행렬(Matrix)

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

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

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

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

간단한 표현법: 행렬(Matrix)

숫자들의 직사각형 배열(rectangular array of numbers)을 **매트릭스(matrix)**라 하고, 배열에 있는 각 숫자를 **엘리먼트(element, entry)**라고 합니다. 아래는 매트릭스의 예입니다.

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

KVector2클래스의 수정

기존의 KVector2를 수정하여 몇 가지 정적 변수를 추가하고, 벡터를 선형 보간(Linear Interpolation)하는 Lerp()를 추가합니다. KVector2.h 파일에 영벡터(zero vector), right 벡터, up 벡터를 추가합니다.

```
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()함수는 두 벡터 begin과 end 사이의 선형 보간을 수행합니다. Lerp()의 세번째 파라미터가 0이면 begin()을 리턴하고, 1이면 end()를 리턴합니다.

```
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;
}
```

KMatrix2 클래스

```
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;
```

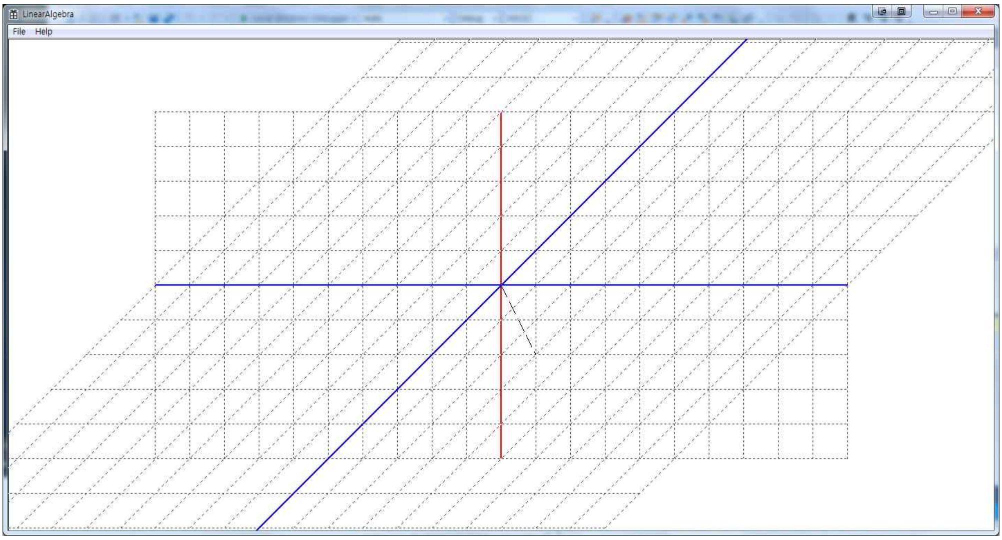
```
        _22 = e22;
    }
    ~KMatrix2() {}
    void Set(float e11, float e12, float e21, float e22)
    {
        _11 = e11;
        _12 = e12;
        _21 = e21;
        _22 = e22;
    }
};

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;
}
```

쉬어 변환(Shear Transform)

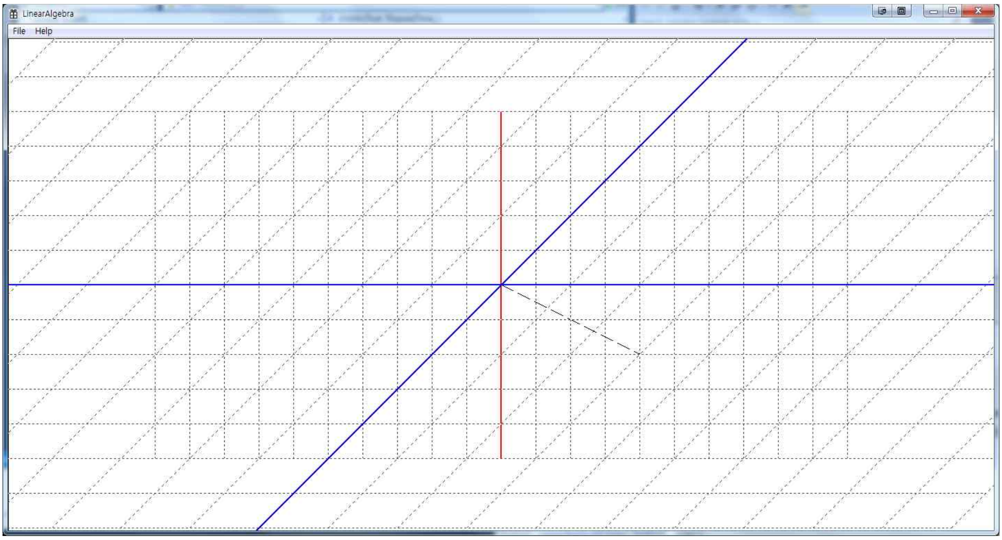
```
        KMatrix2    transform = KMatrix2(
            1, 1,
            0, 1);
```



[그림] 쉬어 변환(Shear Transform): i(1,0)과 j(1,1) 벡터를 베이시스로 사용합니다.

크기 변환(Scale Transform)

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



[그림] 스케일 변환(Scale Transform): i(3,0)을 베이시스로 사용하면, x축을 따라서 크기가 3증가합니다.

회전 변환(Rotation Transform)

2차원 평면에서, 임의의 점 Q=(a,b)를 θ만큼 회전시킨 새로운 점 Q'□=(a',b')을 구하는 문제를 생각해 봅시다.

□ 큐 프라임(prime)이라고 읽습니다.

Q = (a,b) = a(1,0) + b(0,1)은 Q = $|x축\ \ y축| \begin{vmatrix} a \\ b \end{vmatrix}$ 즉, Q = $\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \begin{vmatrix} a \\ b \end{vmatrix}$로 표현되며, 이것은 a,b가 임의의 실수이므로, x축 = (1,0)과 y축 = (0,1)이 표현할 수 있는 모든 2차원 점(point)들을 의미합니다((1,0), (0,1) 벡터가 스팬하는 공간이므로).

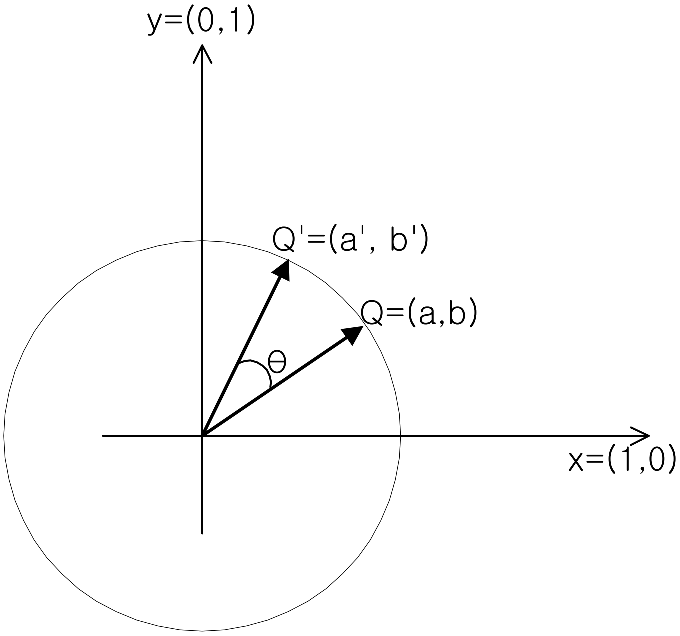


그림. 회전 변환: Q=(a,b)를 θ만큼 회전시킨 새로운 점 Q'=(a',b')을 구하는 문제는 베이시스를 변환하는 문제, 즉 축을 회전시키는 문제로 생각할 수 있습니다.

회전된 새로운 위치 Q'=(a',b')는 x축과 y축이 θ만큼 회전된 - 즉 베이시스가 변환된 - 새로운 점이며 회전 변환된 베이시스를 $x' = (x_1, y_1),\ y' = (x_2, y_2)$라하면 Q' = (a',b')은 다음과 같습니다.

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

이때 새로운 축 x', y'은 올소노멀 베이시스 조건을 만족해야 합니다.

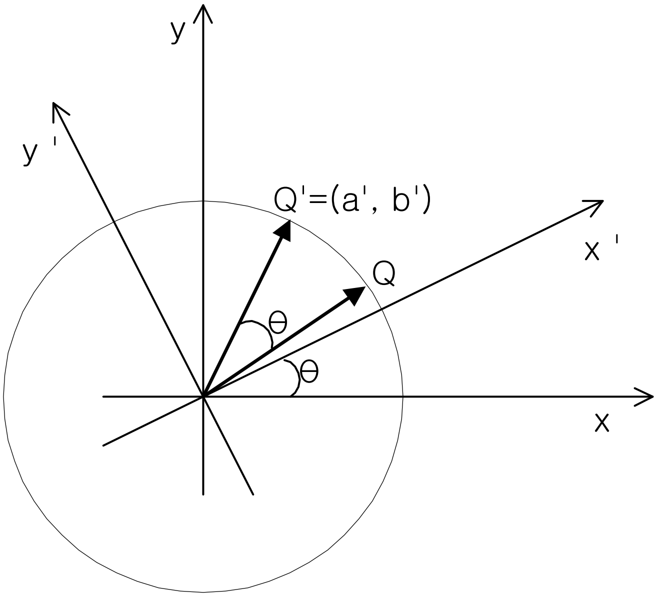


그림. 축변환(axis transformation): 회전 변환을 축을 변화시키는 문제로 생각하면 변환된 새로운 축을 구성하는 올소노멀 베이시스를 구하면 됩니다.

새로운 축 x', y'은 삼각함수(trigonometrix functions)에 의해서 구할 수 있습니다. 아래 그림을 참조하세요.

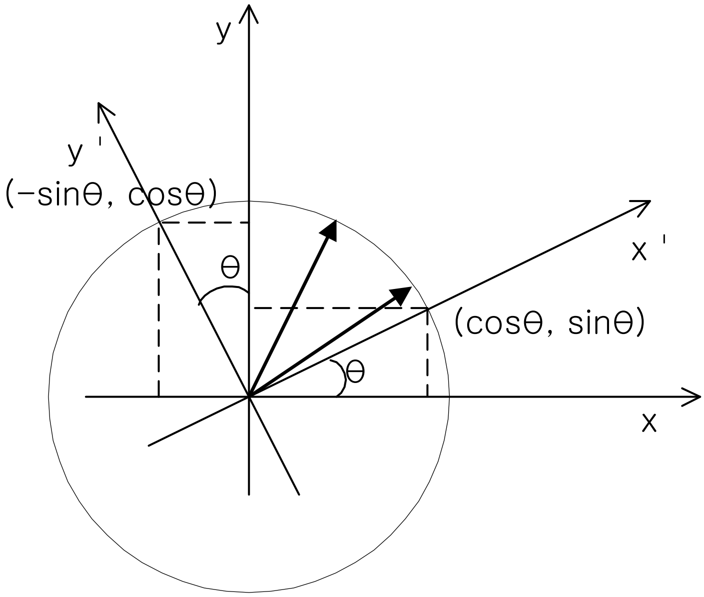


그림. 회전 변환된 새로운 축: 새로운 축은 반지름이 1인 원의 원주를 따라 움직이므로 항상 올소노멀 베이시스 조건을 만족합니다. 이것은 $sin^2\theta + cos^2\theta = 1$에 의해서도 명확합니다.

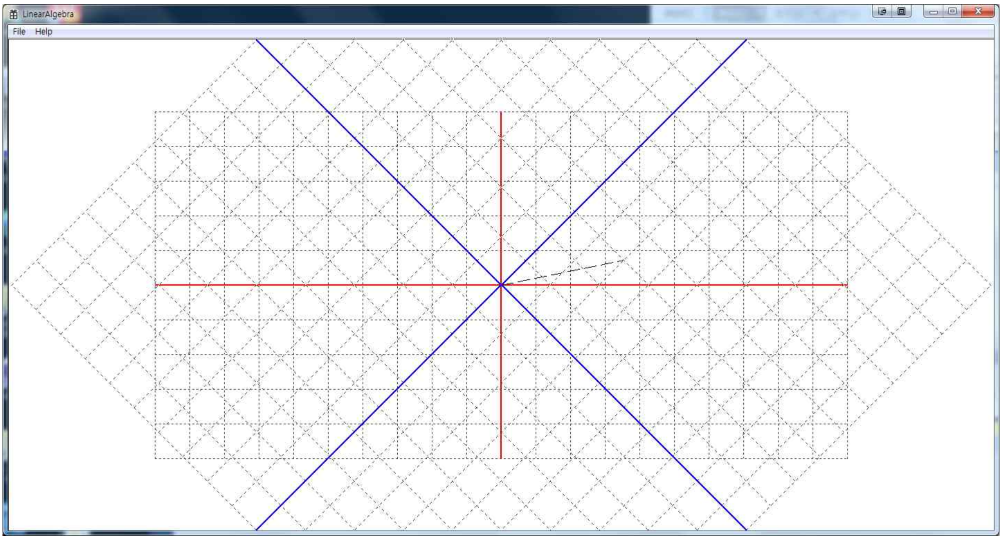
회전 변환된 새로운 축은 다음과 같습니다.

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

이제 우리는 회전된 새로운 점의 위치를 다음과 같이 나타낼 수 있다.

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

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



[그림] 회전 변화(Rotation Transform): 단위원에 대한 삼각함수의 규칙을 이용하면, 회전된 새로운 베이시스를 얻는 것이 가능합니다.

선형 변환(Linear Transform)

① 변환되기 전의 좌표계에서 선(Line)은 변환 후에도 선(Line)이어야 합니다.
② 원점의 위치가 변하지 않아야 합니다.

이렇게 되기 위해서는, 변환전의 격자(Grid)는 변환 후에도 같은 간격을 유지하고, 모두 서로 평행이어야 합니다. 크기 변환, 쉬어 변환 및 회전 변환은 선형 변환이지만, 위치를 바꾸는 변환은 선형 변환이 아닙니다.

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

위의 예에서, 매트릭스 $\begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix}$를 **선형 변환 매트릭스(linear transformation matrix)**라고 합니다.

하나 이상의 벡터 입력에 대한 표현

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

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

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

행렬(Matrix)

숫자들의 직사각형 배열(rectangular array of numbers)을 **매트릭스(matrix)**라 하고, 배열에 있는 각 숫자를 **엘리먼트(element, entry)**라고 합니다. 아래는 매트릭스의 예입니다.

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

특별히 행(row)이 하나뿐인 매트릭스를 **행 매트릭스(row matrix)**라 하고, 열(column)이 하나뿐인 매트릭스를 **열 매트릭스(column matrix)**라 하는데, **벡터(vector)**는 행 매트릭스 혹은 열 매트릭스로 표현할 수 있습니다. 벡터를 행 매트릭스로 표현하면 점(point) 표현과 구분할 수 없으므로 많은 컴퓨터 그래픽스 책에서는 벡터를 열 매트릭스로 표현합니다. 매트릭스의 행의 수가 m이고 열의 수가 n일 때, 이를 **m×n**('엠 바이 엔'이라고 읽습니다) 매트릭스라고 나타냅니다.

A,B,C,D를 매트릭스라 할 때 매트릭스의 합 A+B를 계산하기 위해서 매트릭스의 각 엘리먼트를 합(sum)합니다. 합은 매트릭스의 행과 열의 수가 같은 경우에만 성립합니다. 아래 매트릭스를 고려해 봅시다.

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

위 매트릭스가 주어졌을 때, A + B = $\begin{vmatrix} -2 & 4 & 5 & 4 \\ 1 & 2 & 2 & 3 \\ 7 & 0 & 3 & 5 \end{vmatrix}$ 이며, A+C와 B+C는 매트릭스 덧셈을 할 수 없습니다.

a,b,c,d,m,r,n을 실수(real number) 상수라고 하고, w,x,y,z를 변수라고 합시다. 실수와 매트릭스의 곱 cA는 매트릭스의 각 엘리먼트에 c를 곱해서 얻습니다. 매트릭스간의 곱은 약간 복잡합니다. A가 m×r 매트릭스이고, B가 r×n 매트릭스일 때, **매트릭스의 곱(product) AB**는 m×n 매트릭스이며, (i,j) 엘리먼트는 A매트릭스의 i행과 B매트릭스의 j열의 각 엘리먼트를 곱한 것을 더해서 구합니다. 예를 들어, 아래 매트릭스 A,B를 봅시다.

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

매트릭스의 곱 AB의 (2,3) 엘리먼트는 2*4 + 6*3 + 0*5 = 26입니다.

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

그림. 매트릭스의 곱 AB의 결과: A의 행과 B의 열의 각 요소를 곱한 것의 합이 곱의 각 엘린먼트를 결정합니다.

AB의 결과는 $\begin{vmatrix} 12 & 27 & 30 & 13 \\ 8 & -4 & 26 & 12 \end{vmatrix}$입니다. 매트릭스의 합은 교환법칙(commutative law)이 성립하지만, <u>곱은 교환법칙이 성립하지 않는 다는 것</u>을 주의해야 합니다. 즉 매트릭스 곱은 아래의 성질이 있습니다.

$$AB \neq BA$$

m과 n이 같은 정방 행렬(square matrix)에서 왼쪽위에서 오른쪽아래 방향의 대각 엘리먼트가 모두 1인 행렬을 특별히 **항등 행렬(identity matrix)** I라 하는데, I는 매트릭스 곱셈의 항등원이 됩니다. 즉 m×m 행렬 A는 아래의 성질을 만족합니다.

$$AI = IA = A$$

AB = BA = I를 만족하는 B가 존재할 때, B를 A의 **역(inverse)**이라 하고, A^{-1}로 나타냅니다. 매트릭스의 역의 존재 여부는 매우 중요한데, 예를 들면, 역의 존재는 선형방정식(linear equation)의 해(solution)가 존재함을 의미합니다.

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

그림. 4×4항등행렬: 대각 성분이 1이고, 나머지 엘리먼트는 모두 0인 정방 행렬을 항등 행렬이라고 합니다.

동차 함수(Homogeneous Function)의 정의

$f(tx,ty) = t^{\alpha} f(x,y)$

예) $f(x,y) = x^2 + y^2$

$f(tx,ty) = (tx)^2 + (ty)^2 = t^2x^2 + t^2y^2 = t^2(x^2+y^2)$
$f(tx,ty) = t^2 f(x,y)$

예) $f(x,y) = x^2 + y^2 + 2$
$f(tx,ty) = (tx)^2 + (ty)^2 = t^2x^2 + t^2y^2 + 2 = t^2(x^2+y^2) + 2$
$f(tx,ty) \neq t^{\alpha} f(x,y)$

호모지니어스 함수는 입력에 변수가 지정되더라도, 미분을 적용했을 때, 원래 함수의 미분식을 유지하는 유용한 특징을 가지고 있어서, 미분 방정식에서 유용하게 사용합니다.

여러 선형 방정식의 경우(System of Linear Equation)

f(x,y)=3x + 4y - 6
f(x,y)=7x + 8y

위의 선형 방정식에서 모든 방정식의 결과 값이 모두 0이 되도록 하는 x와 y를 구하기 위해서 식을 다음과 같이 쓸 수 있습니다.

3x + 4y - 6 = 0
7x + 8y = 0

각 선형 방정식의 상수항을 등호의 우변으로 이항하면 식을 다음과 같이 정리할 수 있습니다.

3x + 4y = 6
7x + 8y = 0

일반적인 형태의 선형 방정식 각각 동차 함수가 아닙니다. 상수항이 포함되어 있기 때문입니다.

선형시스템(Linear System)

$$\begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} x + \begin{bmatrix} -1 \\ 1 \end{bmatrix} (y) = \begin{bmatrix} 2 \times x + (-1) \times (y) \\ 1 \times x + 1 \times (y) \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

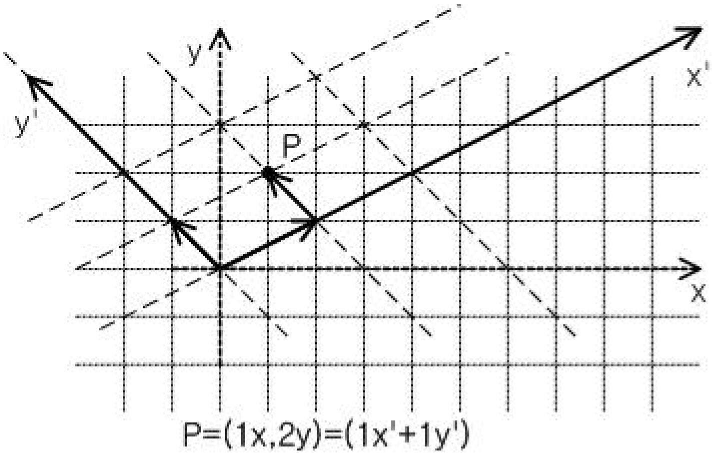
$$2x - 1y = 2x - y = x'$$
$$1x + 1y = x + y = y'$$

$$2x - 1y = 0$$
$$1x + 1y = 0$$

$$2x - 1y = 1$$
$$1x + 1y = 2$$

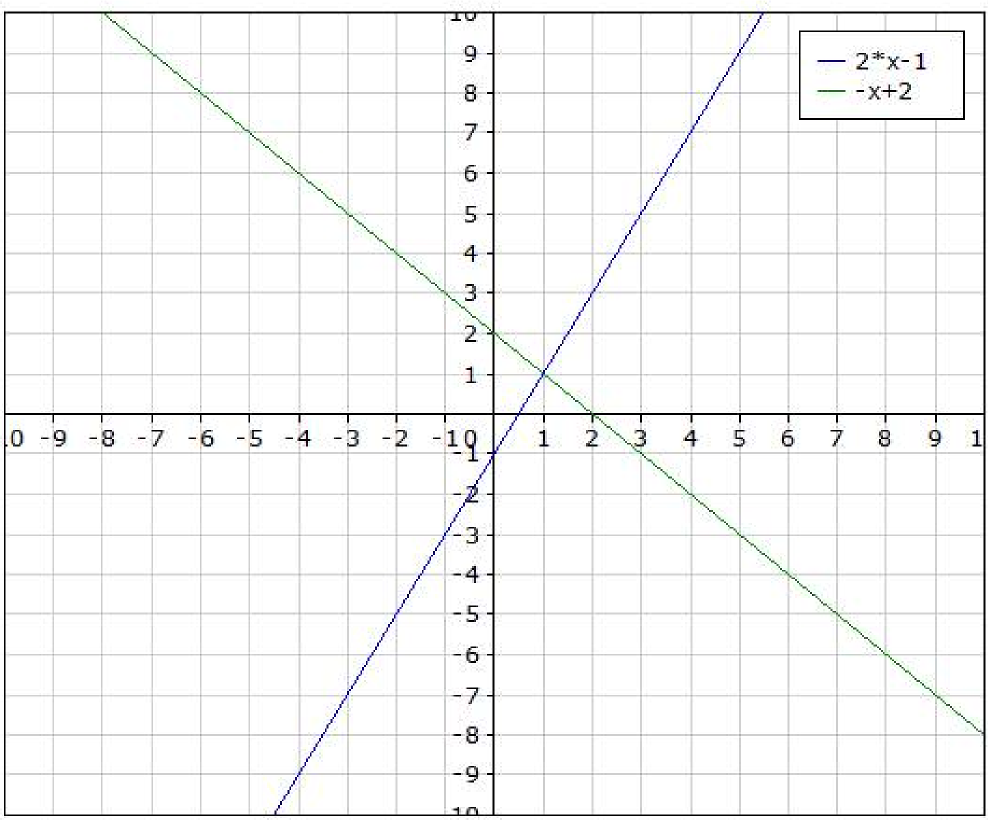
위 연립 방정식을 풀이한다는 것은, 변환 전 (1,2)위치의 위치가, 새로운 베이시스 (2,1), (-1,1)에서는 어떤 위치가 되는지 찾는 것입니다.

$y = 2x - 1$
$1x + 1(2x - 1) = 2$
$3x = 3$
$x = 1$
$y = 1$



[그림] 연립방정식 풀이의 의미: 변환 전 (1,2)위치가 새로운 베이시스 (2,1), (-1,1)에서는 어떤 위치가 되는지 찾는 것입니다.

또한 이것은 두 직선 y=2x-1과 y=-x+2 이 (1,1)위치에서 서로 교차한다는 의미입니다.



[그림] 선형 시스템의 해(Solution): 선형 시스템의 해는 두 직선의 교점을 좌표를 구하는 것으로 해석할 수 있습니다.

아래 식들을 각각 **선형 방정식(linear equation)**이라고 합니다.

$$ax + by = c$$
$$ax + by + cz = d$$

선형 방정식의 유한 집합(finite set)을 선형 방정식 시스템(system of linear equation) 혹은 **선형 시스템(linear system)**이라고 합니다. 다음과 같은 선형 시스템을 고려해 봅시다.

$$3x + 4y + 5z = 6$$
$$7x + 8y + 9z = 0$$

1x + 2y + 3z = 4

위 선형 시스템에서 변수를 생략한 아래의 매트릭스를 **오그멘티드 매트릭스(augmented matrix)**라고 합니다.

$$\begin{vmatrix} 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 0 \\ 1 & 2 & 3 & 4 \end{vmatrix}$$

시스템의 모든 상수항이 0인 경우, 시스템은 변수의 수와 계수(coefficient)의 값에 상관없이 모든 변수의 값이 0이면 성립하는 당연한 해(trivial solution)를 가지는데 이러한 시스템을 **호모지니어스(homogeneous)**하다고 합니다. 우리는 후에 3차원 변환 매트릭스(3D transform matrix)를 homogeneous 매트릭스로 표현할 것인데, 이것은 이동 변환을 일관된 매트릭스 연산으로 표현하기 위해 우변(right side)이 0인 오그멘티드augmented 매트릭스, 즉 **호모지니어스homogeneous 매트릭스**□를 사용할 것이기 때문입니다.

□ 시스템이 당연한 해를 가지기 위해서는 선형 방정식의 상수텀이 모두 0이어야 합니다. **입력 벡터에 특정한 가정**을 하면 호모지니어스 시스템을 만들 수 있는데, 이렇게 만들어진 오그멘티드 매트릭스를 호모니지어스 매트릭스라고 합니다.

가우스 소거법(Gaussian elimination)

(선형 시스템의 해를 구하는 방법)

어파인 변환(Affine Transformation)

크기 변환(scaling transform) 매트릭스는 다음과 같이 표현할 수 있습니다.

$$\begin{vmatrix} s & 0 \\ 0 & t \end{vmatrix}$$

이것은 x축의 방향으로 s, y축의 방향으로 t만큼 스케일된 변환을 나타냅니다. 그렇다면 2차원에서의 위치 이동(translation)을 2×2 변환 매트릭스로 나타낼 수 있을까요? 예를 들면 원점 (0,0)을 새로운 위치 (x',y') = (a,b)로 옮기는 위치 변환을 생각해 봅시다. 이것은 다음과 같은 선형 시스템(linear system)으로 표현할 수 있습니다.

x' = 1·x + 0·y + a
y' = 0·x + 1·y + b

선형 시스템의 호모지니어스 매트릭스는 $\begin{vmatrix} 1 & 0 & a \\ 0 & 1 & b \end{vmatrix}$이므로 이것을 2×1 매트릭스 $\begin{vmatrix} x \\ y \end{vmatrix}$와 곱할 수는 없습니다. 하지만, 우리는 가상의 선형 방정식을 추가할 수 있습니다.

1 = 0·x + 0·y + 1

이제 다음과 같은 선형 시스템을 고려할 수 있습니다.

x' = 1·x + 0·y + a
y' = 0·x + 1·y + b
1 = 0·x + 0·y + 1

2차원 벡터를 나타내기 위해 세번째 가상의 점 w를 추가하고, 이 값을 항상 1이라고 가정합니다. 이제 위치 (x,y)는 (x,y,w) 즉, (x,y,1)로 표현되며, 호모지니어스 매트릭스는 아래와 같습니다.

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

그러므로 변환 식은 다음과 같이 표현할 수 있습니다.

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

그러므로 호모지니어스 매트릭스로 표현된 회전 변환은 다음과 같습니다.

$$\begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

크기 변환은 다음과 같이 나타낼 수 있습니다.

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

5 KMatrix3 클래스의 구현

KMatrix3 클래스

```
class KMatrix3
{
public:
    static KMatrix3 zero;
    static KMatrix3 identity;

public:
    float   _11, _12, _13;
    float   _21, _22, _23;
    float   _31, _32, _33;

public:
    KMatrix3(float e11 = 1.0f, float e12 = 0.0f, float e13 = 0.0f
        , float e21 = 0.0f, float e22 = 1.0f, float e23 = 0.0f
        , float e31 = 0.0f, float e32 = 0.0f, float e33 = 1.0f)
    {
```

```
        _11 = e11; _12 = e12; _13 = e13;
        _21 = e21; _22 = e22; _23 = e23;
        _31 = e31; _32 = e32; _33 = e33;
    }
    ~KMatrix3() {}
    void Set(float e11, float e12, float e13
        , float e21, float e22, float e23
        , float e31, float e32, float e33)
    {
        _11 = e11; _12 = e12; _13 = e13;
        _21 = e21; _22 = e22; _23 = e23;
        _31 = e31; _32 = e32; _33 = e33;
    }

    void SetIdentity()
    {
        _11 = 1.0f; _12 = 0.0f; _13 = 0.0f;
        _21 = 0.0f; _22 = 1.0f; _23 = 0.0f;
        _31 = 0.0f; _32 = 0.0f; _33 = 1.0f;
    }

    void SetRotation(float theta)
    {
        SetIdentity();
        _11 = cosf(theta); _12 = -sinf(theta);
        _21 = sinf(theta); _22 = cosf(theta);
    }

    void SetShear(float shearXParallelToY, float shearYParallelToX)
    {
        SetIdentity();
        _11 = 1.0f; _12 = shearYParallelToX;
        _21 = shearXParallelToY; _22 = 1.0f;
    }

    void SetScale(float uniformScale)
    {
        SetIdentity();
        _11 = uniformScale;
        _22 = uniformScale;
        _33 = uniformScale;
    }
```

```
    void SetTranslation(float tx, float ty)
    {
        SetIdentity();
        _13 = tx;
        _23 = ty;
    }

    bool GetBasis(KVector2& basis_, int basisIndexFrom0_)
    {
        if (basisIndexFrom0_ == 0) {
            basis_.x = _11;
            basis_.y = _21;
        }
        else if (basisIndexFrom0_ == 1)
        {
            basis_.x = _12;
            basis_.y = _22;
        }
        else
        {
            return false;
        }

        return true;
    }
};

inline KVector2 operator*(const KVector2& v, const KMatrix3& m)
{
    KVector2 temp;
    temp.x = v.x*m._11 + v.y*m._21 + 1.0f*m._31;
    temp.y = v.x*m._12 + v.y*m._22 + 1.0f*m._32;
    const float z = v.x*m._13 + v.y*m._23 + 1.0f*m._33;
    temp.x /= z; // homogeneous divide
    temp.y /= z;
    return temp;
}

inline KVector2 operator*(const KMatrix3& m, const KVector2& v)
{
    KVector2 temp;
    temp.x = m._11*v.x + m._12*v.y + m._13 * 1.0f;
    temp.y = m._21*v.x + m._22*v.y + m._23 * 1.0f;
```

```
    const float z = m._31*v.x + m._32*v.y + m._33*1.0f;
    temp.x /= z; // homogeneous divide
    temp.y /= z;
    return temp;
}

inline KMatrix3 operator*(float scalar, const KMatrix3& m)
{
    KMatrix3 temp;
    temp._11  =  scalar*m._11;  temp._12  =  scalar*m._12;  temp._13  =
scalar*m._13;
    temp._21  =  scalar*m._21;  temp._22  =  scalar*m._22;  temp._23  =
scalar*m._23;
    temp._31  =  scalar*m._31;  temp._32  =  scalar*m._32;  temp._33  =
scalar*m._33;
    return temp;
}

// composition: matrix-matrix multiplication
inline KMatrix3 operator*(const KMatrix3& m0, const KMatrix3& m1)
{
    KMatrix3 temp;
    temp._11 = m0._11*m1._11 + m0._12*m1._21 + m0._13*m1._31;
    temp._12 = m0._11*m1._12 + m0._12*m1._22 + m0._13*m1._32;
    temp._13 = m0._11*m1._13 + m0._12*m1._23 + m0._13*m1._33;
    temp._21 = m0._21*m1._11 + m0._22*m1._21 + m0._23*m1._31;
    temp._22 = m0._21*m1._12 + m0._22*m1._22 + m0._23*m1._32;
    temp._23 = m0._21*m1._13 + m0._22*m1._23 + m0._23*m1._33;
    temp._31 = m0._31*m1._11 + m0._32*m1._21 + m0._33*m1._31;
    temp._32 = m0._31*m1._12 + m0._32*m1._22 + m0._33*m1._32;
    temp._33 = m0._31*m1._13 + m0._32*m1._23 + m0._33*m1._33;
    return temp;
}
```

@