Computer Graphics and Programming

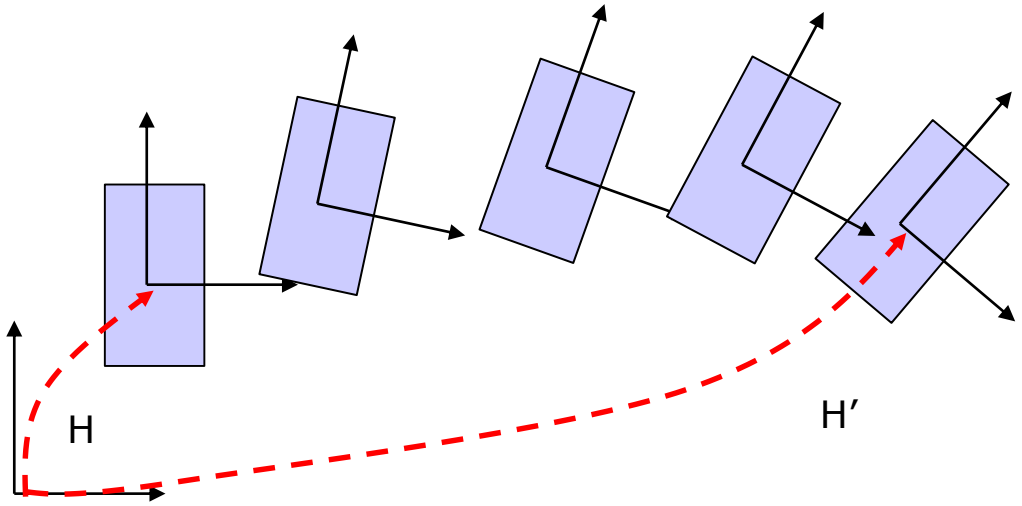
Moving Coordinate(Animation)
Lecture 6

Jeong-Yean Yang

2020/10/22

1 Moving Coordinate Transform

Smooth Coordinate Transform



- Continuous Transformation from H to H'
  - Translation

$$X \rightarrow X_d$$

$$e = X_d - X$$

$$X' = X + Ke \quad (0 < K < 1)$$

Rotation

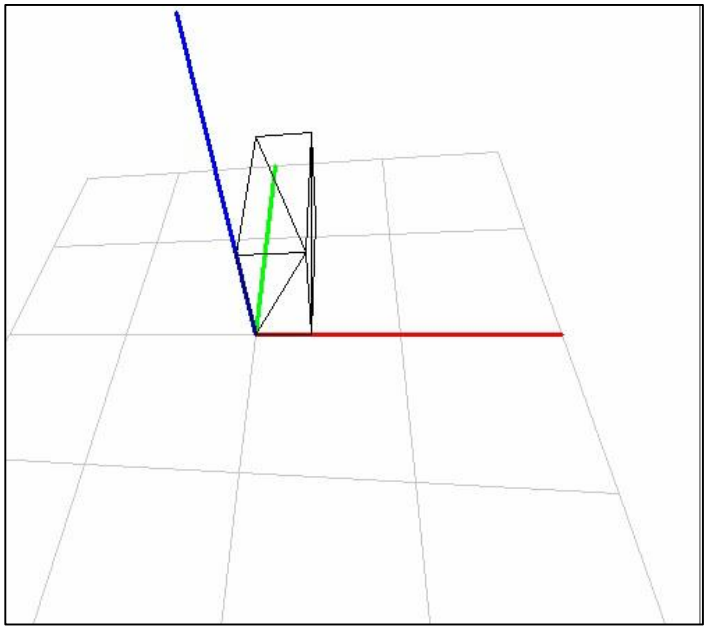
$$\Theta \rightarrow \Theta_d$$

$$e = \Theta_d - \Theta$$

$$\Theta' = \Theta + Ke \quad (0 < K < 1)$$

Ex) uWnd-43-MC-Translation

Trans (0,0,0)→(3,0,0)



$$X \rightarrow X_d$$

$$e = X_d - X$$

$$X' = X + Ke \quad (\ 0 < K < 1)$$

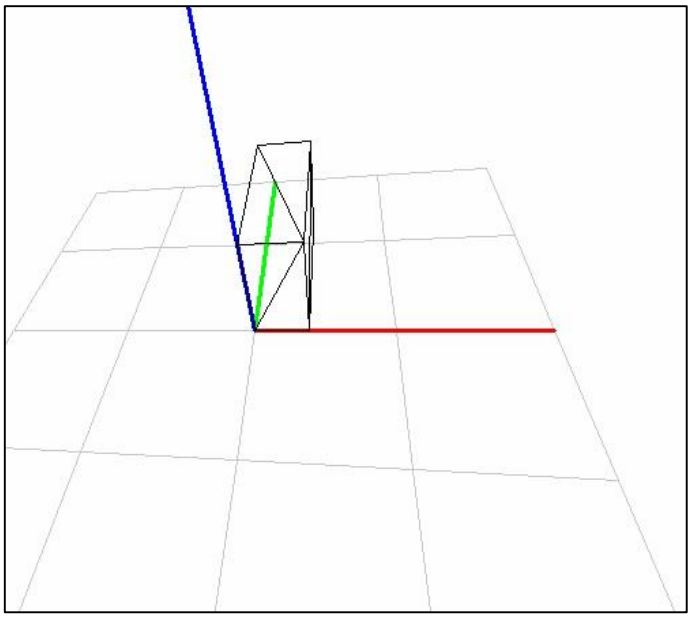
```
box.MakeBox(1,2,3);
Xd   = uVector(0,0,0);
```

```
void uWnd::Run()
{
    hMat h;

    // Run Box Movement
    uVector X    = box.H.C();
    uVector e    = Xd-X;
    uVector dx   = e*0.1;

    if (e.Norm()>0.0001)
    {
        box.H    = box.H.Trans(X+dx);
        bRedraw = TRUE;
    }
```

```
void uWnd::OnKeyDown(UINT nChar, UINT nRepCnt,
{
    switch(nChar){
    case '1':   Xd  = uVector(3,0,0);   break;
    case '2':   Xd  = uVector(0,0,0);   break;
    }
```

Ex) uWnd-44-MC-Rotation

Rot (0,0,0) → (30,50,0)



$$\Theta \rightarrow \Theta_d$$

$$e = \Theta_d - \Theta$$

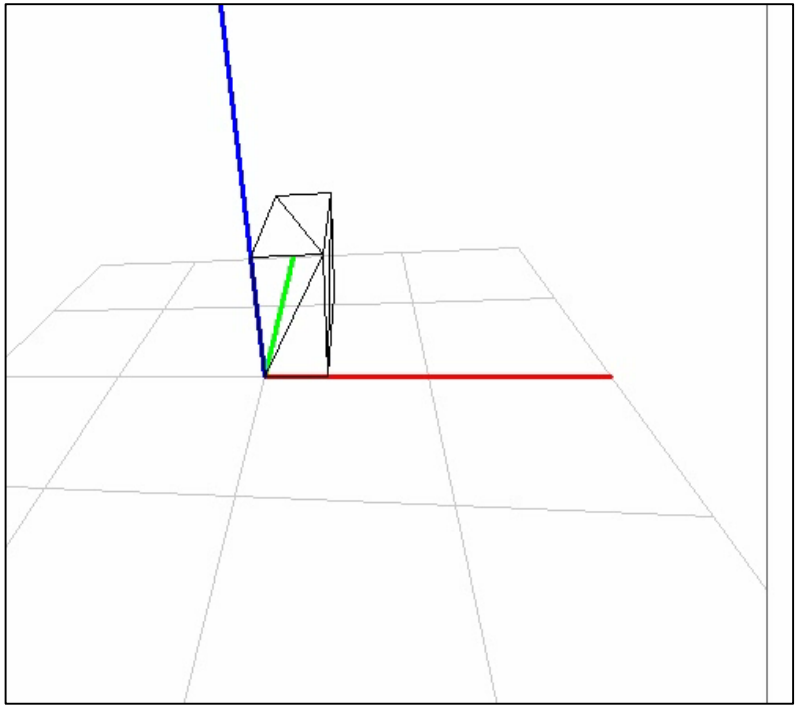
$$\Theta' = \Theta + Ke \quad (\ 0 < K < 1)$$

```
box.MakeBox(1,2,3);
qd  = uVector(0,0,0);
```

```
void uWnd::Run()
{
    hMat h;

    // Run Box Movement
    uVector q   = box.q;
    uVector e   = qd-q;
    uVector dq  = e*0.1;

    if (e.Norm()>0.0001)
    {
        q       = q+dq;
        box.H   = box.H.RotX(q.x) * box.H.RotY(q.y);
        box.q   = q;
        bRedraw = TRUE;
    }
```

```
void uWnd::OnKeyDown(UINT nChar, UINT nRepCnt,
{
    switch(nChar){
    case '1':   qd  = uVector(30,50,0); break;
    case '2':   qd  = uVector(0,0,0);   break;
    }
```


Ex) Translation + Rotation



$$X \rightarrow X_d$$
$$e = X_d - X$$
$$X' = X + Ke \rightarrow H_{Trans}$$

$$\Theta \rightarrow \Theta_d$$
$$e = \Theta_d - \Theta$$
$$\Theta' = \Theta + Ke \rightarrow H_{Rot}$$

$$H' = H_{Trans} H_{Rot}$$

- Translation and Rotation for Continuous Transform in every step.

2 Quaternion as Homogeneous Transform

Quaternion for Homogeneous Transform New classs, hQuat

- Matrix Multiplication is easier that Vector calculus

$$\hat{v}' = \cos\theta\hat{v} + (1-\cos\theta)(\hat{v}\cdot\hat{u})\hat{u} + \sin\theta\left(\hat{u}\times\hat{v}\right)$$

- hQuat has the feature of Quaternion

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

$$H = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) & 0 \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) & 0 \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\hat{v}' = H\hat{v}$$

Ex) uWnd-46-MC-Quaternion hMat.h

```
class uVector;
class hMat;
class hQuat
{
public:
        hQuat();
        hQuat(float,uVector);
        hQuat(float,float,float,float);
public:
        float   q,x,y,z;
        hQuat   Unit();
        hMat    operator*(hMat);
};
```

p.14, Quaternion

$$if\ \ q=(s,\hat{u})=\left(\cos\frac{\theta}{2}+\sin\frac{\theta}{2}\hat{u}\right),\ \ |\hat{u}|=1$$

```
hQuat::hQuat(float q,uVector v)
{
    q       = RAD(q);
    this->q = cos(q/2);
    uVector u   = v.Unit()*sin(q/2);
    x   = u.x;
    y   = u.y;
    z   = u.z;
}
```

hQuat(angle,axis)

hQuat(angle,q_1, q_2, q_3)

hMat operator*(hMat) →

```
hMat h;
hQuat q;
h = q*h;
```

hQuat(Quaternion) → hMat(H matrix)

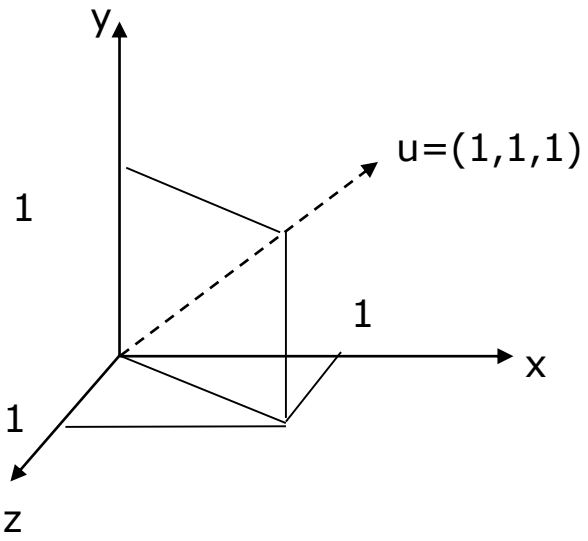
```
class hMat
{
public:
        hMat();
        hMat(hQuat);
public:
        float v[16];
public:
        void    I();
        hMat    Scale(float);
        hMat    Trans(float x,float y
        hMat    Trans(uVector);
        hMat    Trans(hVector);
        hMat    RotX(float q);
        hMat    RotY(float q);
        hMat    RotZ(float q);
        hMat    operator*(hMat);
        hVector operator*(hVector);
        uVector operator*(uVector);
        void    operator=(uVector);
        void    operator=(hQuat);

        uVector O();
};
```

$$H = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) & 0 \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) & 0 \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
void hMat::operator =(hQuat q)
{
    float qs    = q.q*q.q;
    float xs    = q.x*q.x;
    float ys    = q.y*q.y;
    float zs    = q.z*q.z;

    v[0]    = qs+xs-ys-zs;
    v[1]    = 2*(q.x*q.y+q.q*q.z);
    v[2]    = 2*(q.x*q.z-q.q*q.y);

    v[4]    = 2*(q.x*q.y-q.q*q.z);
    v[5]    = qs-xs+ys-zs;
    v[6]    = 2*(q.y*q.z+q.q*q.x);

    v[8]    = 2*(q.x*q.z+q.q*q.y);
    v[9]    = 2*(q.y*q.z-q.q*q.x);
    v[10]   = qs-xs-ys+zs;
}
```

Ex)uWnd-46-MC-Quaternion

```
void uWnd::Run()
{
    hQuat q(box.q.x, uVector(1,1,1));
    box.q.x+=1;

    box.H    = q;
    bRedraw = TRUE;

    if (bRedraw)
    {
        axis.Update();
        ground.Update();
        box.Update();

        Redraw();
        bRedraw = FALSE;
    }
}
```



- Rotation along (1,1,1) vector is very Easy.
- See Result of HW 10.

3 Inverse Homogeneous Transform

Back to Definition of H

$$H = \begin{bmatrix} R & T \\ P & 1 \end{bmatrix} \qquad X = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

$$X' = HX = \begin{bmatrix} R & T \\ P & 1 \end{bmatrix} X = \begin{bmatrix} R & T \\ P & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} Rx + T \\ Px + 1 \end{bmatrix}$$

- R: Rotation
- T: Translation
- 1: homogeneous factor (Distance)
- P: projection → Used in 3D vision and Graphics.
- → Not for Robotics.

Inverse of Homogeneous Transform

$$H = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \qquad HH^{-1} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} I_{3\times3} & 0 \\ 0 & 1 \end{bmatrix} = I_{4\times4}$$

$$HH^{-1} = \begin{bmatrix} Ra+Tc & Rb+Td \\ c & d \end{bmatrix} = I$$

$$c = 0, d = 1$$

$$HH^{-1} = \begin{bmatrix} R_{3\times3}a_{3\times3} & R_{3\times3}b_{3\times1}+T_{3\times1} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} I_{3\times3} & 0 \\ 0 & 1 \end{bmatrix}$$

1 $R_{3\times3}a_{3\times3} = I_{3\times3}$

2 $R_{3\times3}b_{3\times1} + T_{3\times1} = 0_{3\times1}$

Inverse of Rotation Matrix

$$R_{3\times3}a_{3\times3} = I_{3\times3}$$ **Remind rotation matrix**

$$R^{-1} = a_{3\times3}$$

$$R_x(\theta)^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^{-1} = R_x(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = R_x(\theta)^T$$

$$\therefore R^{-1} = R^T$$

- When inverse matrix is equal to Transpose,
  - It is called “Orthogonal transform”

Inverse of Homogeneous Transform

$$R_{3\times3}b_{3\times1} + T_{3\times1} = 0_{3\times1}$$

$$R_{3\times3}b_{3\times1} = -T_{3\times1}$$

$$\therefore b_{3\times1} = -R_{3\times3}{}^{-1}T_{3\times1} = -R_{3\times3}{}^{T}T_{3\times1}$$

$$H = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \quad \rightarrow \quad H^{-1} = \begin{bmatrix} R^T & -R^T T \\ 0 & 1 \end{bmatrix}$$

Inverse Homogeneous Transform
hMat::Inv()

```
hMat hMat::Inv()
{
    hMat ret;
    ret.v[0]    = v[0];
    ret.v[1]    = v[4];
    ret.v[2]    = v[8];
    ret.v[4]    = v[1];
    ret.v[5]    = v[5];
    ret.v[6]    = v[9];
    ret.v[8]    = v[2];
    ret.v[9]    = v[6];
    ret.v[10]   = v[10];

    uVector pos = 0()*-1;
    pos = ret*pos;

    ret.v[12]   = pos.x;
    ret.v[13]   = pos.y;
    ret.v[14]   = pos.z;
    return ret;
}
```

R^T

$-T$

$-R^T T$

$$H^{-1} = \begin{bmatrix} R^T & -R^T T \\ 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} v[0] & v[4] & v[8] & v[12] \\ v[1] & v[5] & v[9] & v[13] \\ v[2] & v[6] & v[10] & v[14] \\ v[3] & v[7] & v[11] & v[15] \end{bmatrix}$$

$$R = \begin{bmatrix} v[0] & v[4] & v[8] \\ v[1] & v[5] & v[9] \\ v[2] & v[6] & v[10] \end{bmatrix}, R^T = \begin{bmatrix} v[0] & v[1] & v[2] \\ v[4] & v[5] & v[6] \\ v[8] & v[9] & v[10] \end{bmatrix}$$

Inverse Trans(x,y,z) = Trans(-x,-y,-z)
ex) uWnd-48-MC-Inv

Remind pivot,
Ch5. pp. 29

$$H = H_{Trans} H_{Rot} H_{-Trans}$$

$$= H_{Trans} H_{Rot} H_{Trans}^{-1} = H_{-Trans}^{-1} H_{Rot} H_{-Trans}$$

```
if (1)
{
    //case 1: Pivot Trans(x,0,0)*RotZ()*Trans(-x,0,0);
    box.H   = h.Trans(0.5,0,0)*h.RotZ(box.q.z)*h.Trans(-0.5,0,0);
    box.q.z+=1;
    bRedraw = TRUE;
}
else
{
    //case 2: Pivot inv(T)*RotZ()*T;
    hMat t  = h.Trans(-0.5,0,0);
    box.H   = t.Inv()*h.RotZ(box.q.z)*t;
    box.q.z+=1;
    bRedraw = TRUE;
}
```

- Inverse transform is useful for complex multiplication

$$case1) H_{pivot} = H_{trans}(\mathrm{x},\mathrm{y},\mathrm{z})\,\mathrm{RotZ(q)}\,\mathrm{H}_{trans}(-x,-y,-z)$$

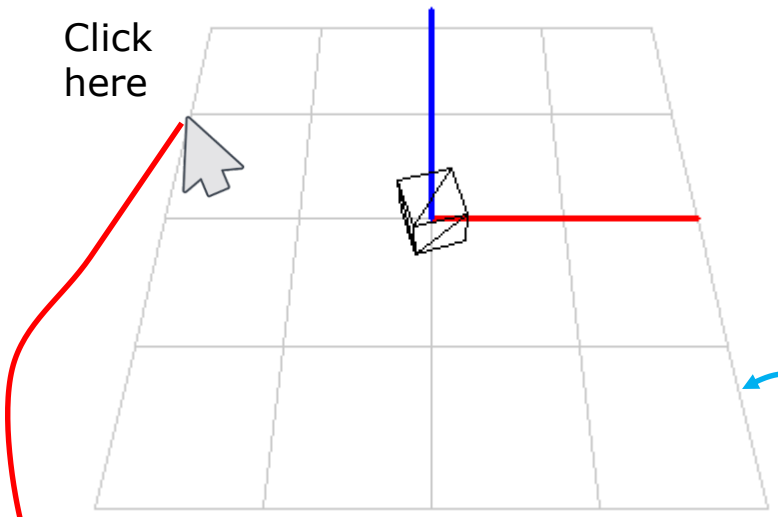
$$case2) H_{pivot} = \mathrm{H}_{trans}^{-1}(-x,-y,-z)\,\mathrm{RotZ(q)}\,\mathrm{H}_{trans}(-x,-y,-z)$$

4

Inverse Perspective Matrix

One of the Most Advanced Methods in 3D Graphics

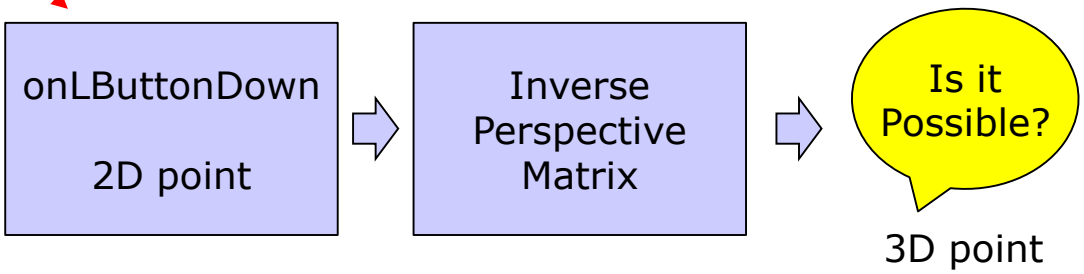
Can We Get 3D position by clicking 2D?



- From 2D into 3D?

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \in R^2 \xrightarrow[???]{} \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} \in R^3$$

- Ans: Extra Information is required.
  - Here, z = 0 plane.



Inverse Perspective Matrix

Normalized
2D vector

$$v_p = Pv = \begin{pmatrix} \frac{\cot(\alpha/2)}{W/H} & 0 & 0 & 0 \\ 0 & \cot(\alpha/2) & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{\cot(\alpha/2)}{W/H} x \\ \cot(\alpha/2) y \\ \frac{n+f}{n-f} z + \frac{2nf}{n-f} \\ -z \end{pmatrix} = \begin{pmatrix} \frac{\cot(\alpha/2)}{W/H} \frac{x}{-z} \\ \cot(\alpha/2) \frac{y}{-z} \\ -\frac{n+f}{n-f} + \frac{2nf}{n-f} \frac{1}{-z} \\ 1 \end{pmatrix}$$

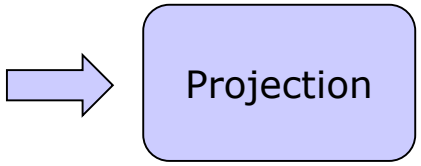
$$v = P^{-1} v_p = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{W/H}{\cot(\alpha/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\cot(\alpha/2)} & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{n-f}{2nf} & \frac{n+f}{2nf} \end{pmatrix} \begin{pmatrix} \frac{\cot(\alpha/2)}{W/H} x \\ \cot(\alpha/2) y \\ \frac{n+f}{n-f} z + \frac{2nf}{n-f} \\ -z \end{pmatrix}$$

Inverse Perspective Matrix

Remind Projection Process

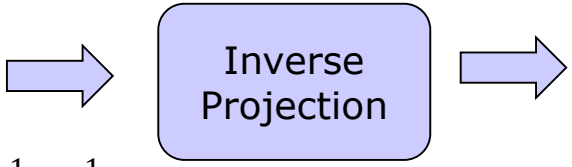
H: T, R for camera walk

$$v' = Hv$$
$$v_p = Pv'$$
$$v_g = Sv_p$$




```
// projection from vertex(3d) to temp(2d)
for (i=0;i<nVer;i++)
{
    pTemp[i]    = Hg*H*pVer[i];
    pTemp[i]    = pCam->Projection(pTemp[i]);
}
```

$$v_p = S^{-1}v_g$$
$$v' = P^{-1}v_p$$
$$v = H^{-1}v' = H^{-1}P^{-1}v_p$$



Process 0) $v_p = S^{-1}v_g$

Process 1) $v' = P^{-1}v_p$

Process2) $v = H^{-1}v'$

- Process 0: Scaling [-1, 1] space into [640,480]

Process 1

Process 1) $v' = P^{-1}v_p$

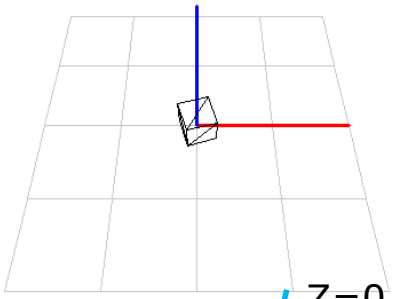
$$v' = P^{-1}v_p = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{W/H}{\cot(\alpha/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\cot(\alpha/2)} & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{n-f}{2nf} & \frac{n+f}{2nf} \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ 0 \\ h \end{pmatrix} = \begin{pmatrix} \frac{W/H}{\cot(\alpha/2)} x_p \\ \frac{1}{\cot(\alpha/2)} y_p \\ -h \\ \frac{n+f}{2nf} h \end{pmatrix}$$

- Click information only has 2Dim. Position, (x_p, y_p)
  - Assume that 'h' is needed by next calculation
  - **Assume that Zp is zero(It will be compensated later)**

Process 2

Process 1) $v' = P^{-1}v_p$

Process2) $v = H^{-1}v'$



$$v = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{H^{-1}} \begin{pmatrix} \frac{W/H}{\cot(\alpha/2)} x_p \\ \frac{1}{\cot(\alpha/2)} y_p \\ -h \\ \frac{n+f}{2nf} h \end{pmatrix} = \underbrace{\begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{H^{-1}} \begin{pmatrix} t_x \\ t_y \\ t_z \\ \frac{n+f}{2nf} h \end{pmatrix}$$

Unknown value

$$= \begin{pmatrix} \cdots \\ \cdots \\ h_{31}t_x + h_{32}t_y + h_{33}t_z + h_{34}\frac{n+f}{2nf}h \\ \frac{n+f}{2nf}h \end{pmatrix} = \begin{pmatrix} \cdots \\ \cdots \\ \frac{h_{31}t_x + h_{32}t_y + h_{33}t_z}{\frac{n+f}{2nf}h} + h_{34} \\ 1 \end{pmatrix} \Rightarrow \quad \therefore \frac{n+f}{2nf}h = \frac{h_{31}t_x + h_{32}t_y + h_{33}t_z}{\text{z-h}_{34}}$$

$$\therefore z2 = \frac{n+f}{2nf}h = \frac{h_{31}t_x + h_{32}t_y + h_{33}t_z}{\text{z-h}_{34}}$$

$$v = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = H^{-1} \begin{pmatrix} t_x \\ t_y \\ t_z \\ \frac{n+f}{2nf}h \end{pmatrix} = H^{-1} \begin{pmatrix} t_x / z2 \\ t_y / z2 \\ t_z / z2 \\ 1 \end{pmatrix}$$

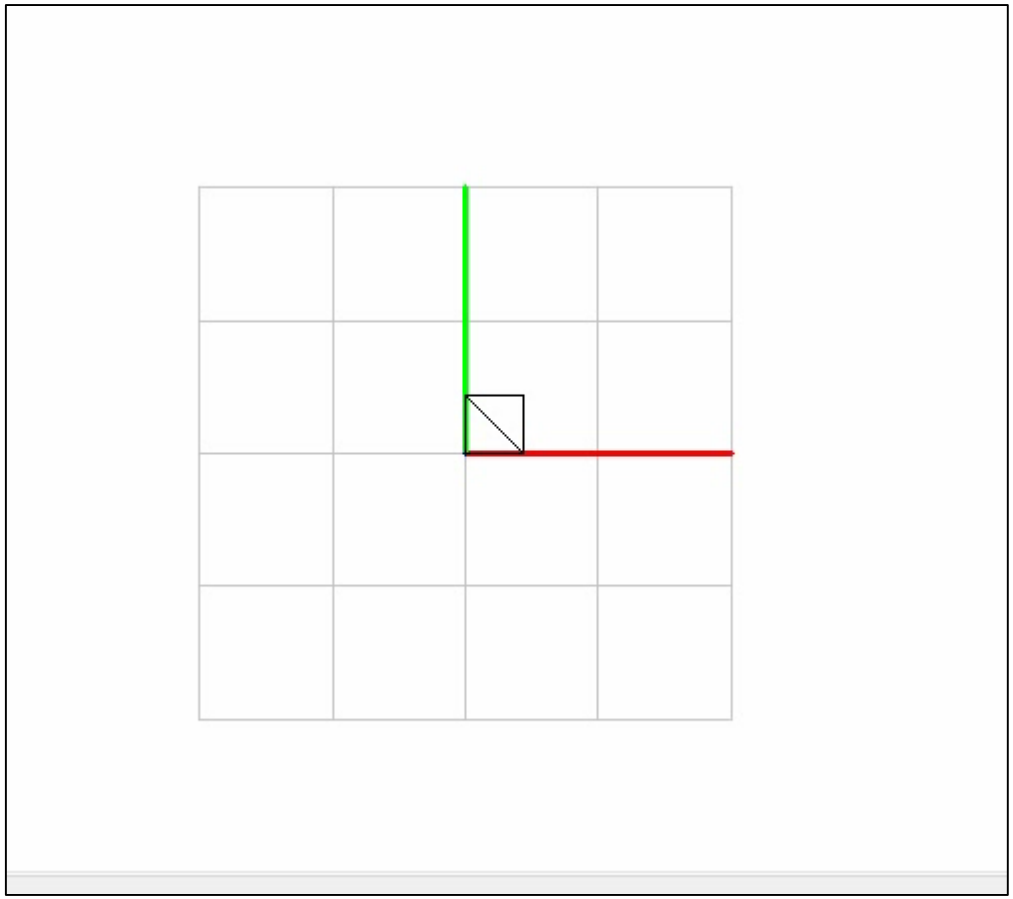
$$\therefore v = H^{-1} \begin{pmatrix} t_x / z2 \\ t_y / z2 \\ t_z / z2 \\ 1 \end{pmatrix}$$

```
hMat hi     = (T*R).Inv();
float z2    = (hi.v[2]*v.x + hi.v[6]*v.y+ hi.v[10]*v.z)/(z-hi.v[14]);
v   = v*(1./z2);
v   = hi*v;
v.z = z;
return v;
```


Ex) uWnd-49-MC-Invp

This example is one of the most complex examples in the field of Graphics

- Right click for Movement



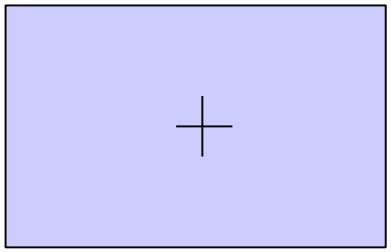
Ex) uWnd-49-MC-Invp
uCam::Inv()

```
uVector uCam::Inv(uVector t,float z)
{
    uVector v;
    t.x-=320;
    t.y =240-t.y;

    t.x = t.x/S.v[0];
    t.y = t.y/S.v[5];
    t.z = 0;

    // Projection matrix
    float n=-1;
    float f=-65535;
    float angle     = 90;
    float aspect    = 1;
    float ct  = 1./tan(RAD(angle)/2);

    hMat pi;
    pi.v[0] = aspect/ct;
    pi.v[5] = 1./ct;
    pi.v[10]    = 0;
    pi.v[11]    = (n-f)/(2*n*f);
    pi.v[15]    = (n+f)/(2*n*f);

    pi.v[12]    = 0;
    pi.v[13]    = 0;
    pi.v[14]    = -1;

    v           = pi*t;
```

$$v_p = \begin{pmatrix} x_p \\ y_p \\ 0 \\ h \end{pmatrix}$$

$plane, z = 0$

(0,0)



(640,480)

Click coordinate of onLButtonDown()

Inverse scaling of [-1,1] → [-320,320]

Process 1) $v' = P^{-1}v_p$

```
hMat hi     = (T*R).Inv();
float z2    = (hi.v[2]*v.x + hi.v[6]*v.y+ hi.v[10]*v.z)/(z-hi.v[14]);
v   = v*(1./z2);
v   = hi*v;
v.z = z;
return v;
```

$$z_2 = \frac{n+f}{2nf} h = \frac{h_{31}t_x + h_{32}t_y + h_{33}t_z}{\text{z-h}_{34}}$$

$$v = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \\ t_z \\ \frac{n+f}{2nf} h(=z2) \end{pmatrix} = \begin{pmatrix} t_x / z2 \\ t_y / z2 \\ t_z / z2 \\ 1 \end{pmatrix}$$

Process2) $v = H^{-1}v'$

Right mouse click changes Xd

```
void uWnd::OnRButtonDown(UINT nFlags, CPoint point)
{
    Xd = cam.Inv(uVector(point.x,point.y,0),0);
    CWnd::OnRButtonDown(nFlags, point);
}
```

$$v_p = \begin{pmatrix} x_p & y_p & 0 & h \end{pmatrix}^T$$

$$z = 0$$