Homogeneous Function

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

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) $f(x,y) = x^2 + y^2 + 2$

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

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

System of Linear Equation

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

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

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

Linear System

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

$$\begin{aligned} 2x - 1y = 2x - y = x' \\ 1x + 1y = x + y = y' \end{aligned}$$

$$\begin{aligned} 2x - 1y = 0 \\ 1x + 1y = 0 \end{aligned}$$

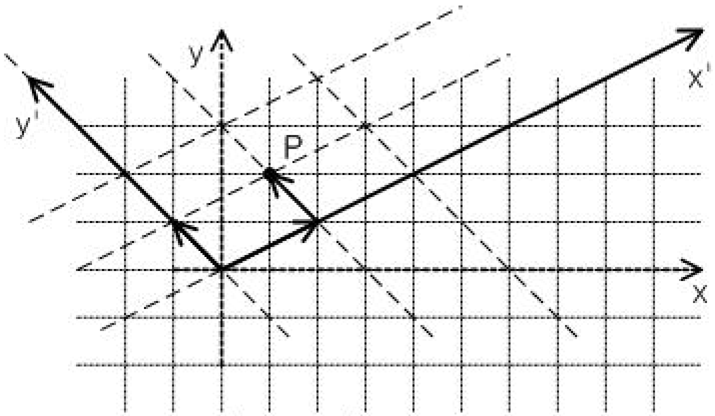
linearly independent

now assume, x'=1, y'=2

$$
\begin{aligned}
2x - 1y &= 1 \\
1x + 1y &= 2
\end{aligned}
$$

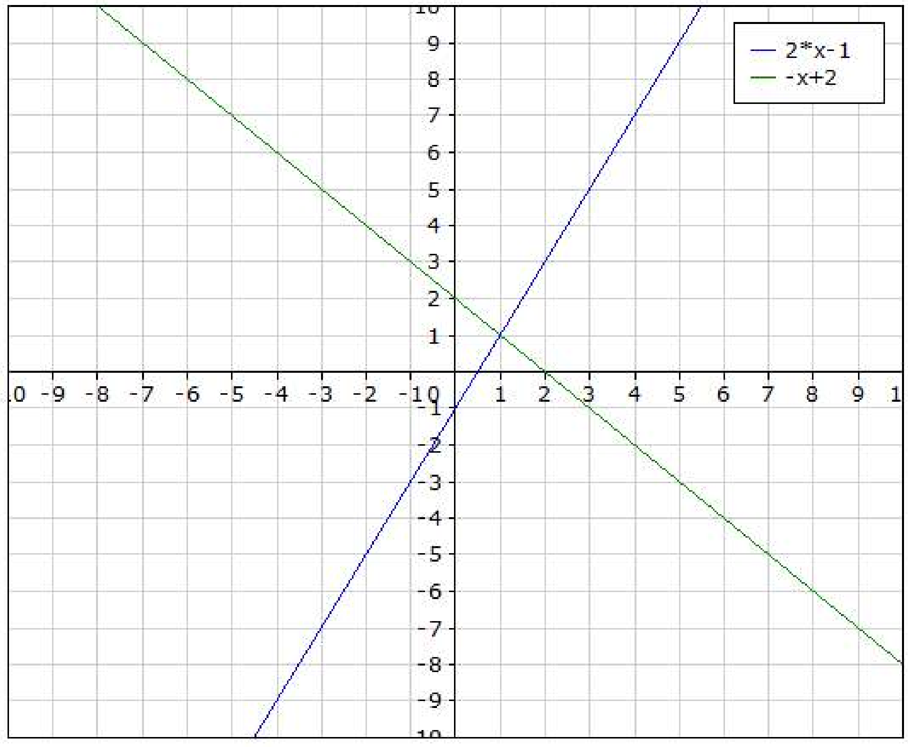
Solving Linear System

$$
\begin{aligned}
&y = 2x - 1 \\
&1x + 1(2x - 1) = 2 \\
&3x = 3 \\
&x = 1 \\
&y = 1
\end{aligned}
$$



[Fig] Solution of linear system: finding position at given basis (2,1), (-1,1) when position (1,2) at standard basis is known.

Finding intersection point of two lines, y=2x-1 and y=-x+2.



[Fig] Solution of linear system: solution is the intersection point of two lines

Homogeneous Matrix

linear equation

$$ax + by = c$$

$$ax + by + cz = d$$

system of linear equation==**linear system**

$$
\begin{aligned}
3x + 4y + 5z &= 6 \\
7x + 8y + 9z &= 0 \\
1x + 2y + 3z &= 4
\end{aligned}
$$

augmented matrix

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

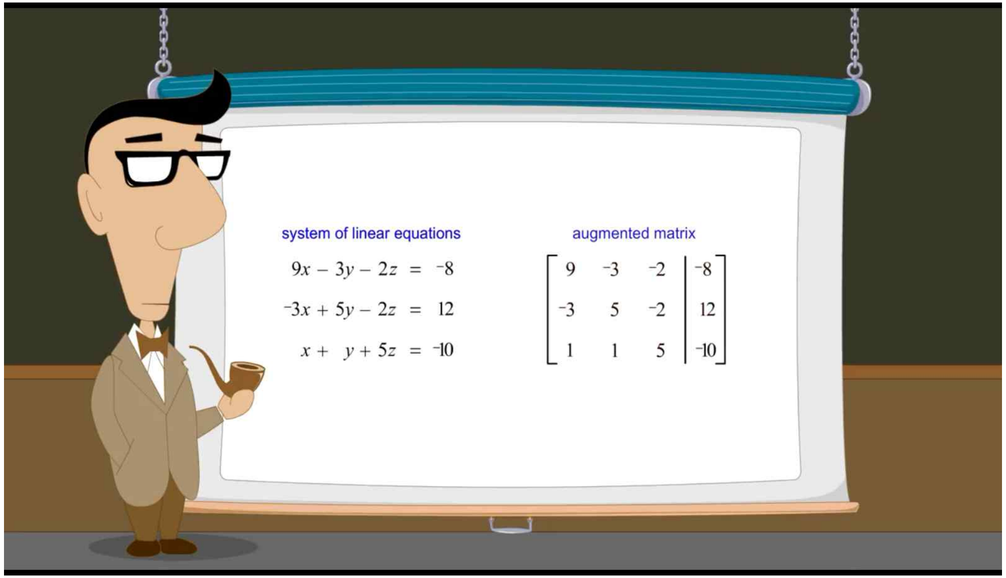
homogeneous : if all constant terms are 0

homogeneous matrix

Gaussian elimination

Title: Algebra 54 - Gaussian Elimination

Link: https://www.youtube.com/watch?v=2GKESu5atVQ



[Fig] Gaussian elimination

Affine Transformation

scaling transform

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

Translation Transform

move position to (a,b) from (0,0)

$$x' = 1 \cdot x + 0 \cdot y + a$$
$$y' = 0 \cdot x + 1 \cdot y + b$$

We can't multiply homogeneous matrix $\begin{vmatrix} 1 & 0 & a \\ 0 & 1 & b \end{vmatrix}$ with 2×1 matrix $\begin{vmatrix} x \\ y \end{vmatrix}$.

Adding one more linear equation

$$1 = 0 \cdot x + 0 \cdot y + 1$$

Now, linear system will be:

$$x' = 1 \cdot x + 0 \cdot y + a$$
$$y' = 0 \cdot x + 1 \cdot y + b$$
$$1 = 0 \cdot x + 0 \cdot y + 1$$

There is a virtual axis w, and it's value is always 1.
(x,y) --> (x,y,w) --> (x,y,1)

Then homogeneous matrix will be:

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

Translation transform equation will be:

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

2-dimensional 3×3 Rotation matrix

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

2-dimensional 3×3 Scale matrix

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

affine transform!

We assume w is always 1, it means if w' is not equal to 1 after transformation, then all x',y'must be modified to make $(x',y',1)$.

class KMatrix3

SetTranslation()

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

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

Matrix Vector Multiplication

Vector is assumed as (x,y,1)

```
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;
}
```

Note the need for Homogeneous divide!

Matrix Matrix Multiplication

```
// 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;
}
```

Full list of class KMatrix3

```
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;
```

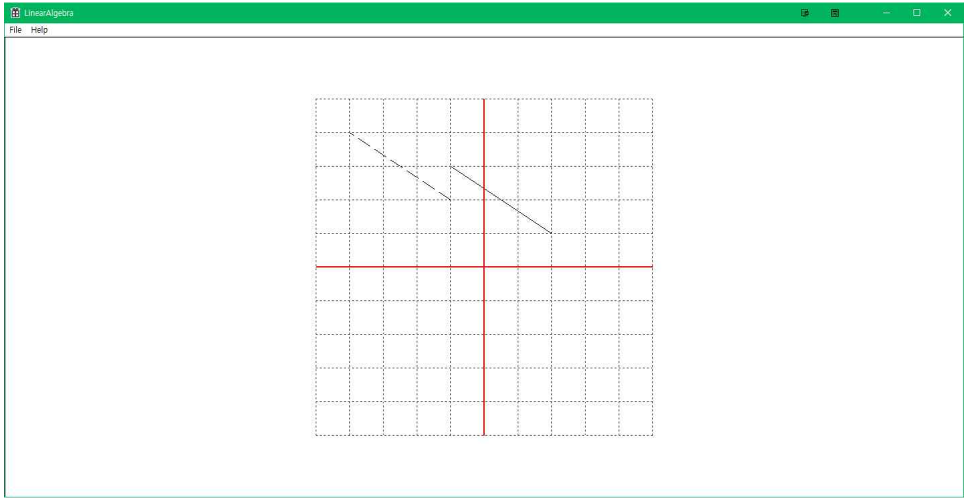
```
            basis_.y = _22;
        }
        else
        {
            return false;
        }

        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;
}
```

LinearAlgebra_Step06 Homogeneous Matrix3 Project

Now we use 3×3 matrix for 2-d transformations.



[Fig] Affine Transform: with 3×3 matrix, we can do 2-d affine transform

Dotted line: translation then rotation

Solid line: rotation then translation

Exercise

1 Implement GetInverse() function for class KMatrix3

[End of Document]