x - rhs.x, lhs.y - rhs.y);
    return temp;
}
```

namespace KVectorUtil

```
    KVector2 ScreenToWorld(const KVector2& v0_);
    void DrawCircle(HDC hdc, const KVector2& center, float radius, int numSegment
        , int lineWidth = 1, int penStyle = PS_SOLID, COLORREF color = RGB(0, 0, 0));
```

KVectorUtil.cpp

```
KVector2 KVectorUtil::ScreenToWorld(const KVector2& v0_)
{
    KMatrix2 m0;
    m0.Set( g_basis2.basis0, g_basis2.basis1 );
```

```
    KMatrix2 m1;
    m1.Set( g_screenCoordinate.axis0, g_screenCoordinate.axis1 );

    // Vscreen = Mscreen * Mworld * Vworld
    // MworldInv * MscreenInv * Vscreen = Vworld
    KVector2 v = v0_ - g_screenCoordinate.origin; // inverse translation
    KMatrix2 m1Inv = m1.GetInverse();
    KMatrix2 m0Inv = m0.GetInverse();
    v = m0Inv * m1Inv * v;
    return v;
}

void KVectorUtil::DrawCircle( HDC hdc, const KVector2& center, float radius, int numSegment
    , int lineWidth, int penStyle, COLORREF color)
{
    const double dt = (2.0 * M_PI) / numSegment;
    double theta = 0;
    const KVector2 p0 = KVector2(radius, 0.0f);
    KVector2 p1 = p0;
    KVector2 p2;
    for (int i = 0; i <= numSegment; ++i)
    {
        KMatrix2 m;
        theta += dt;
        m.SetRotation((float)theta);
        p2 = m * p0;
```

```
        DrawLine(hdc, p1, p2, lineWidth, penStyle, color);
        p1 = p2;
    }
}
```

LinearAlgebra.cpp

```
#include <windowsx.h>

    case WM_LBUTTONDOWN:
        OnLButtonDown( GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam));
        break;

void OnLButtonDown(int x, int y)
{
}

    {
        KBasis2     basis2;
        basis2.SetInfo(KVector2(1, 0), KVector2(0, 1));
        KVectorUtil::SetBasis2(basis2);
```

```
        KVectorUtil::DrawGrid(g_hdc, 10, 10);
        KVectorUtil::DrawAxis(g_hdc, 10, 10, RGB(255,0,0), RGB(255,0,0));
    }

    KVectorUtil::DrawCircle(g_hdc, KVector2(0, 0), 1, 5);
    POINT mousePoint;
    GetCursorPos(&mousePoint);
    ScreenToClient(g_hwnd, &mousePoint);
    KVector2 vmouse = KVectorUtil::ScreenToWorld( KVector2(mousePoint.x, mousePoint.y) );
    KVector2 vdir = vmouse;
    vdir.Normalize();
    KVectorUtil::DrawLine(g_hdc, KVector2(0, 0), vdir* 1.5f, 2, PS_DASH);
```

@