Computer Graphics and Programming
Lecture 2

Vector and Matrix Calculation

Jeong-Yean Yang

2020/10/22

Vector Space

- Vector Definition in 3 Dim. Space

$$v = (x, y, z)$$

Linear Vector Space

Vector always
Pass through origin.

- Euclidean Distance(Two Norm)

$$\| v \| = \sqrt{x^2 + y^2 + z^2}$$

- Vector Addition:

$$v = (x, y, z),\ a = (a_x, a_y, a_z)$$

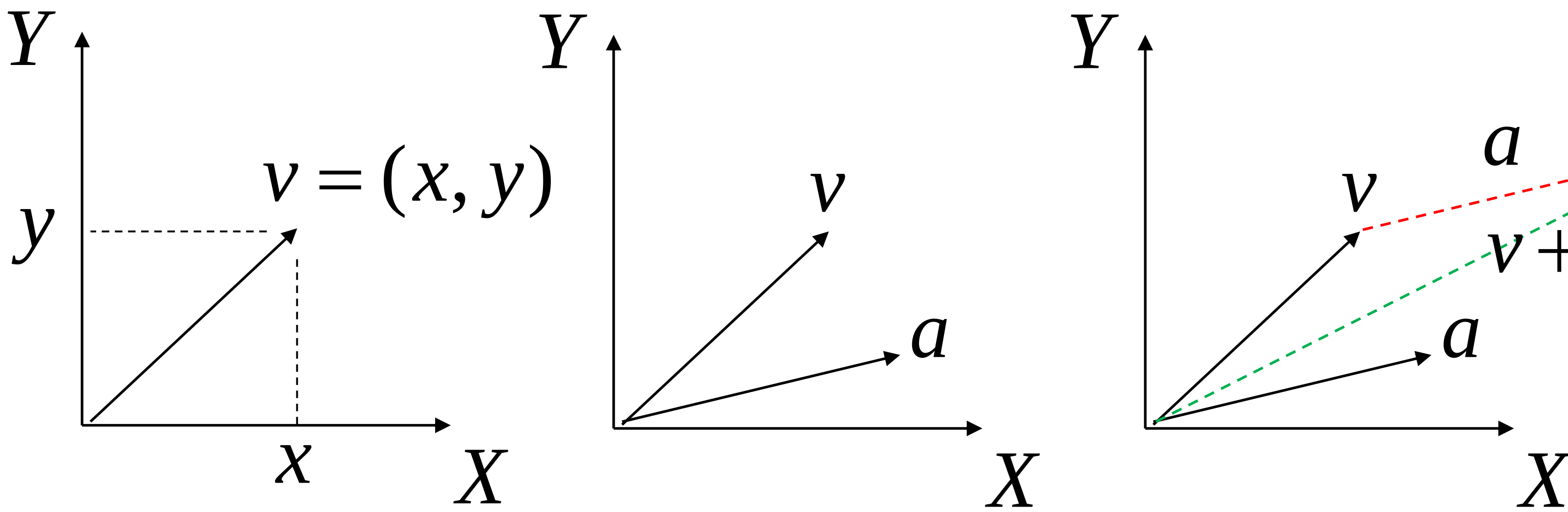
$$v' = v + a = (x + a_x, y + a_y, z + a_z)$$

Moving Vector (1)

- Simple vector movement = Vector addition

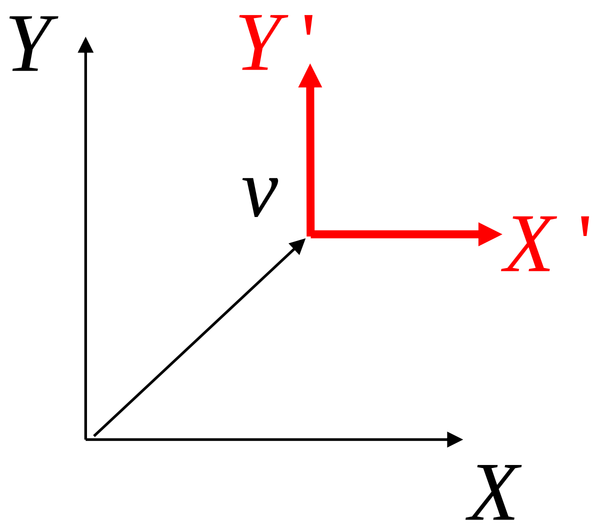
$$v' = v + a$$

- Example in 2D



Moving Vector (2)
: Moving Coordinate

- It is called Transform in Graphics and Robotics



$$v \ in \ \text{XY space}$$

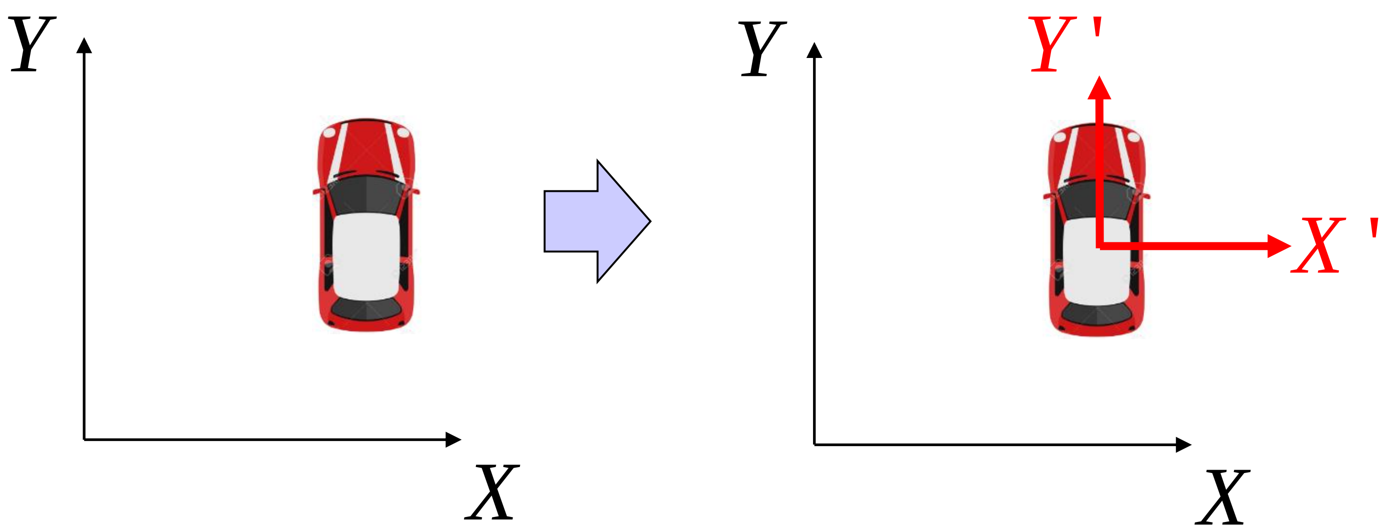
$$\Downarrow$$

$$v' \ in \ \text{X'Y' space}$$

$$v \ in \ XY \ space: \ v = (x, y)$$

$$v \ in \ X'Y' \ space = v' = \ (x', y') = (0,0) \text{ in X'Y' space}$$

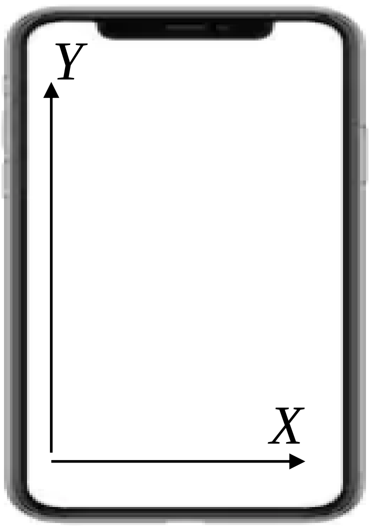
Basic Concept: Think Object in 2D space



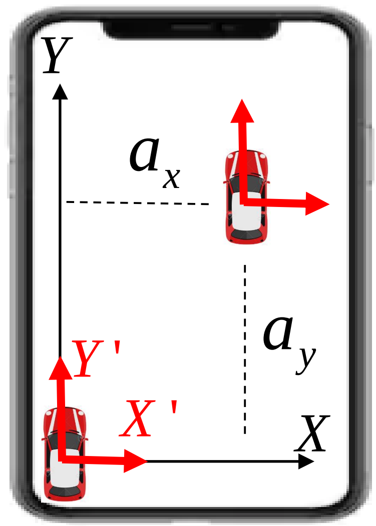
- We set the Coordinate on each Object.
  - Car, air plane, bullet, and so on.
- Moving is not vector addition,

  but moving a car is moving coordinate, X'Y'.

Graphics with Coordinate Transform



Screen Coordinate
XY



Set Car Coordinate
With X'Y'

v' : Car Position

$v' = (0,0)$ in X'Y'

$$Moving\ (a_x, a_y)$$

$$X = X' + a_x$$

$$Y = Y' + a_y$$

Transform

$$v' = (0,0)\ in\ X'Y'$$

$$v = (0 + a_x, 0 + a_y)\ in\ XY$$

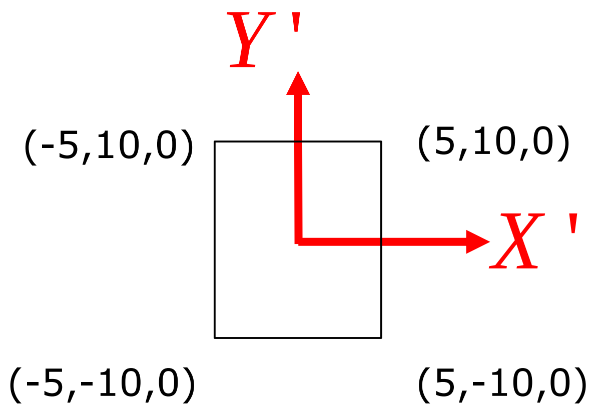
Demo)

uWnd-08-Transform-Trans

Translation Example
Demo: uWnd-08-Transform-Trans

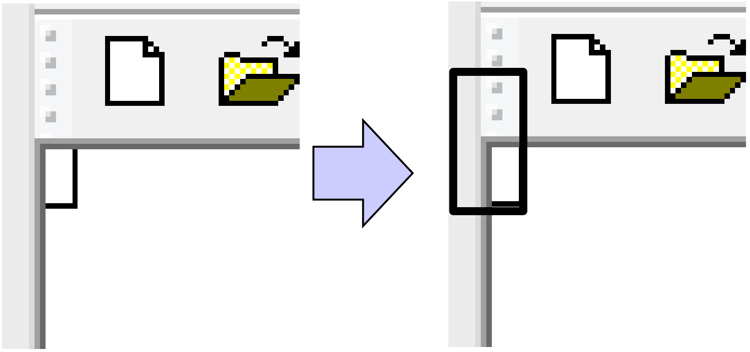
1) Create a rectangle car

```
car[0]  = uVector(-5,-10,0);
car[1]  = uVector(5,-10,0);
car[2]  = uVector(5,10,0);
car[3]  = uVector(-5,10,0);
```

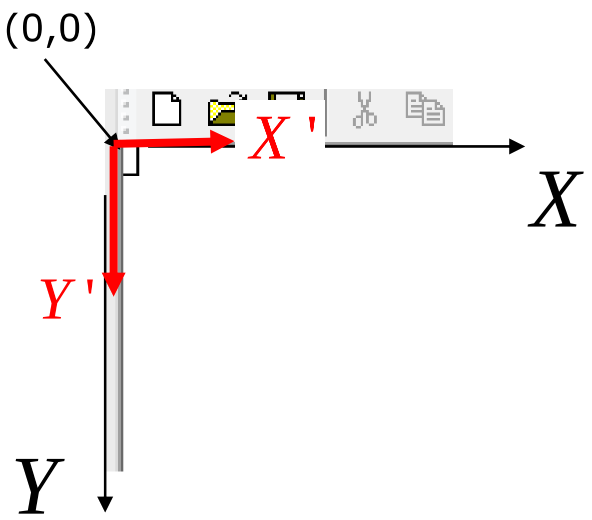


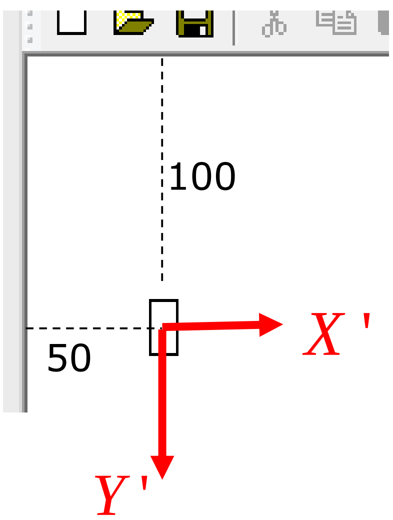
2) Draw a rectangle car

```
void uWnd::Draw(CDC *pDC)
{
    pDC->MoveTo(car[0].x,car[0].y);
    pDC->LineTo(car[1].x,car[1].y);
    pDC->LineTo(car[2].x,car[2].y);
    pDC->LineTo(car[3].x,car[3].y);
    pDC->LineTo(car[0].x,car[0].y);
}
```



Translation (50,100,0)





Translation with t(50,100,0)

```
car[0]  = uVector(-5,-10,0);
car[1]  = uVector(5,-10,0);
car[2]  = uVector(5,10,0);
car[3]  = uVector(-5,10,0);

uVector t(50,100,0);
car[0]  = car[0]+t;
car[1]  = car[1]+t;
car[2]  = car[2]+t;
car[3]  = car[3]+t;
```

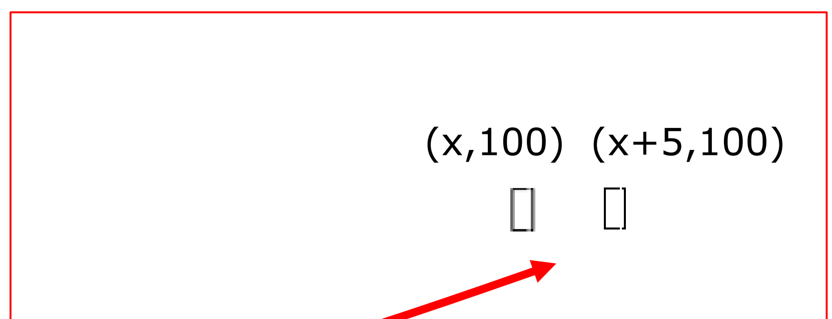
Animation with Translation (Using uWnd::Run)

- Windows has Timer function
  - Event OnTimer is subclassed → uWnd::Run()

- Repaint screen
  - Calling Invalidate() repaint window → uWnd::Redraw()

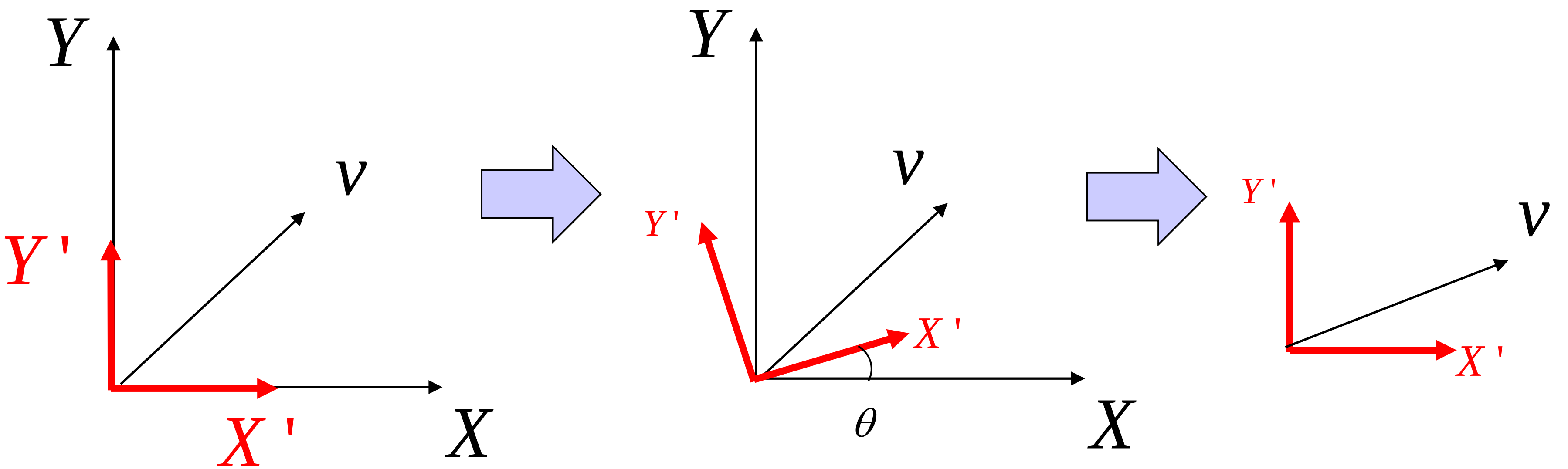
```
void uWnd::Run()
{
    for (int i=0;i<4;i++)
    car[i]  = car[i]+uVector(5,0,0);
    Redraw();
}
```

Demo: uWnd-09-uWnd-Run



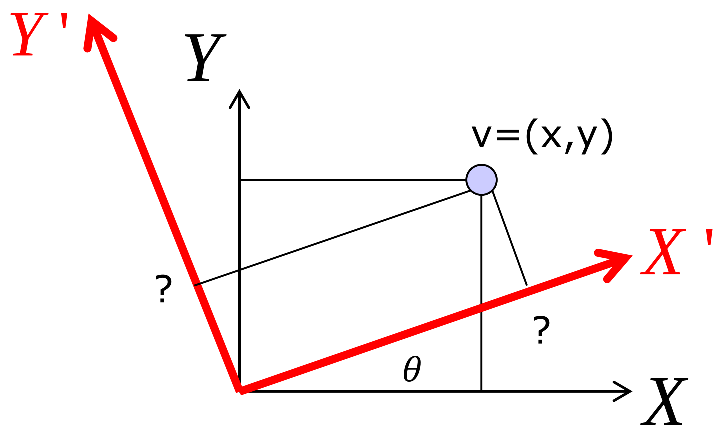
Transform= Translation + Rotation +Scaling

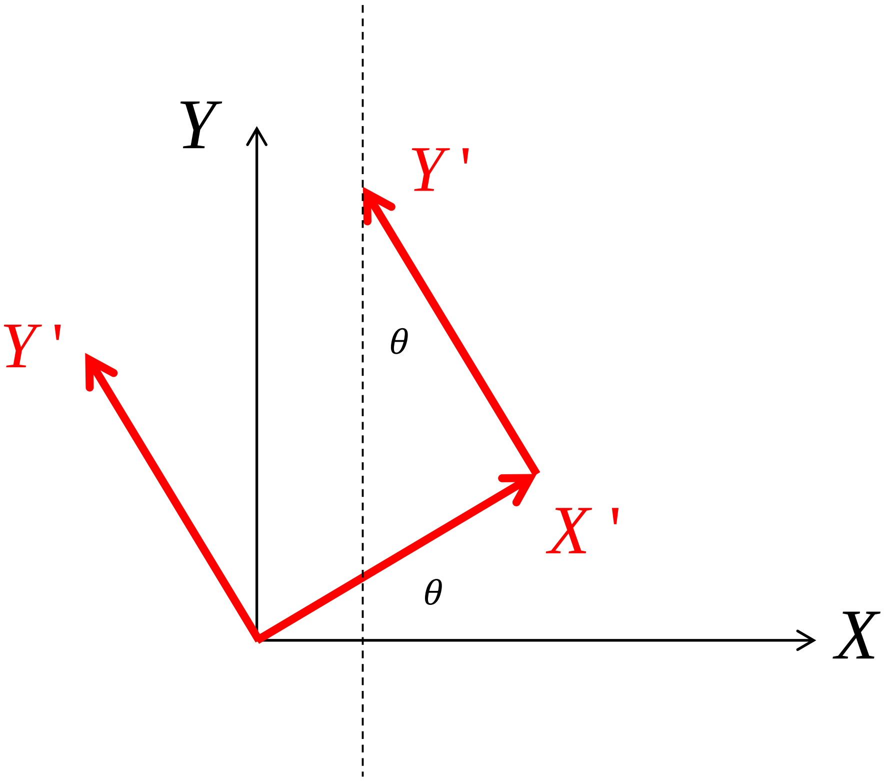
- Coordinate Rotation



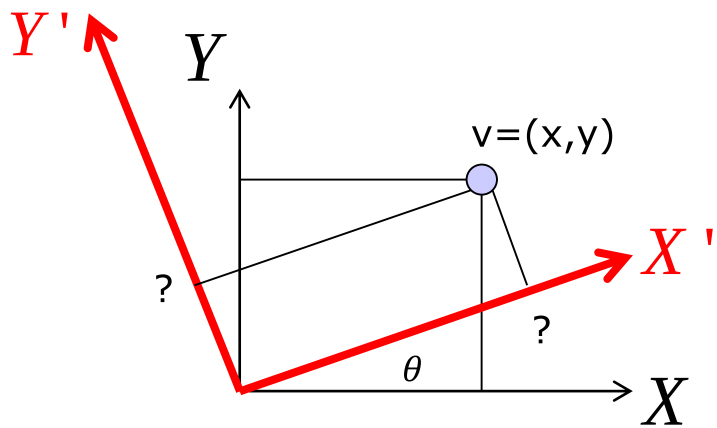
$$v \ in \ XY = v' \ in \ X'Y'$$

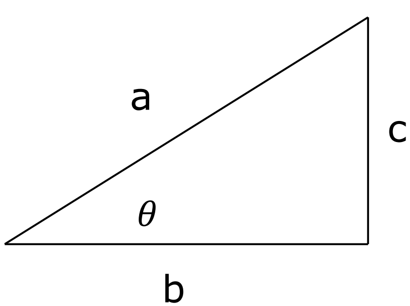
Rotation Formula





Rotation Formula

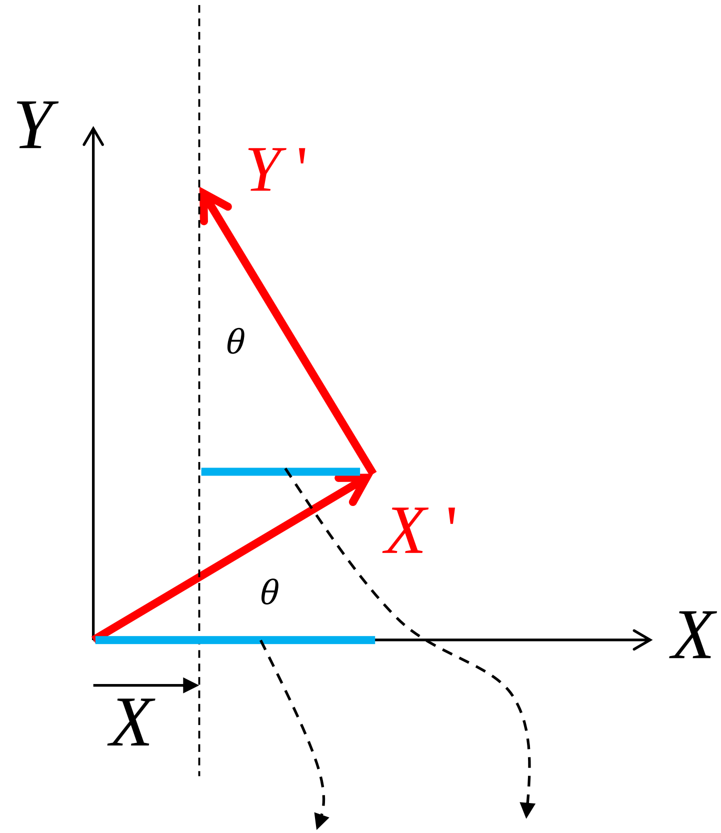




$$\cos\theta = \frac{b}{a} \quad \rightarrow \quad b = a\cos\theta$$

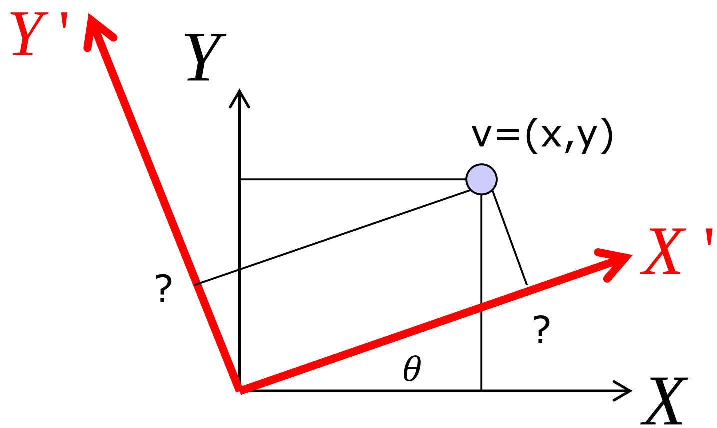
$$\sin\theta = \frac{c}{a}$$

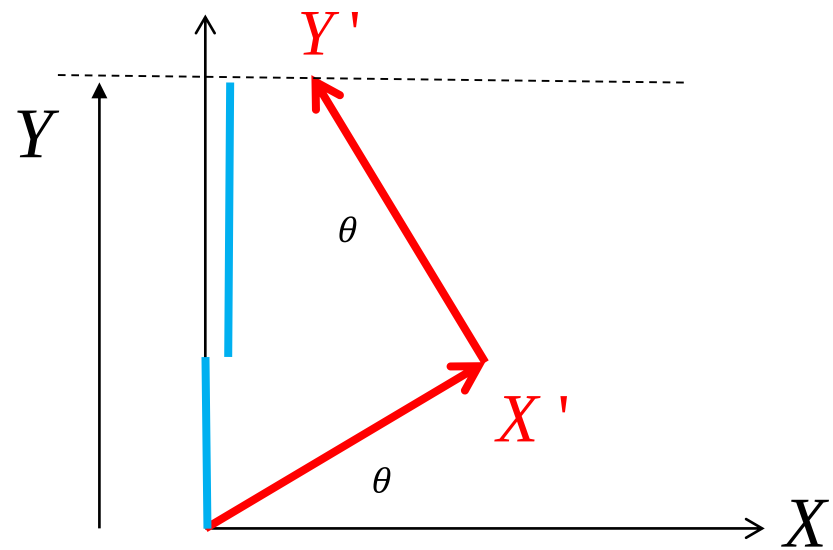
$$\tan\theta = \frac{c}{b}$$

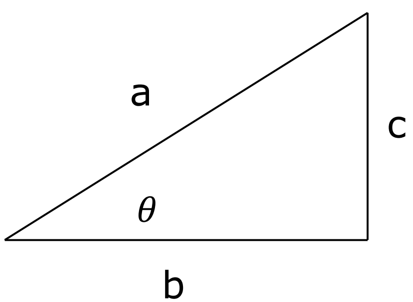


$$X = X'\cos\theta - Y'\sin\theta$$

Rotation Formula







$$\cos\theta = \frac{b}{a} \rightarrow b = a\cos\theta$$

$$\sin\theta = \frac{c}{a}$$

$$\tan\theta = \frac{c}{b}$$

$$X = X'\cos\theta - Y'\sin\theta$$

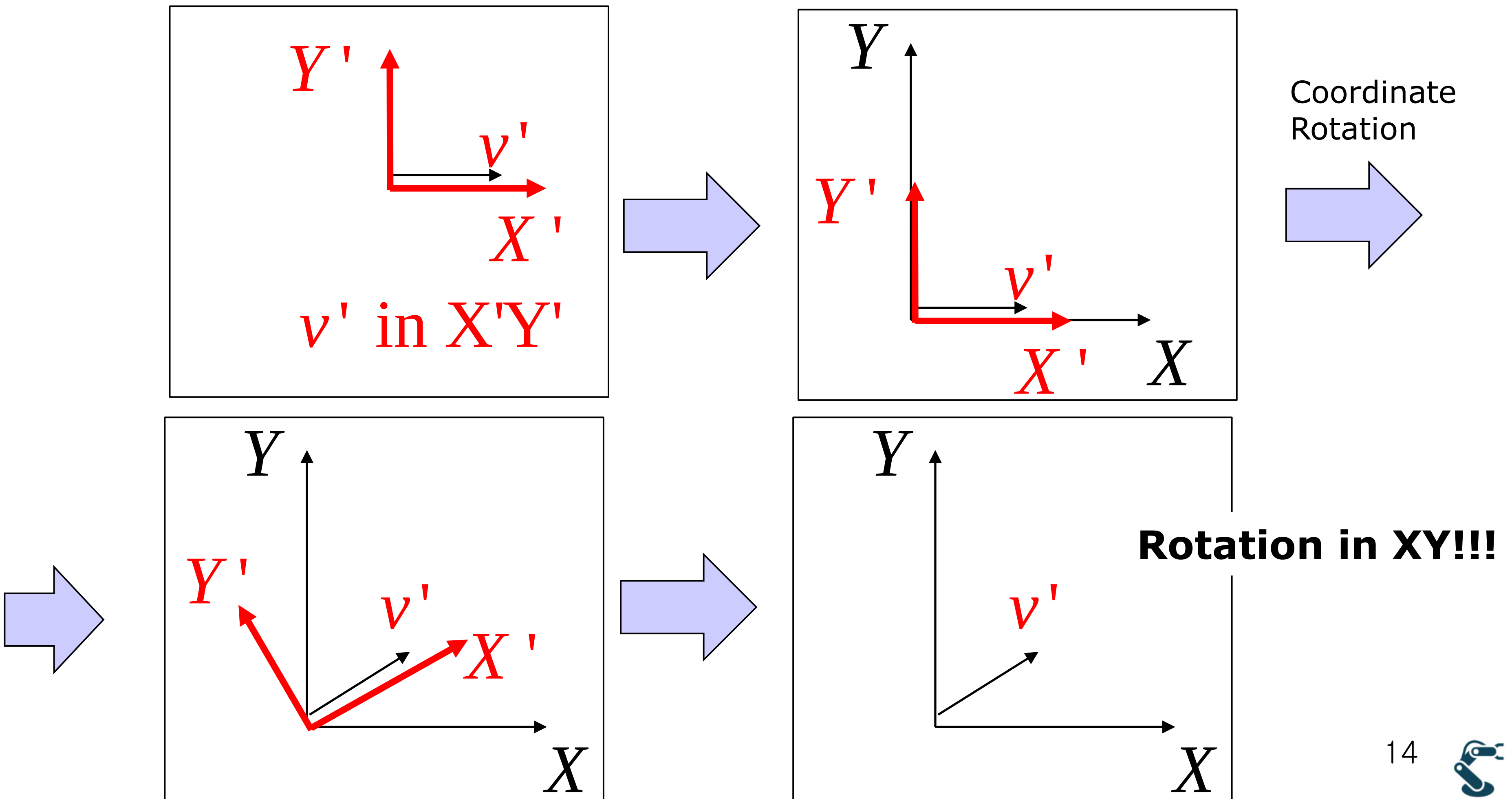
$$Y = X'\sin\theta + Y'\cos\theta$$

Rotation Formula

$$X = X'\cos\theta - Y'\sin\theta$$

$$Y = X'\sin\theta + Y'\cos\theta$$

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X' \\ Y' \end{pmatrix}$$



Remind that Sin, Cos in C/C++ use Radian.

- R =sin(q)
  - q is NOT degree, but is RADIAN

- Radian, π

$$\pi(rad) = 180^\circ(\deg)$$

- Conversion, RAD(q) or DEG(R)

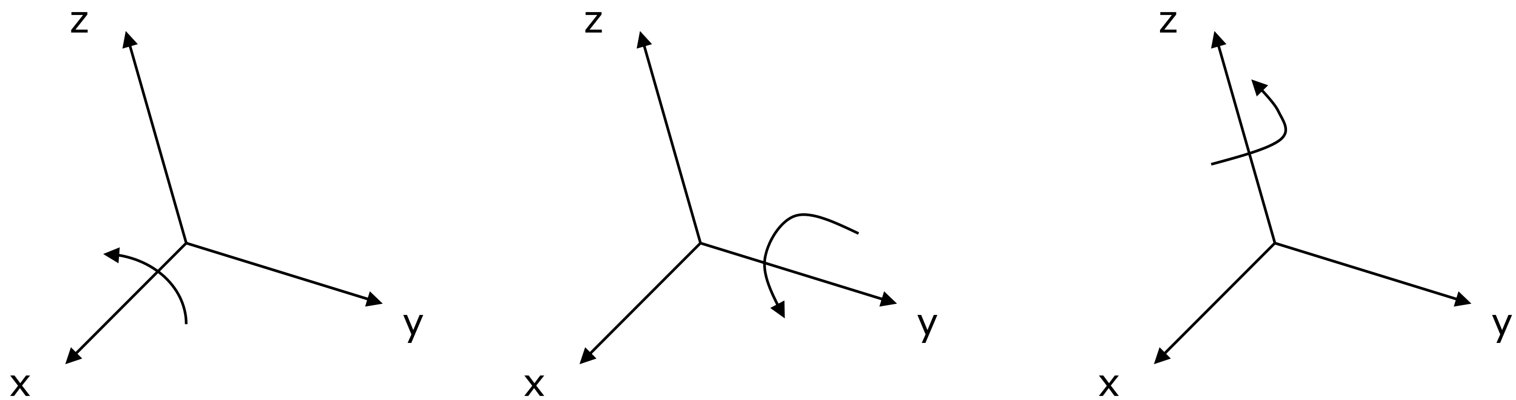
$$\frac{\pi(rad)}{180} = \frac{180^\circ}{180}(\deg) = 1^\circ(\deg)$$

$$\therefore \frac{\pi x}{180}(rad) = x^\circ(\deg)$$

$$\frac{\pi(rad)}{\pi} = 1(rad) = \frac{180^\circ}{\pi}(\deg)$$

$$\therefore x(rad) = \frac{x180^\circ}{\pi}(\deg)$$

Rotation along x, y, and z



$$R_x(\theta)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{bmatrix}\qquad R_y(\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}\qquad R_z(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$example)\ v'=R_x v=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{bmatrix}\begin{pmatrix}x\\ y\\ z\end{pmatrix}=\begin{pmatrix}x\\ y\cos\theta - z\sin\theta\\ y\sin\theta + z\cos\theta\end{pmatrix}$$

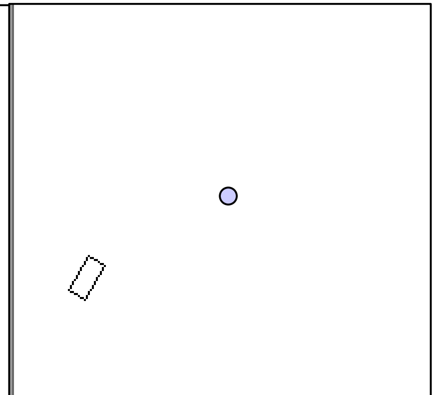
Demo: Rotation
uWnd-11-Transform-Rot

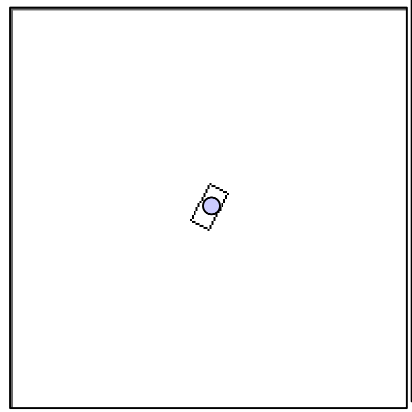
```
uVector uVector::Rot(float q)
{
    uVector ret;
    float r = RAD(q);
    float s = sin(r);
    float c = cos(r);

    ret.x   = c*x-s*y;
    ret.y   = s*x+c*y;
    ret.z   = z;
    return ret;
}
```

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X' \\ Y' \end{pmatrix}$$

Demo: Rotation
uWnd-11-Transform-Rot

```
uWnd::uWnd()
{
```
v
```
    car[0]   = uVector(-5,-10,0);
    car[1]   = uVector(5,-10,0);
    car[2]   = uVector(5,10,0);
    car[3]   = uVector(-5,10,0);

    uVector t(100,100,0);
```
v'
```
    car[0]   = car[0]+t;
    car[1]   = car[1]+t;
    car[2]   = car[2]+t;
    car[3]   = car[3]+t;

    float q = 30;
```
v''
```
    car[0]   = car[0].Rot(q);
    car[1]   = car[1].Rot(q);
    car[2]   = car[2].Rot(q);
    car[3]   = car[3].Rot(q);
}
```





```
uWnd::uWnd()
{
```
v
```
    car[0]   = uVector(-5,-10,0);
    car[1]   = uVector(5,-10,0);
    car[2]   = uVector(5,10,0);
    car[3]   = uVector(-5,10,0);

    float q = 30;
```
v'
```
    car[0]   = car[0].Rot(q);
    car[1]   = car[1].Rot(q);
    car[2]   = car[2].Rot(q);
    car[3]   = car[3].Rot(q);

    uVector t(100,100,0);
```
v''
```
    car[0]   = car[0]+t;
    car[1]   = car[1]+t;
    car[2]   = car[2]+t;
    car[3]   = car[3]+t;
}
```

$$v' = v + T$$

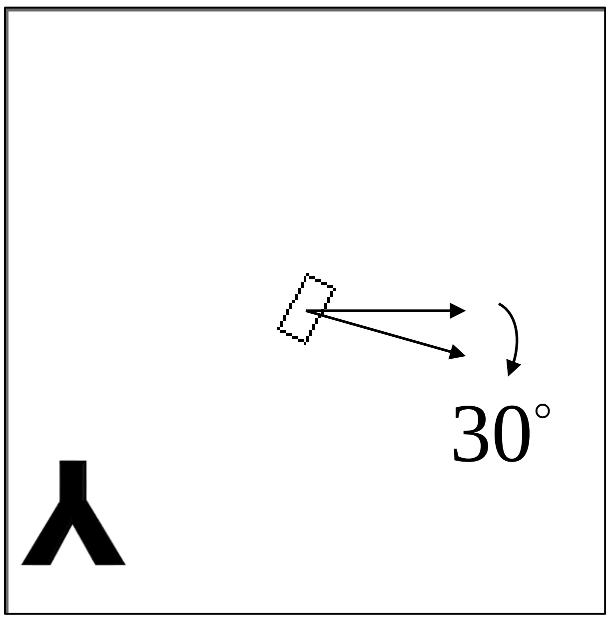
$$v'' = R * v'$$

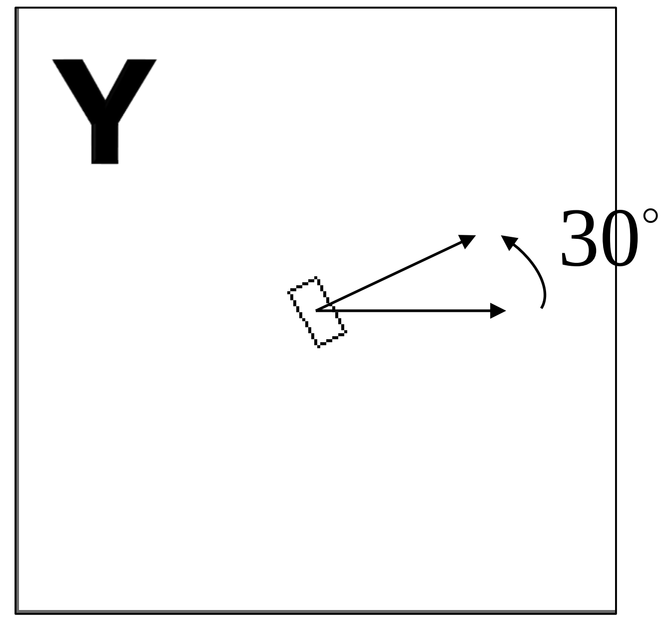
$$v' = R * v$$

$$v'' = v' + T$$

Is it Rotation with 30 degree?

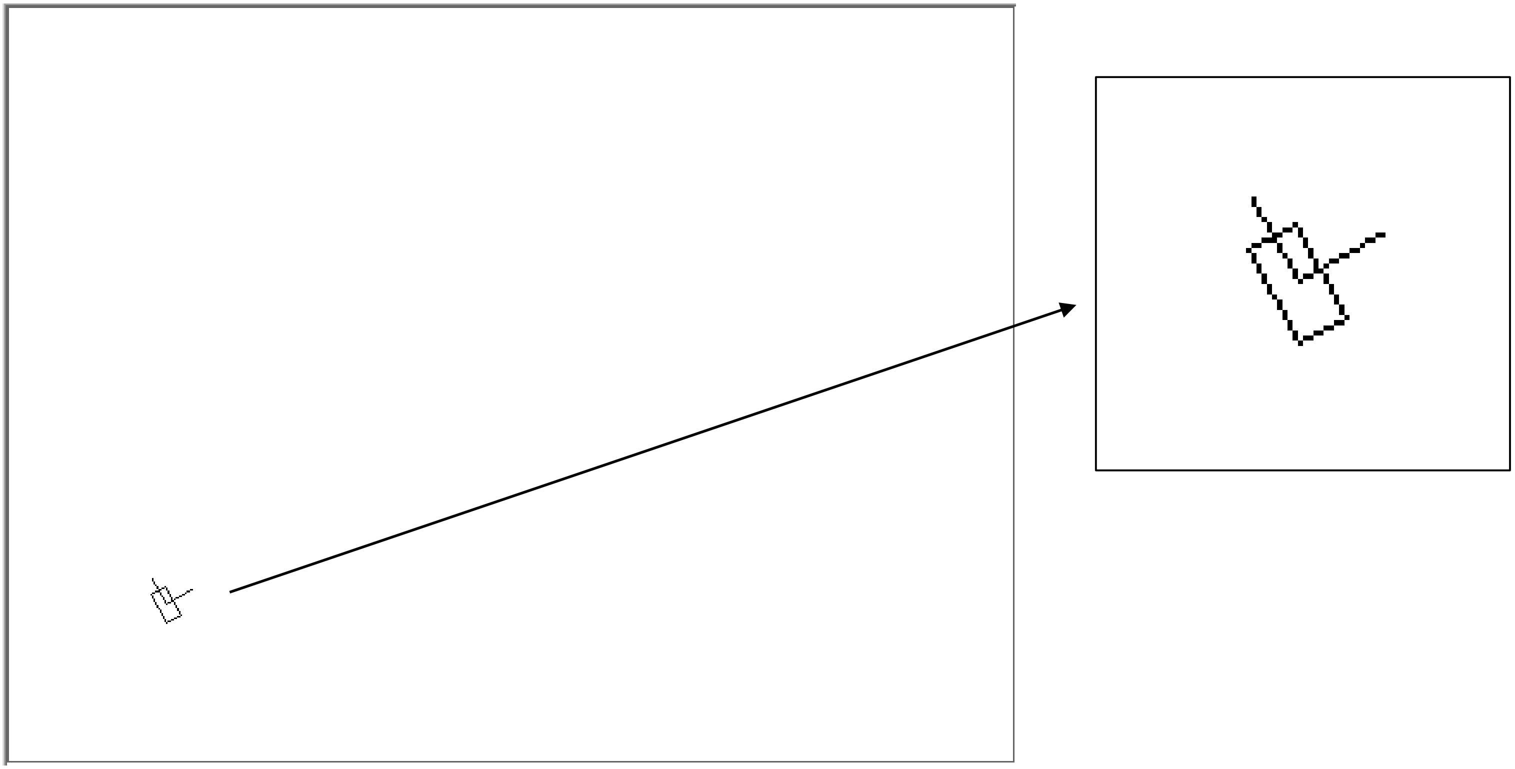
- It does NOT seem 30 Deg. Why?
  - + degree is defined as Counter Clock wise in general.



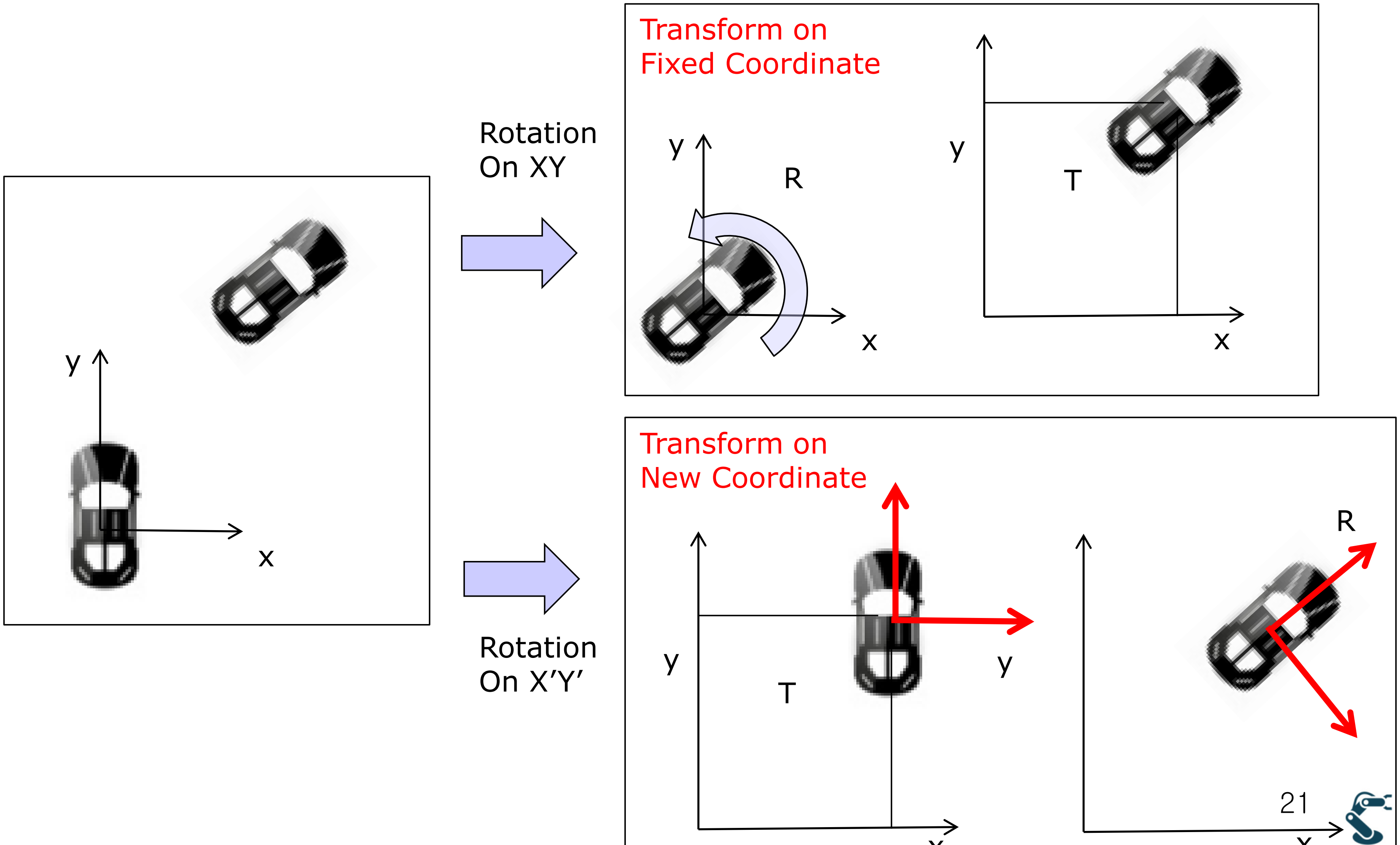


Demo: Reverse Y Axis and Design Object Class uObj

- Demo: uWnd-12-NewDesign
- You check how C++ class is implemented

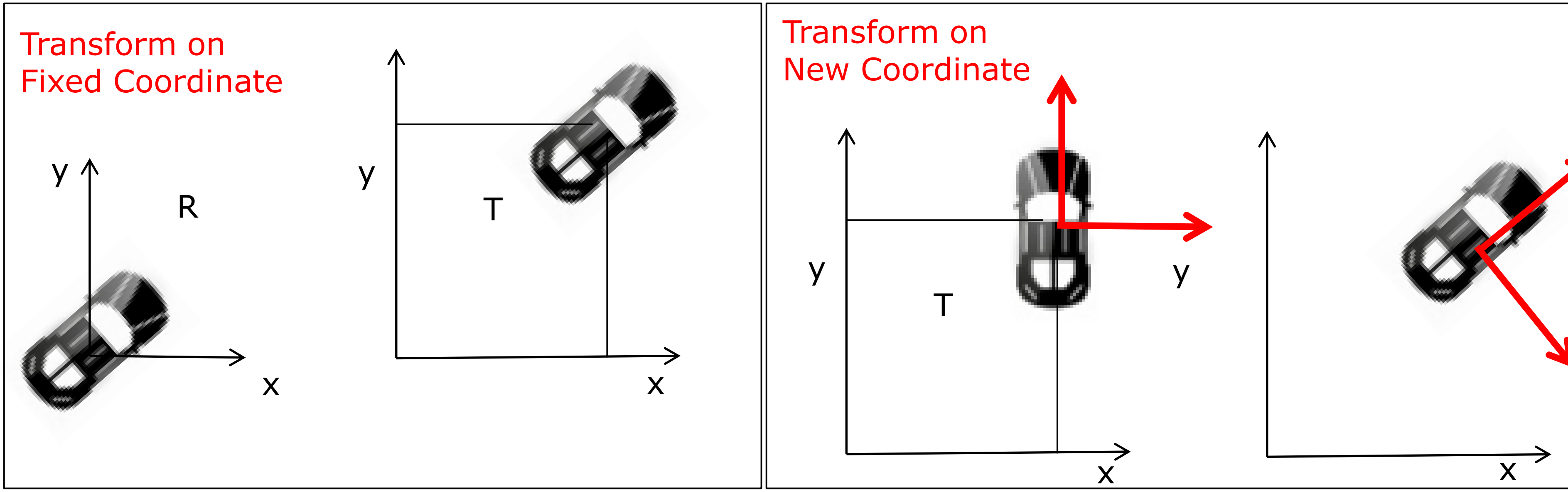


Coordinate Transform
Rotation, Translation



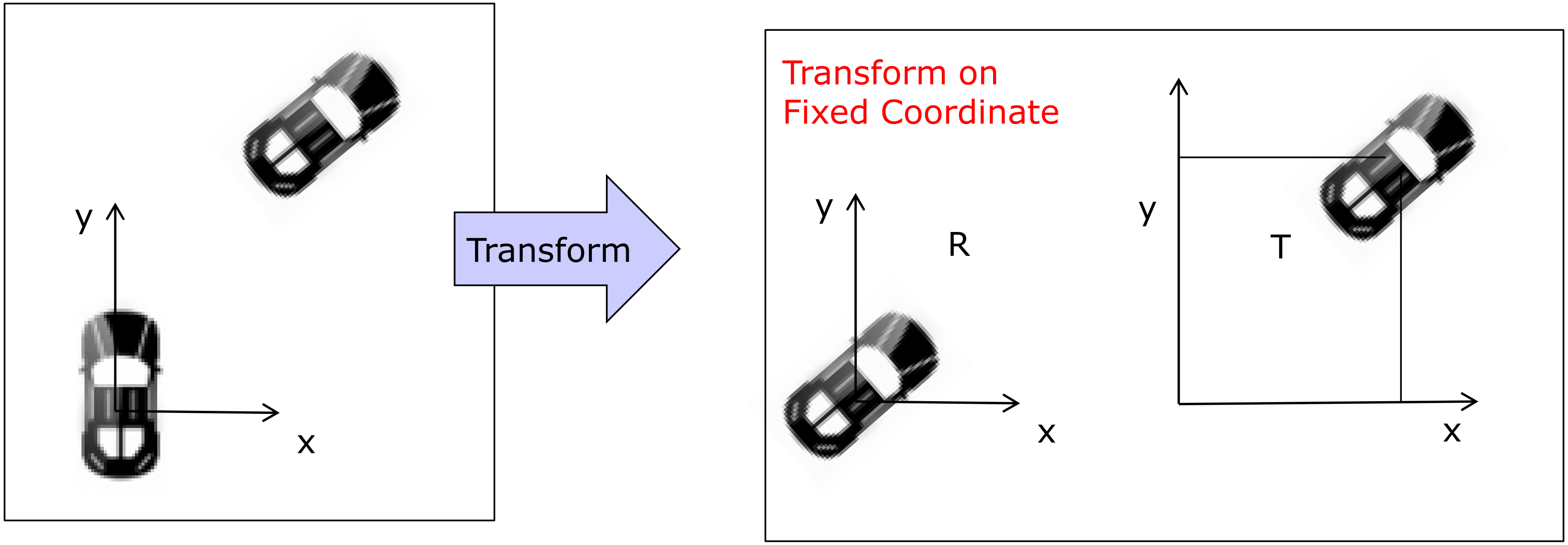
Two types
Fixed Coordinate Vs. Moving(New) Coordinate

- **Both cases are same**



- **Fixed coordinate is easy for Vector Calculation**

First Think Fixed Coordinate



$$v' = Rv + T$$

- v'= (R * v) + T
- **Here, a Good Idea is proposed…**

Homogeneous Transform

- Can we use Matrix Multiplication?

$$v' = Rv + T \Longrightarrow X' = HX$$

- Homogeneous Vector, X

$$X = \begin{pmatrix} v \\ 1 \end{pmatrix}$$

- Homogeneous Matrix, H

$$X' = \begin{pmatrix} v' \\ 1 \end{pmatrix} = \begin{pmatrix} Rv + T \\ 1 \end{pmatrix} = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ 1 \end{pmatrix} = H \begin{pmatrix} v \\ 1 \end{pmatrix} = HX$$

Homogeneous Vector and Matrix

$$X = \begin{bmatrix} v \\ 1 \end{bmatrix} \qquad H = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

- Because v=(x,y,z), Because R_{3x3}, T_{3x1},

$$X = \begin{bmatrix} v \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad H = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Matrix for ONLY Translation and ONLY Rotation

- General form with rotation and translation

$$H = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \qquad X = \begin{bmatrix} v \\ 1 \end{bmatrix}$$

- Translation

$$H = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \qquad I = I_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$X' = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v \\ 1 \end{bmatrix} = \begin{bmatrix} Iv + T \\ 1 \end{bmatrix} = \begin{bmatrix} v + T \\ 1 \end{bmatrix}$$

Rotation

$$H = \begin{bmatrix} R & O \\ 0 & 1 \end{bmatrix} \qquad O = O_{3\times1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$X' = \begin{bmatrix} R & O \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v \\ 1 \end{bmatrix} = \begin{bmatrix} Rv \\ 1 \end{bmatrix}$$

Go Back to Fixed or Moving(New) Coordinate

- Fixed coordinate is easy for Vector calculation

$$v' = Rv + T$$

- But In Matrix Calculation, Commutative Law is needed

$$A * B \neq B * A$$

- Rule of Matrix Multiplication in Fixed Coordinate

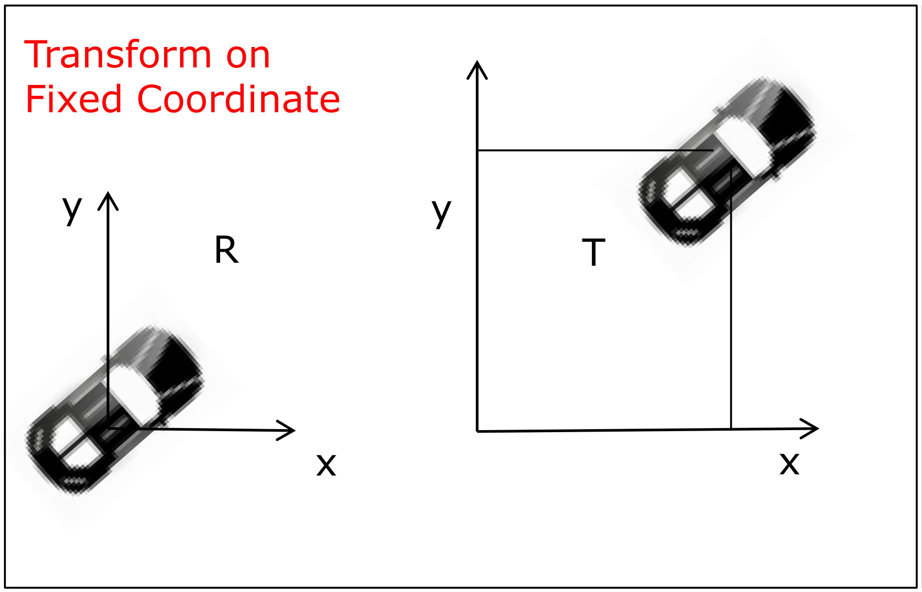
$$\longrightarrow H_{new} H_{old}$$

- Rule of Matrix Multiplication in New Coordinate

$$H_{old} H_{new} \longleftarrow$$

Proof of
Transform in Fixed and New coordinate

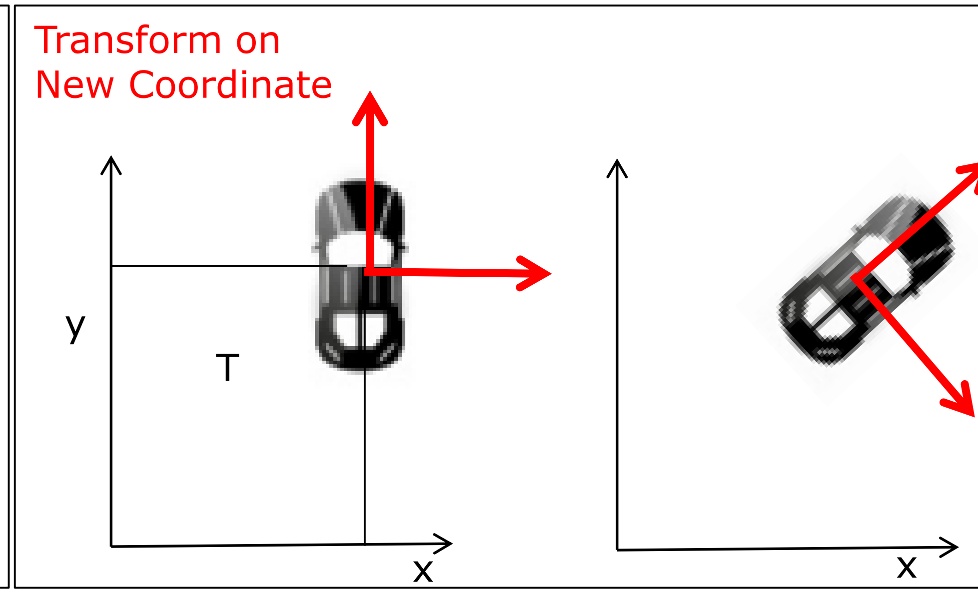
- Fixed coordinate



$$H_T \longrightarrow H_R$$

$$H = \underrightarrow{H_T H_R} = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & O \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

New coordinate



$$H_T \longleftarrow H_R$$

$$H = \underleftarrow{H_T H_R} = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & O \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

We define C++ Class for Homogeneous Transform

- uVector = (x,y,z)

- hVector = (uVector, 1) = (x,y,z,1)

- hMat = $\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ = v[16] $= \begin{bmatrix} \mathrm{v}[0] & \mathrm{v}[4] & \mathrm{v}[8] & \mathrm{v}[12] \\ \mathrm{v}[1] & \mathrm{v}[5] & \mathrm{v}[9] & \mathrm{v}[13] \\ \mathrm{v}[2] & \mathrm{v}[6] & \mathrm{v}[10] & \mathrm{v}[14] \\ \mathrm{v}[3] & \mathrm{v}[7] & \mathrm{v}[11] & \mathrm{v}[15] \end{bmatrix}$

Overload and Override in C++

- Example of Overloading
  - void test(float x, float y, float z)
  - void test(float x, float y)
  - void test(uVector)
  - Function arguments varies for other purposes.

- Overriding : Sub classing → Inheritance → OOP
  - See Overriding after 3D Object modeling.

Overloading of Homogeneous Matrix (hMat)

```
class hMat
{
public:
        hMat();
public:
        float v[16];
public:
        hMat    Trans(float x,float y,float z);
        hMat    Trans(uVector);
        hMat    Trans(hVector);
        hMat    RotX(float q);
        hMat    RotY(float q);
        hMat    RotZ(float q);
        hMat    operator*(hMat);
        hVector operator*(hVector);
        uVector operator*(uVector);
};
```

$$H = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix}$$

- $H = H_A * H_B$
- $\begin{bmatrix} v' \\ 1 \end{bmatrix} = H \begin{bmatrix} v \\ 1 \end{bmatrix}$
- $v = Hv$

See Example
Rotation and Translation

uWnd-12-NewDesign

```
uWnd::uWnd()
{
    car.q = 30;
    car.vertex[0]   = car.vertex[0].Rot(car.q);
    car.vertex[1]   = car.vertex[1].Rot(car.q);
    car.vertex[2]   = car.vertex[2].Rot(car.q);
    car.vertex[3]   = car.vertex[3].Rot(car.q);

    uVector t(100,100,0);
    car.vertex[0]   = car.vertex[0]+t;
    car.vertex[1]   = car.vertex[1]+t;
    car.vertex[2]   = car.vertex[2]+t;
    car.vertex[3]   = car.vertex[3]+t;
}
```

$$v' = Rv + T$$

uWnd-13-Homogeneous-Transform

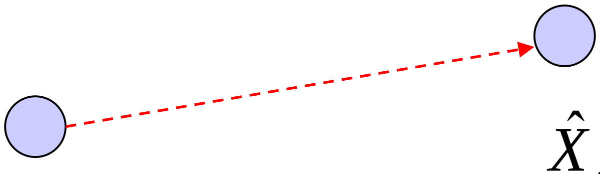
```
uWnd::uWnd()
{
    car.q = 30;
    uVector t(100,100,0);

    hMat h;
    h   = h.Trans(t)*h.RotZ(car.q);
    car.vertex[0]   = h*car.vertex[0];
    car.vertex[1]   = h*car.vertex[1];
    car.vertex[2]   = h*car.vertex[2];
    car.vertex[3]   = h*car.vertex[3];
}
```

$$H = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & O \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} v' \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ 1 \end{bmatrix} \qquad \therefore v' = Rv + T$$

Simple Example of Position Control (Proportional Control, P Control)

$\hat{X} = [x, y]$



$\hat{X}_d = [x_d, y_d]$ Desired Position

- How we control from X to Xd?
- Define a new Parameter, error, e = Xd – X

$$\hat{e} = [x_d - x, y_d - y] = \hat{X}_d - \hat{X}$$

- Update a new Position X' = X + K*e (0<K<1)

Repeat Until a condition

$$\begin{cases} \hat{e} = \hat{X}_d - \hat{X} \\ \hat{X} \leftarrow \hat{X} + K\hat{e} \\ \dots \end{cases}$$

$$\therefore \hat{X} \rightarrow \hat{X}_d$$

Example) uWnd-14-Control-Missile

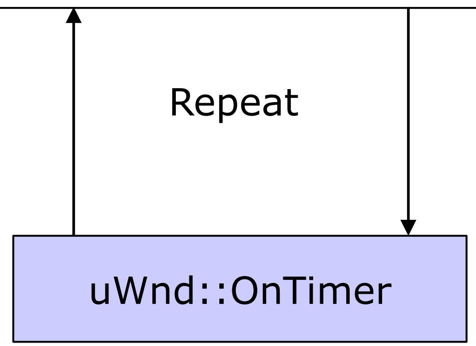
```
void uWnd::Run()
{
    uVector o   = car.Center();
    uVector e   = target-o;

    e= e*0.1;
    car.vertex[0]   = car.vertex[0]+e;
    car.vertex[1]   = car.vertex[1]+e;
    car.vertex[2]   = car.vertex[2]+e;
    car.vertex[3]   = car.vertex[3]+e;
    Redraw();
}
```

$$\hat{e} = \hat{X}_d - \hat{X}$$

$$\hat{X} \leftarrow \hat{X} + K\hat{e}$$

$$\cdots$$

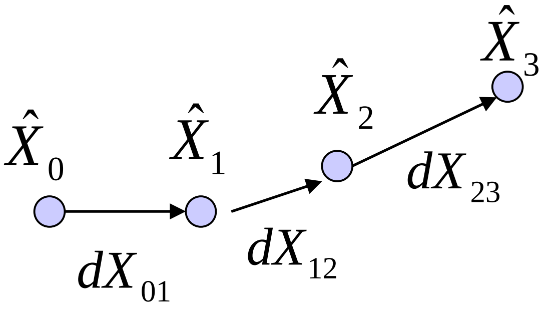
$$\therefore \hat{X} \rightarrow \hat{X}_d$$



- This example moves to clicked positions
- We need to change the angle of a missile

Relative or General Transform

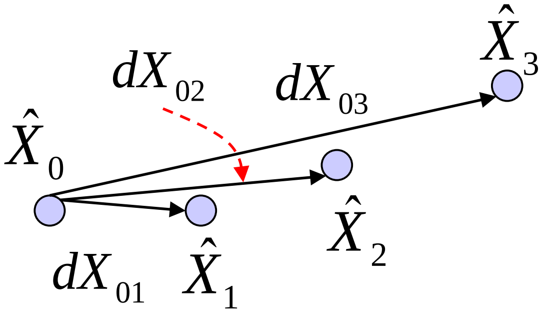
Relative Transform



$$\hat{X}_{i+1} = \hat{X}_i + d\hat{X}_{i,i+1}$$

- Simple
- Vertex is updated in each turn.

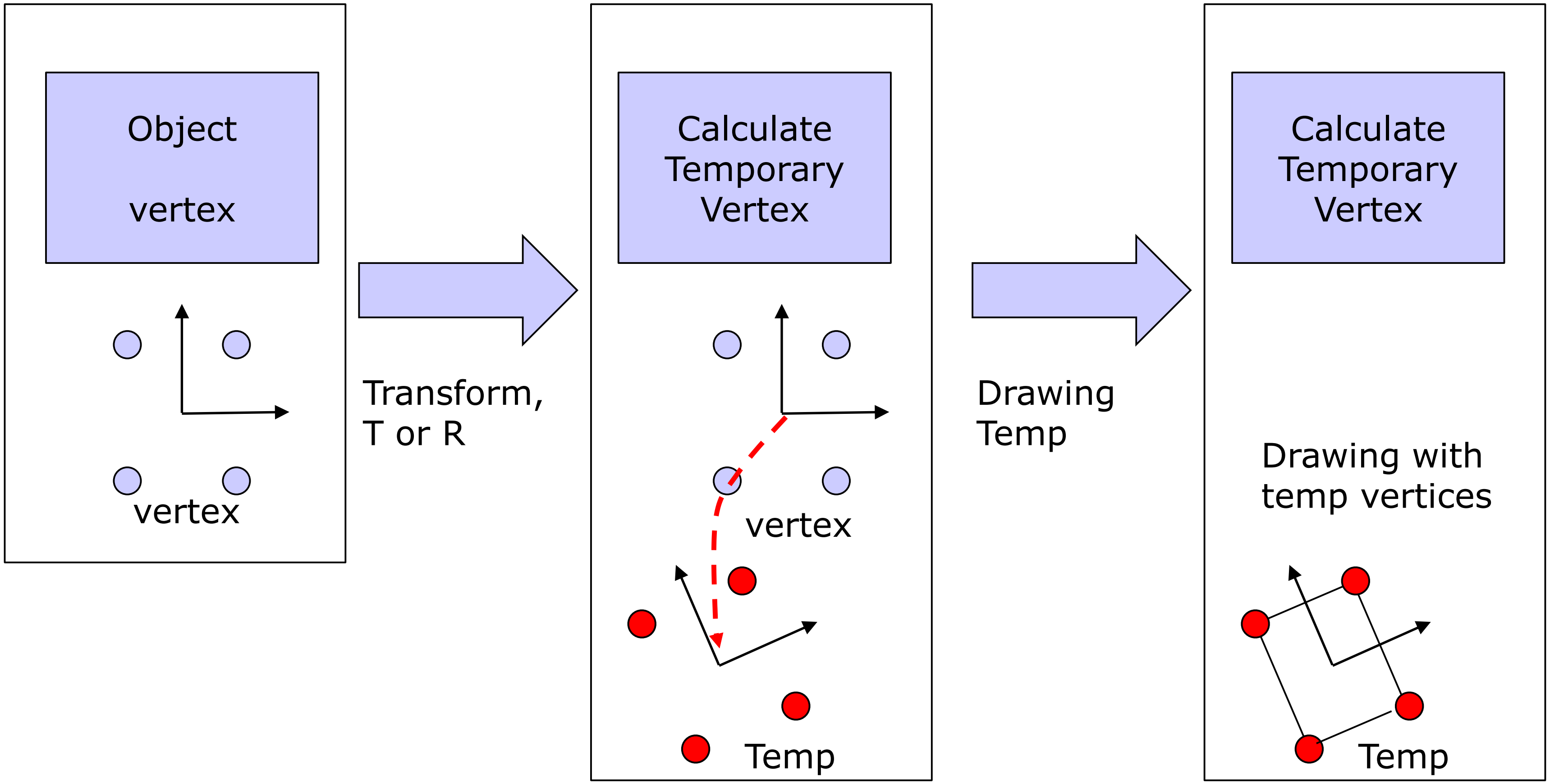
General Transform



$$\hat{X}_{i+1} = \hat{X}_0 + d\hat{X}_{0,i+1}$$

- Transform must be thought from ORIGIN.
- Good for CAD and Graphics

Diagram of General Transform



General Transform
example) uWnd-15-Control-Missile-Temp

```
class uObj
{
public:
    uObj();
public:
    void    Draw(CDC*);
    uVector Center();
protected:
    void    DrawAxis(CDC*);

public:
    float q;
    uVector vertex[4];  // Original data
    uVector temp[4];    // Drawing data

};
```

```
void uWnd::Run()
{
    uVector o    = car.Center();
    uVector e    = target-o;

    e= e*0.1;
    uVector t    = o + e;
    car.temp[0] = car.vertex[0]+t;
    car.temp[1] = car.vertex[1]+t;
    car.temp[2] = car.vertex[2]+t;
    car.temp[3] = car.vertex[3]+t;
    Redraw();
}
```

```
void uObj::Draw(CDC *pDC)
{
    DrawAxis(pDC);

    for (int i=0;i<4;i++)
    if  (i==0)    pDC->MoveTo(temp[i].x,temp[i].y);
    else          pDC->LineTo(temp[i].x,temp[i].y);
    pDC->LineTo(temp[0].x,temp[0].y);
}
```

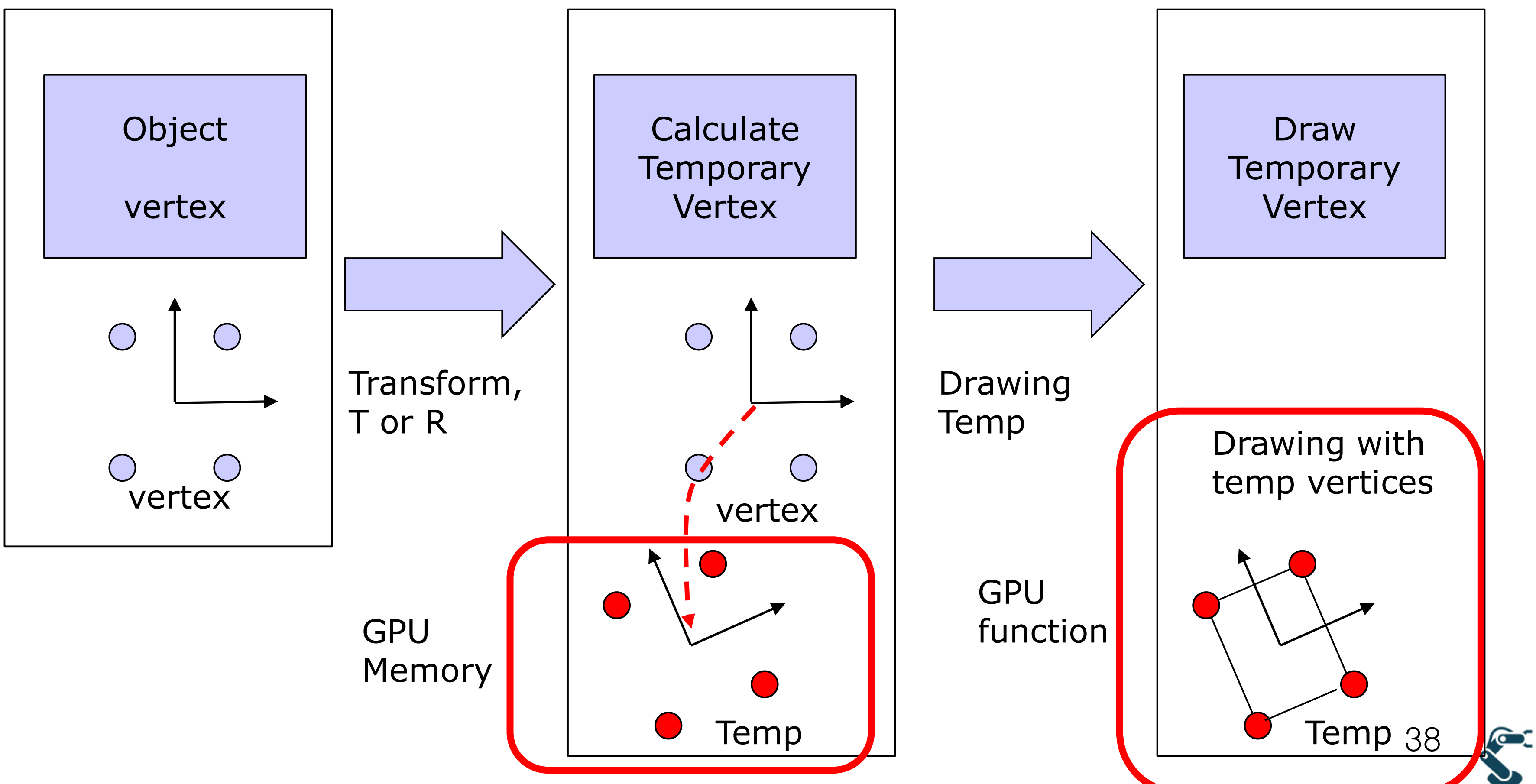
$$\hat{X}_{temp} = \hat{X}_{i+1} = \hat{X}_{0} + d\hat{X}_{0,i+1}$$

Translation, t = o(origin) + e(error)
Temp = vertex + t

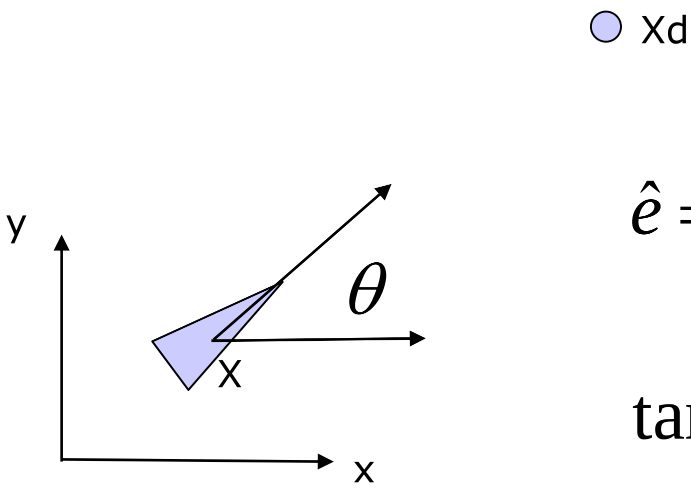
- car.vertex is NOT updated.
- car.temp is updated.

GPU generates temporary vertices.

- Why my GPU has a lack of memory?



How to Calculate Missile's Heading Angle, θ uWnd-16-Control-Missile-Angle



$$\hat{e} = \hat{X}_d - \hat{X} = [e_x, e_y, 0]$$

$$\tan\theta = \frac{y}{x} \quad \left(= \frac{e_y}{e_x} \right)$$

$$\therefore \theta = \tan^{-1}\frac{y}{x}$$

$$\therefore \theta = \mathrm{atan}(y, x) \qquad 0 \le \theta \le 2\pi$$

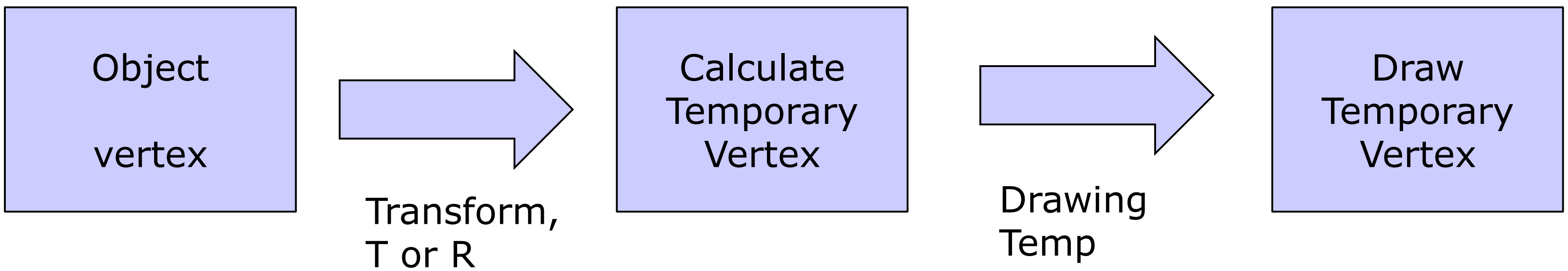
$$\therefore \theta = \mathrm{atan2}(y, x) \qquad -\pi \le \theta \le \pi$$

- Declare a new function at uVector

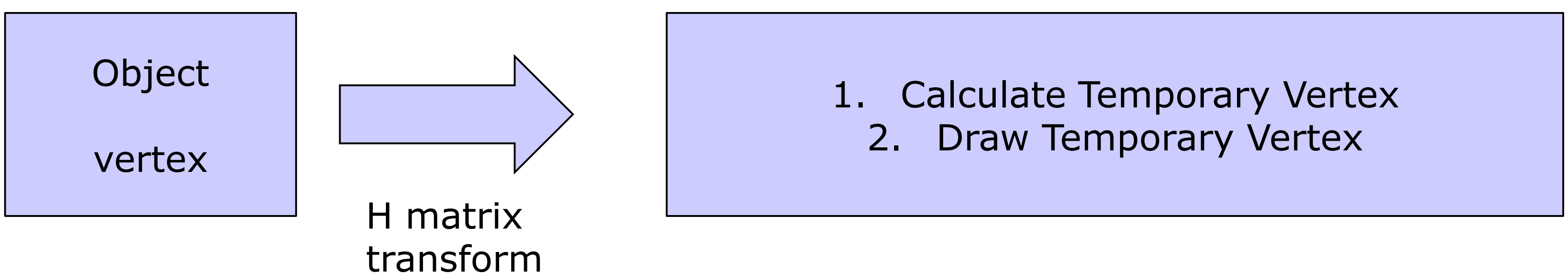
```
float uVector::Angle()
{
    float f = atan2(y,x);
    return DEG(f);
}
```

We will Define Object like, uWnd-17-Control-Missile-Object-Complete

- uWnd-17-control-missile-object-complete changes,



uWnd-15-Control-Missile-Temp



uWnd-17-Control-Missile-Object-Complete

uWnd-17-Control-Missile-Object-Complete

```
class uObj
{
public:
    uObj();
public:
    void    Draw(CDC*);
    uVector Center();
protected:
    void    DrawAxis(CDC*);

public:
    float q;
    uVector vertex[4];  // Original data
    uVector temp[4];    // Drawing data

};
```

uWnd-15-Control-Missile-Temp

```
class uObj
{
public:
    uObj();
public:
    void    Draw(CDC*);

protected:
    void    DrawAxis(CDC*);

public:
    // Transform
    hMat    H;
    float   q;

    // original data
    int     nMax;
    uVector vertex[4];
};
```

uWnd-17-Control-Missile-Object-Complete

```
void uWnd::Run()
{
    uVector o   = car.Center();
    uVector e   = target-o;

    e= e*0.1;
    uVector t   = o + e;
    car.temp[0] = car.vertex[0]+t;
    car.temp[1] = car.vertex[1]+t;
    car.temp[2] = car.vertex[2]+t;
    car.temp[3] = car.vertex[3]+t;
    Redraw();
}
```

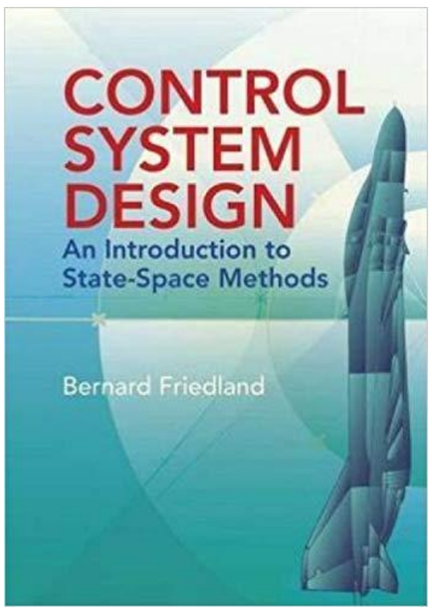
```
void uWnd::Run()
{
    uVector o   = car.H.O();
    uVector e   = target-o;
    e= e*0.1;

    float q = e.Angle()-90;
    car.q   = q;

    uVector t   = o+ e;

    hMat h;
    car.H   = h.Trans(t)*h.RotZ(q);
    Redraw();
}
```

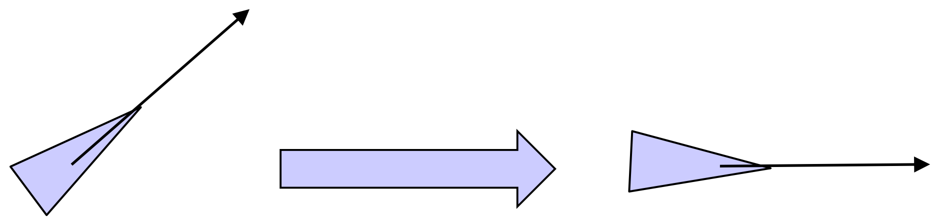
```
void uObj::Draw(CDC *pDC)
{
    DrawAxis(pDC);

    for (int i=0;i<4;i++)
    if (i==0)    pDC->MoveTo(temp[i].x,temp[i].y);
    else         pDC->LineTo(temp[i].x,temp[i].y);
    pDC->LineTo(temp[0].x,temp[0].y);
}
```

```
void uObj::Draw(CDC *pDC)
{
    DrawAxis(pDC);

    int i;
    uVector temp[4];

    // transform vertex
    for (i=0;i<4;i++)
        temp[i] = H*vertex[i];

    // draw
    for (i=0;i<4;i++)
    if (i==0)    pDC->MoveTo(temp[i].x,temp[i].y);
    else         pDC->LineTo(temp[i].x,temp[i].y);
    pDC->LineTo(temp[0].x,temp[0].y);
}
```

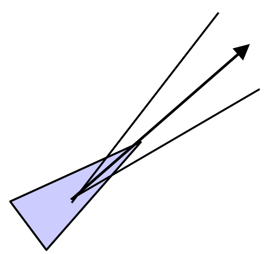
uWnd-18-Real-Missile

- What is the Difference?
- Real Guided Missile CANNOT rotate fast.





Fast rotation is impossible in our physical world



Limitation of Heading angle change