Computer Graphics and Programming
Lecture 5

Extended Primitives

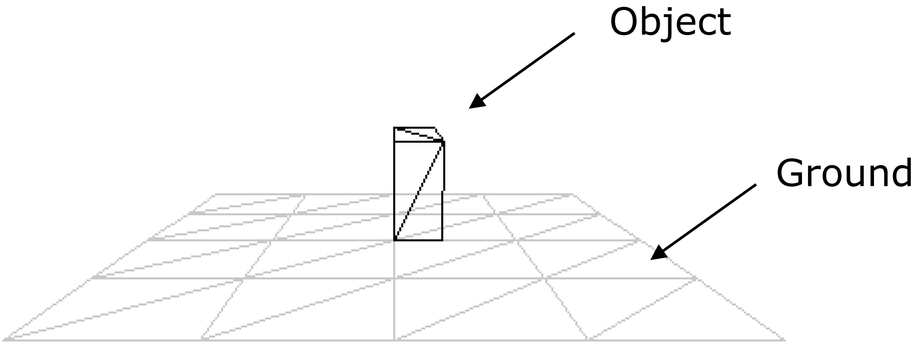
Jeong-Yean Yang

2020/10/22

1 Ground, Axis, and so on.

Ground Modeling

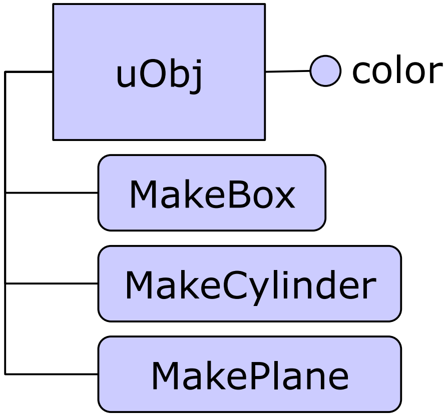
- 3D Environment is confused.
  - Ground(Grid plane) is helpful for intuitive understanding.

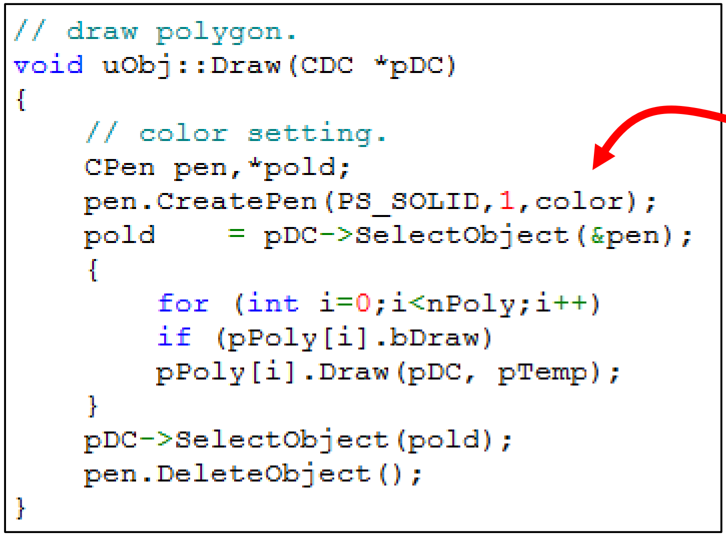


- Extending uObj::MakePlane in Ch. 3

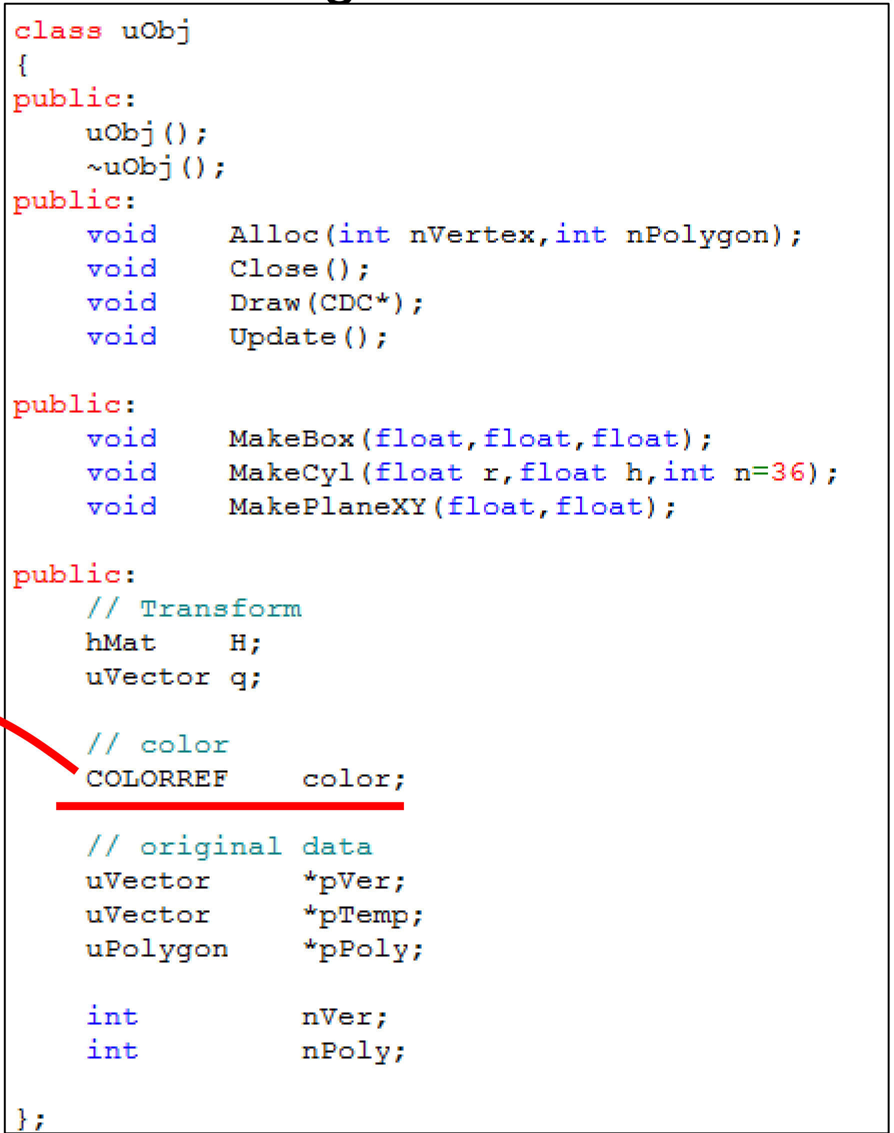
Color for Ground Object

ex) uWnd-31-Ground-Triangle



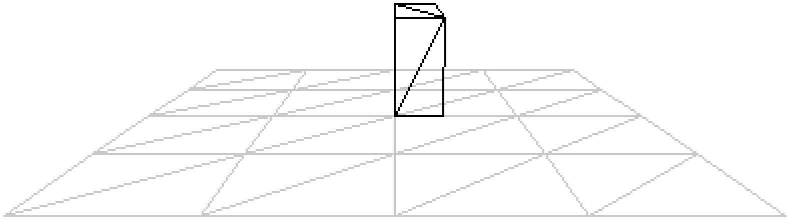


```
// draw polygon.
void uObj::Draw(CDC *pDC)
{
    // color setting.
    CPen pen,*pold;
    pen.CreatePen(PS_SOLID,1,color);
    pold    = pDC->SelectObject(&pen);
    {
        for (int i=0;i<nPoly;i++)
        if (pPoly[i].bDraw)
        pPoly[i].Draw(pDC, pTemp);
    }
    pDC->SelectObject(pold);
    pen.DeleteObject();
}
```



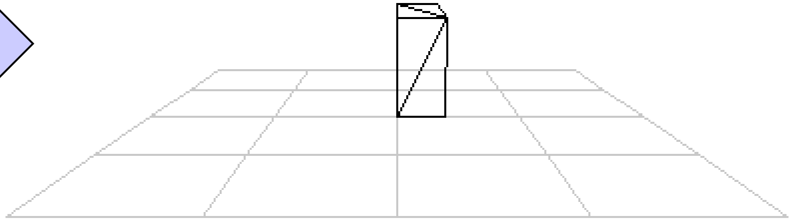
```
class uObj
{
public:
    uObj();
    ~uObj();
public:
    void    Alloc(int nVertex,int nPolygon);
    void    Close();
    void    Draw(CDC*);
    void    Update();

public:
    void    MakeBox(float,float,float);
    void    MakeCyl(float r,float h,int n=36);
    void    MakePlaneXY(float,float);

public:
    // Transform
    hMat    H;
    uVector q;

    // color
    COLORREF    color;

    // original data
    uVector     *pVer;
    uVector     *pTemp;
    uPolygon    *pPoly;

    int         nVer;
    int         nPoly;

};
```

Question: If we modify Ground Object, How can we do?



uWnd-31-Ground-Triangle

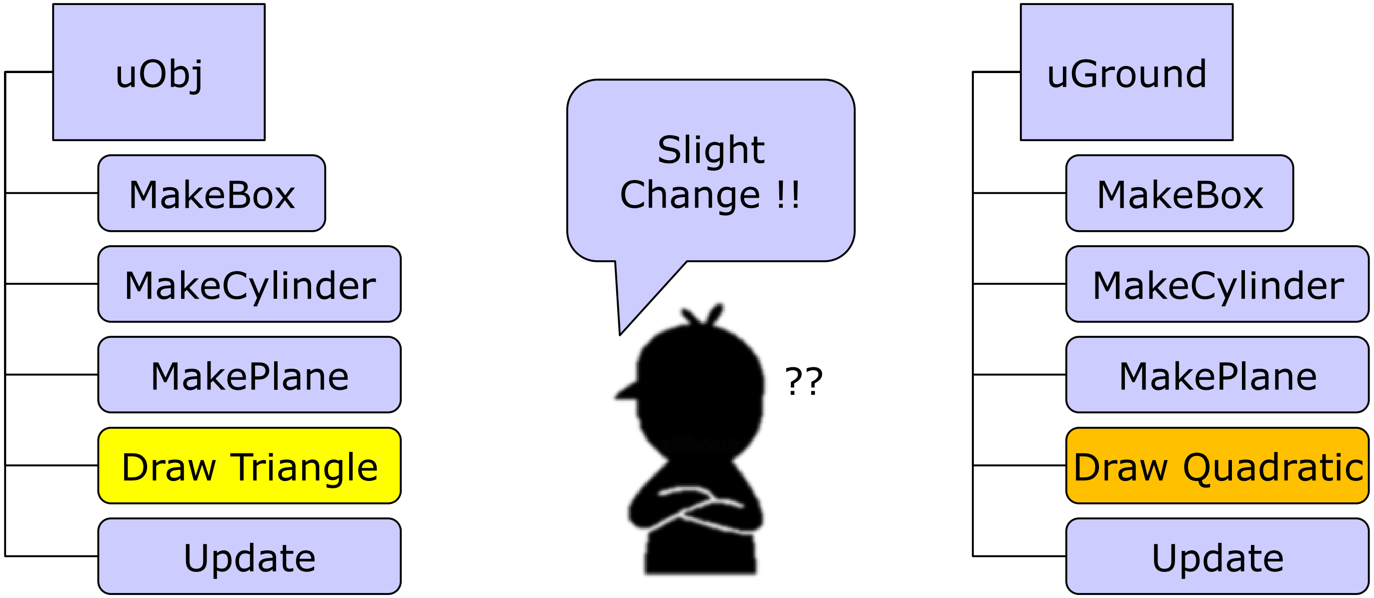
Look better



uWnd-32-Ground-Quad

- We MUST modify uObj class
- Lets **redefine uGround class by subclassing uObj**

Wrapper Class (Subclassing Class+ Overriding function → Inheritance)



uObj
- MakeBox
- MakeCylinder
- MakePlane
- Draw Triangle
- Update

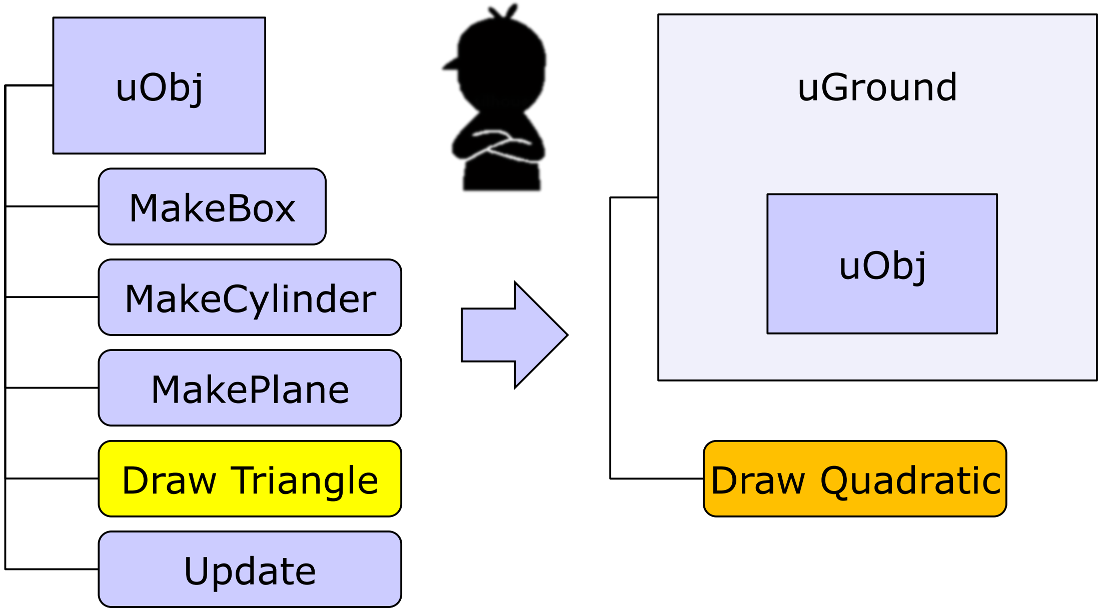
Slight Change !!

??

uGround
- MakeBox
- MakeCylinder
- MakePlane
- Draw Quadratic
- Update

- C++ has been popular with Subclassing Technique.

Wrapper Class
(Subclassing or Inherited Class)



uObj

MakeBox

MakeCylinder

MakePlane

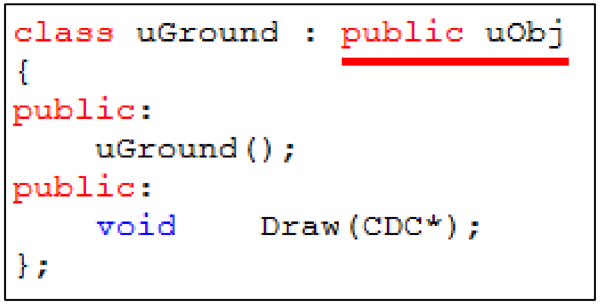
Draw Triangle

Update

uGround

uObj

Draw Quadratic



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

Inheritance

- uGround is inherited by uObj
  - uGround has every features of uObj.

Class uGround: public uObj

uGround ground

ground.MakeBox (o)
ground.Draw (o)

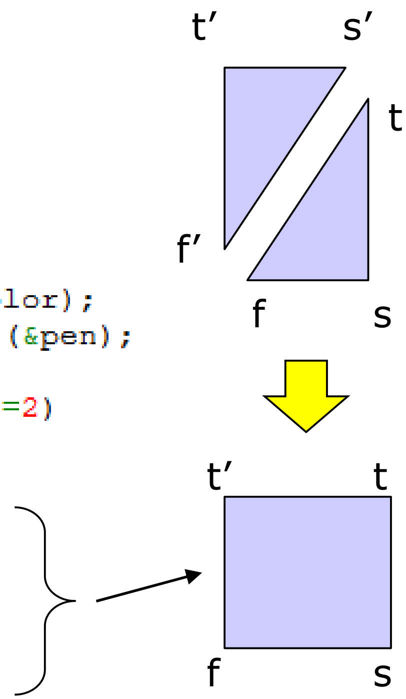
Only Modify uGround::Draw

ex) uWnd-31-Ground-Quad

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

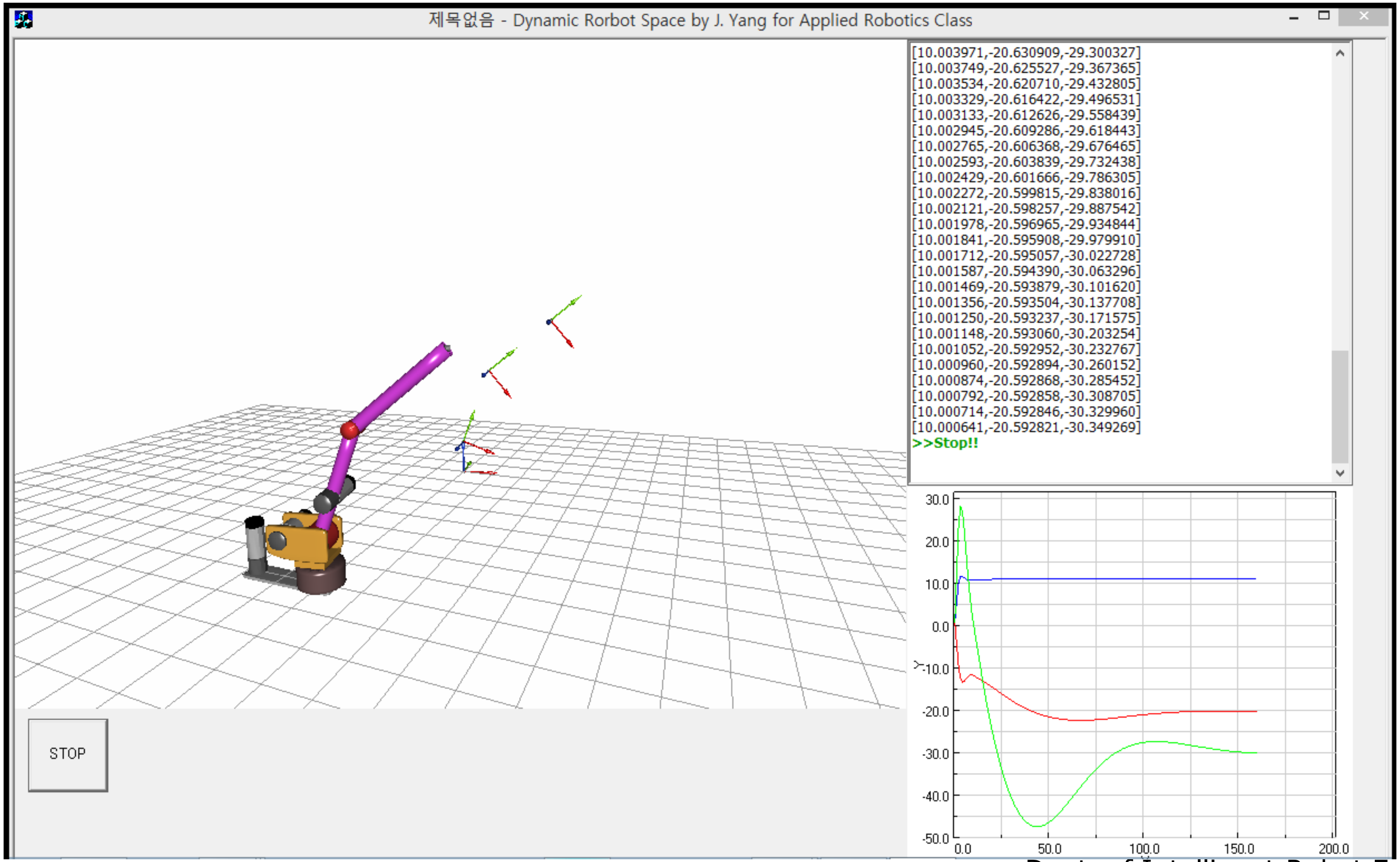
- uObj::Draw()
  - Draw two triangles
- uGround::Draw()
  - Draw one rectangle

```
uGround::uGround()
{
}

void uGround::Draw(CDC *pDC)
{
    // color setting.
    CPen pen,*pold;
    pen.CreatePen(PS_SOLID,1,color);
    pold    = pDC->SelectObject(&pen);
    {
        for (int i=0;i<nPoly;i+=2)
        //if (pPoly[i].bDraw)
        {
            int f,s,t,t2;
            f   = pPoly[i].f;
            s   = pPoly[i].s;
            t   = pPoly[i].t;
            t2  = pPoly[i+1].t;

            pDC->MoveTo( pTemp[f].x, pTemp[f].y);
            pDC->LineTo( pTemp[s].x, pTemp[s].y);
            pDC->LineTo( pTemp[t].x, pTemp[t].y);
            pDC->LineTo( pTemp[t2].x, pTemp[t2].y);
            pDC->LineTo( pTemp[f].x, pTemp[f].y);
        }
    }
    pDC->SelectObject(pold);
    pen.DeleteObject();
}
```



XYZ Axis Modeling

- Axis is also helpful for understanding 3D environment.

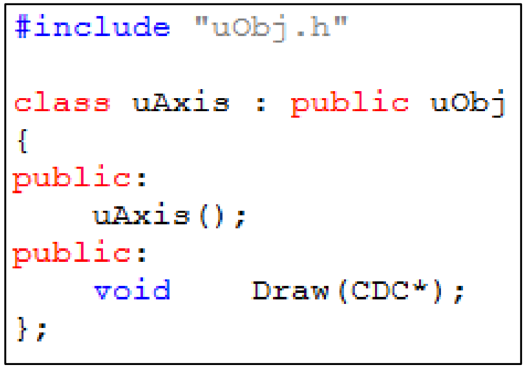

제목없음 - Dynamic Rorbot Space by J. Yang for Applied Robotics Class
STOP
30.0
20.0
10.0
0.0
-10.0
-20.0
-30.0
-40.0
-50.0
Y
0.0
50.0
100.0
150.0
200.0

```
[10.003971,-20.630909,-29.300327]
[10.003749,-20.625527,-29.367365]
[10.003534,-20.620710,-29.432805]
[10.003329,-20.616422,-29.496531]
[10.003133,-20.612626,-29.558439]
[10.002945,-20.609286,-29.618443]
[10.002765,-20.606368,-29.676465]
[10.002593,-20.603839,-29.732438]
[10.002429,-20.601666,-29.786305]
[10.002272,-20.599815,-29.838016]
[10.002121,-20.598257,-29.887542]
[10.001978,-20.596965,-29.934844]
[10.001841,-20.595908,-29.979910]
[10.001712,-20.595057,-30.022728]
[10.001587,-20.594390,-30.063296]
[10.001469,-20.593879,-30.101620]
[10.001356,-20.593504,-30.137708]
[10.001250,-20.593237,-30.171575]
[10.001148,-20.593060,-30.203254]
[10.001052,-20.592952,-30.232767]
[10.000960,-20.592894,-30.260152]
[10.000874,-20.592868,-30.285452]
[10.000792,-20.592858,-30.308705]
[10.000714,-20.592846,-30.329960]
[10.000641,-20.592821,-30.349269]
>>Stop!!
```

uAxis from uObj

Refer to uWnd-32-Axis

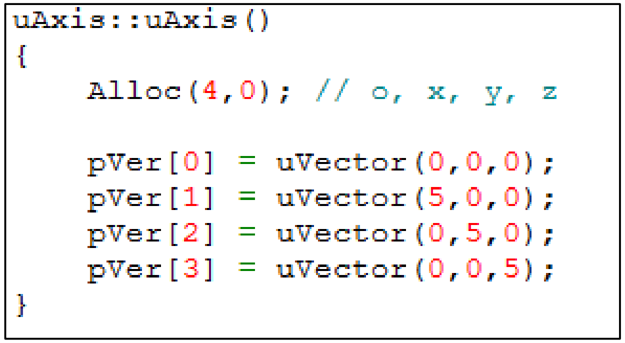
- Subclassing uAxis from uObj



```
#include "uObj.h"

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

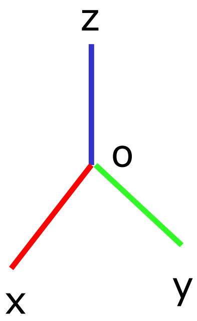
Subclassing uObj

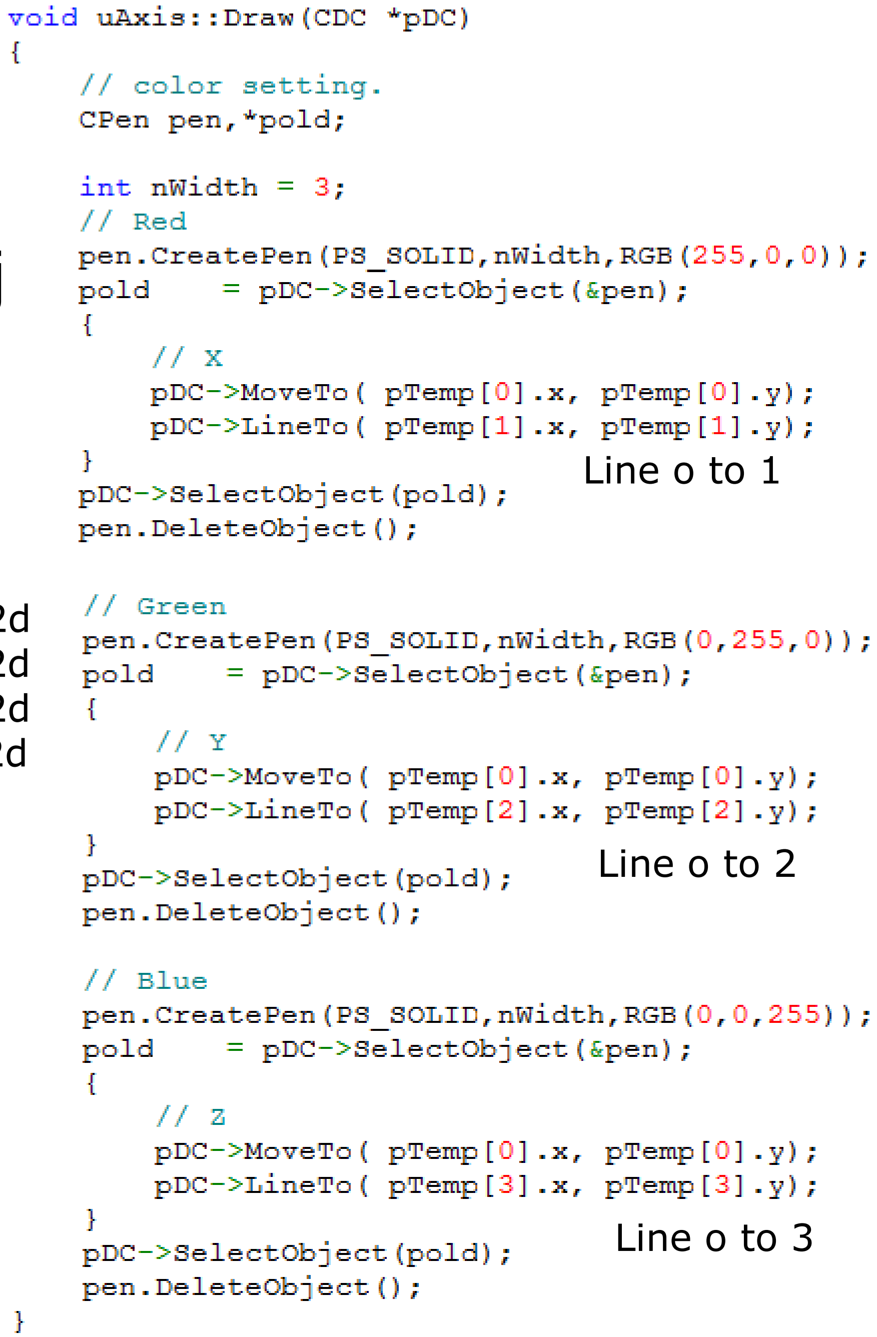
pVer[0] :o
pVer[1] :x
pVer[2] :y
pVer[3] :z
→
pTemp[0]: o in 2d
pTemp[1]: x in 2d
pTemp[2]: y in 2d
pTemp[3]: z in 2d



```
uAxis::uAxis()
{
    Alloc(4,0); // o, x, y, z

    pVer[0] = uVector(0,0,0);
    pVer[1] = uVector(5,0,0);
    pVer[2] = uVector(0,5,0);
    pVer[3] = uVector(0,0,5);
}
```

4 vectors are needed
O(origin), x,y,z





```
void uAxis::Draw(CDC *pDC)
{
    // color setting.
    CPen pen,*pold;

    int nWidth = 3;
    // Red
    pen.CreatePen(PS_SOLID,nWidth,RGB(255,0,0));
    pold     = pDC->SelectObject(&pen);
    {
        // X
        pDC->MoveTo( pTemp[0].x, pTemp[0].y);
        pDC->LineTo( pTemp[1].x, pTemp[1].y);
    }
    pDC->SelectObject(pold);
    pen.DeleteObject();

    // Green
    pen.CreatePen(PS_SOLID,nWidth,RGB(0,255,0));
    pold     = pDC->SelectObject(&pen);
    {
        // Y
        pDC->MoveTo( pTemp[0].x, pTemp[0].y);
        pDC->LineTo( pTemp[2].x, pTemp[2].y);
    }
    pDC->SelectObject(pold);
    pen.DeleteObject();

    // Blue
    pen.CreatePen(PS_SOLID,nWidth,RGB(0,0,255));
    pold     = pDC->SelectObject(&pen);
    {
        // Z
        pDC->MoveTo( pTemp[0].x, pTemp[0].y);
        pDC->LineTo( pTemp[3].x, pTemp[3].y);
    }
    pDC->SelectObject(pold);
    pen.DeleteObject();
}
```

Line o to 1

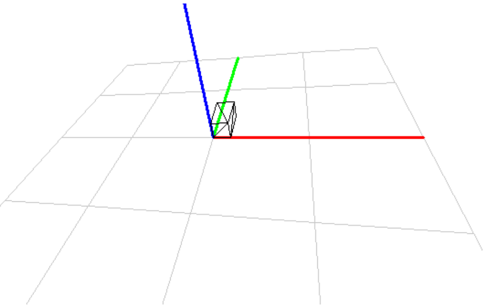
Line o to 2

Line o to 3

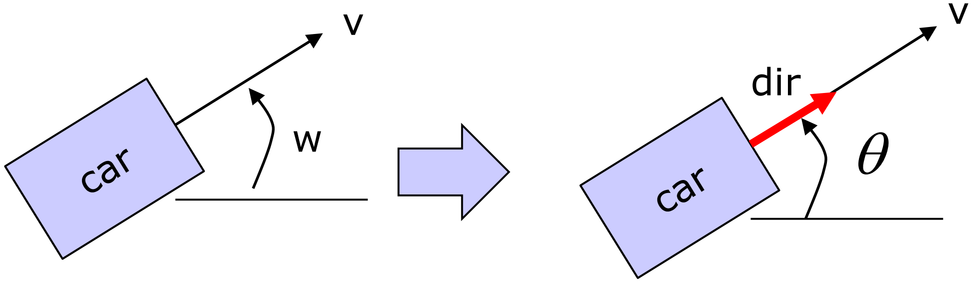
2 Multiple Objects

Extending into Multiple Object

- How to transform Multiple Object?
- Let’s think a Car
  - uWnd-34-Car1 with KEY input for acceleration



Dir: direction vector
Vel: speed
V = dir * vel

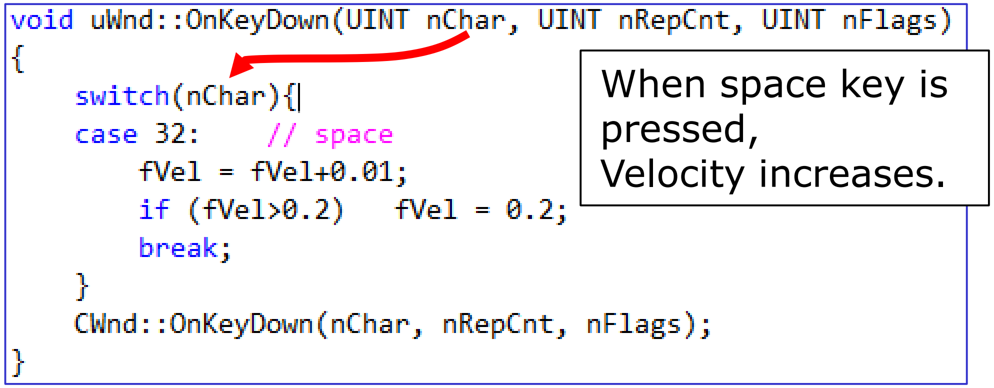


v: velocity
W: angular velocity

$$\theta : heading\ \ angle = \mathrm{atan2}(v.y,\ \ v.x)$$

$$RotZ(\theta)$$

Key Input



```
void uWnd::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    switch(nChar){|
    case 32:     // space
        fVel = fVel+0.01;
        if (fVel>0.2)    fVel = 0.2;
        break;
    }
    CWnd::OnKeyDown(nChar, nRepCnt, nFlags);
}
```

When space key is pressed,
Velocity increases.

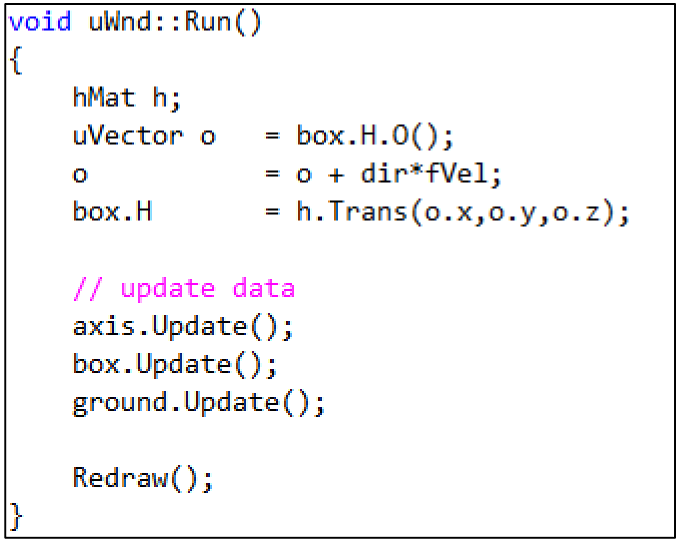
- If you press a space key,
  - Vel ← Vel + 0.01
- Limitation of Maximum speed
  - If (Vel>0.2) Vel = 0.2

car

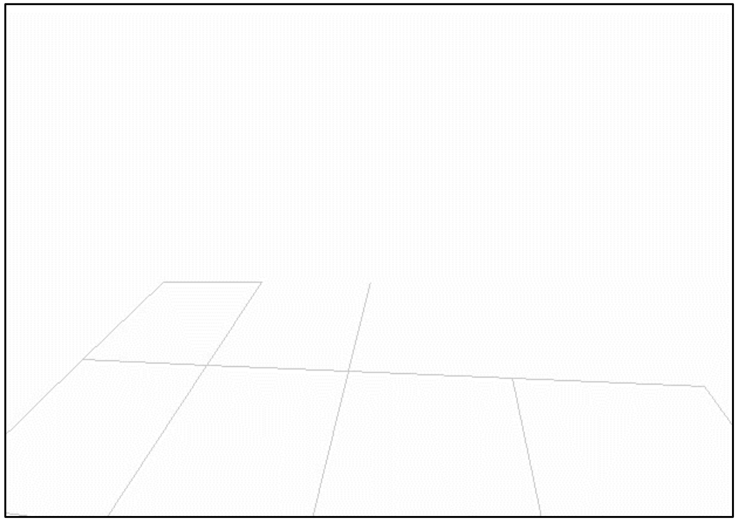
uVector dir(0,1,0) → direction vector with heading angle

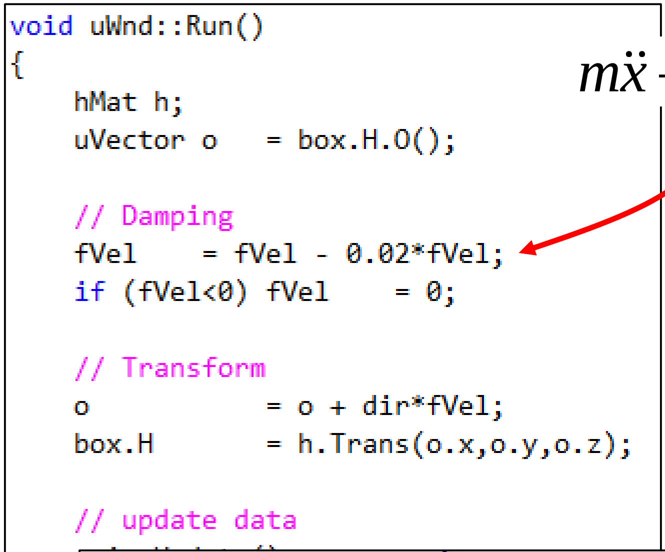
float fVel = 0; → Car's velocity

Car Moving with a Velocity and Damping



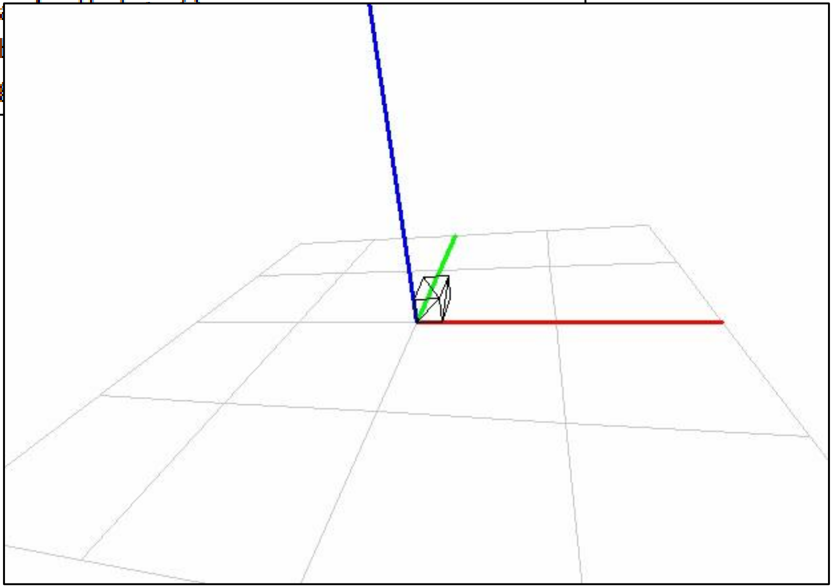
```
void uWnd::Run()
{
    hMat h;
    uVector o     = box.H.O();
    o             = o + dir*fVel;
    box.H         = h.Trans(o.x,o.y,o.z);

    // update data
    axis.Update();
    box.Update();
    ground.Update();

    Redraw();
}
```



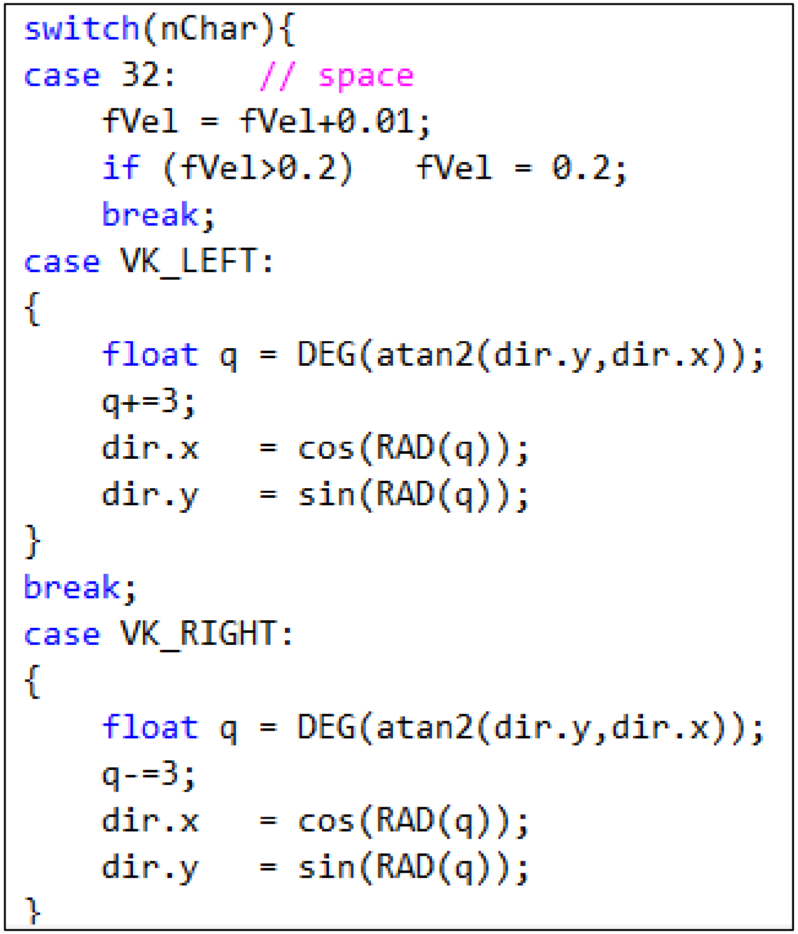


```
void uWnd::Run()
{
    hMat h;
    uVector o   = box.H.O();

    // Damping
    fVel     = fVel - 0.02*fVel;
    if (fVel<0) fVel    = 0;

    // Transform
    o              = o + dir*fVel;
    box.H          = h.Trans(o.x,o.y,o.z);

    // update data
    a
    b
    g
```

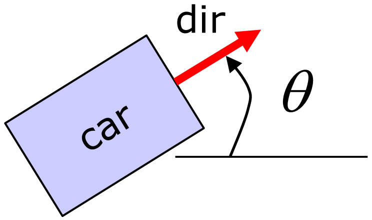
$$m\ddot{x} + \underline{c\dot{x}} + kx = F$$



Car Navigation by
Pressing Left-Right Key for direction change
uWnd-35-Car2



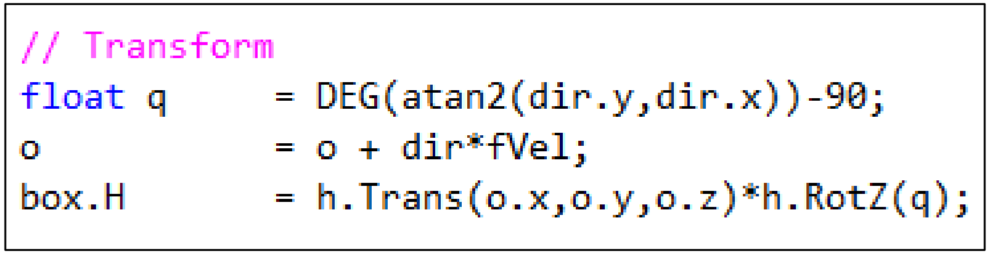
```
switch(nChar){
case 32:     // space
    fVel = fVel+0.01;
    if (fVel>0.2)   fVel = 0.2;
    break;
case VK_LEFT:
{
    float q = DEG(atan2(dir.y,dir.x));
    q+=3;
    dir.x   = cos(RAD(q));
    dir.y   = sin(RAD(q));
}
break;
case VK_RIGHT:
{
    float q = DEG(atan2(dir.y,dir.x));
    q-=3;
    dir.x   = cos(RAD(q));
    dir.y   = sin(RAD(q));
}
```



$$\theta : heading\ \ angle = \mathrm{atan2}(v.y,\ v.x)$$

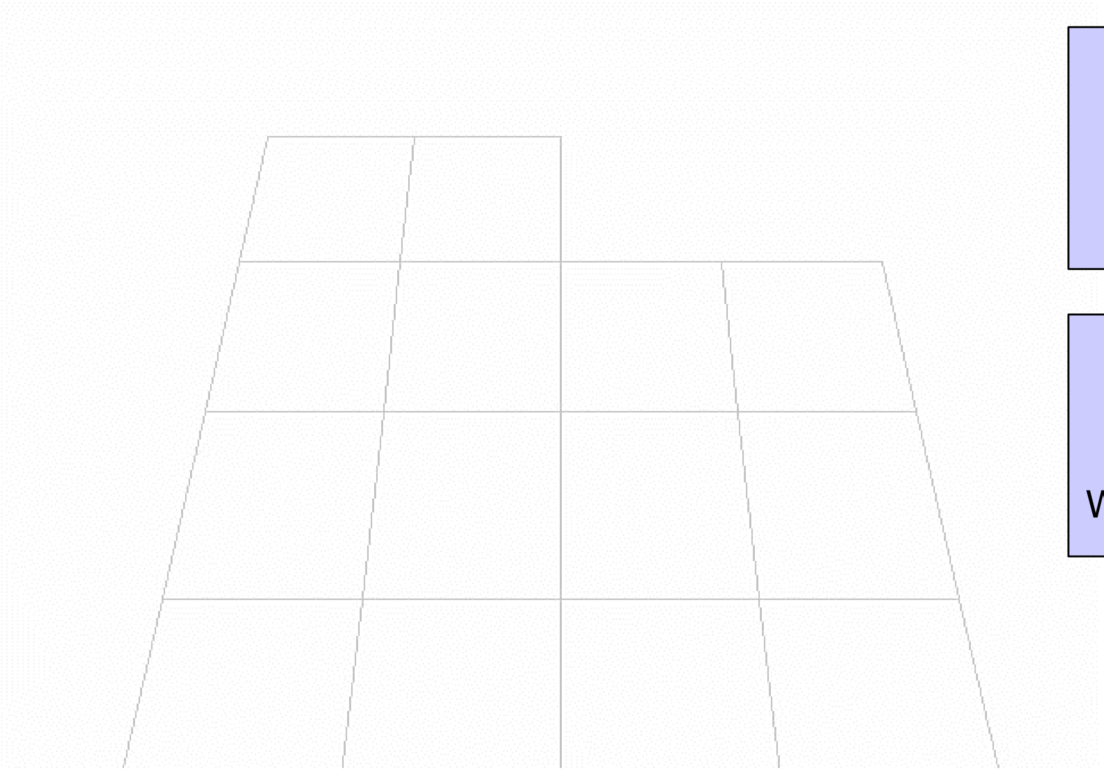
- Left +3 deg, Right -3 deg

→ Counter clock wise along Z



```
// Transform
float q     = DEG(atan2(dir.y,dir.x))-90;
o           = o + dir*fVel;
box.H       = h.Trans(o.x,o.y,o.z)*h.RotZ(q);
```

Example: uWnd-35-Car2



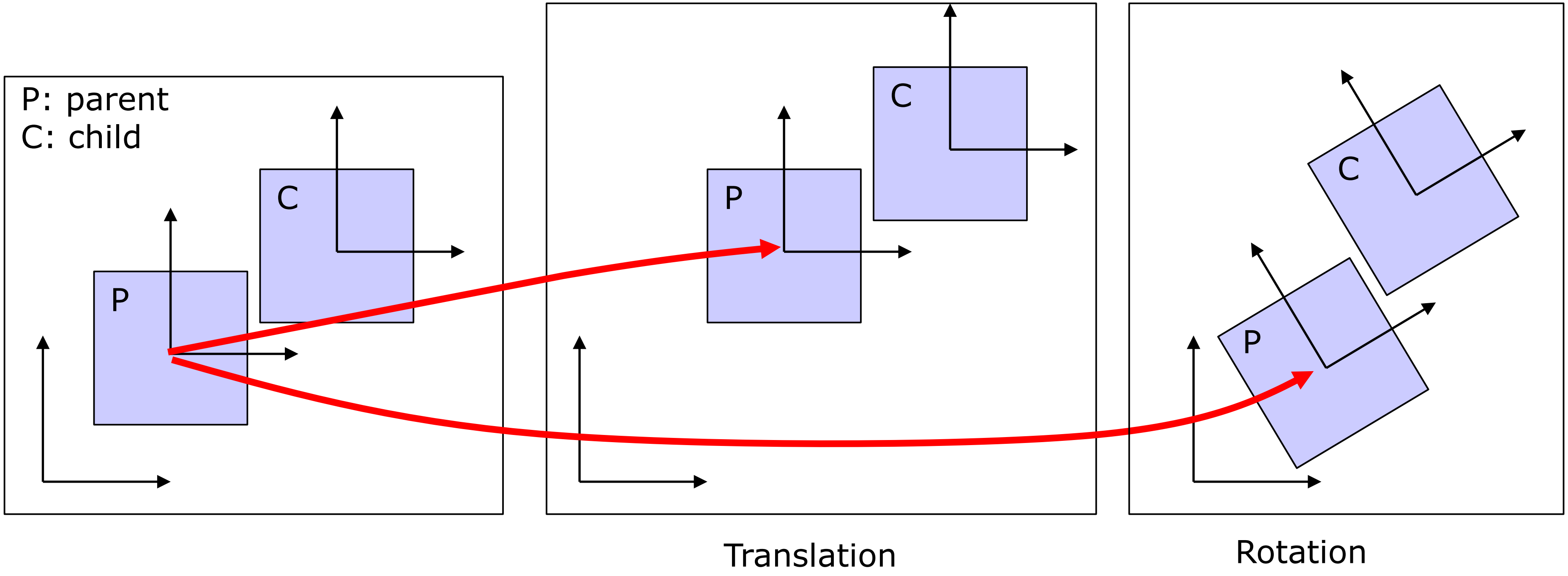
HW 5,8, 9 !!

If we Add Wheel,

Multiple Object Car
Will be very complex

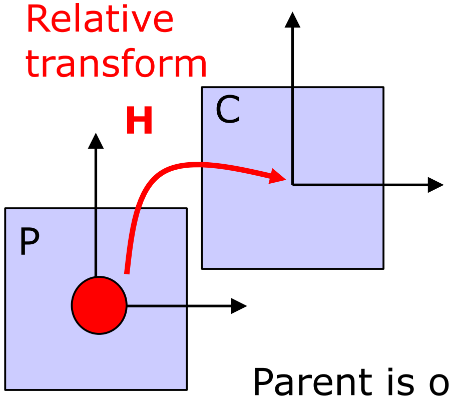
RotZ is NOT done at the center

Multiple Object:
An Object has other Objects

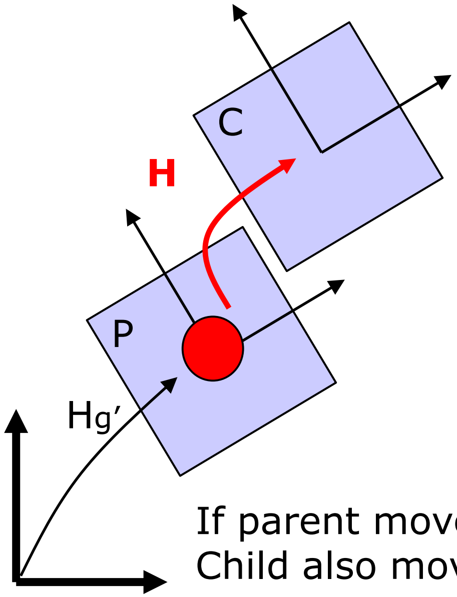


- When Parent, P moves, Child, C also moves.
- Translation is Easy but Rotation is More complex
- We need to design Hierarchical Approach

Child has the Relative Transform, H w.r.t. Parent



Parent is on origin.



If parent moves, Child also moves

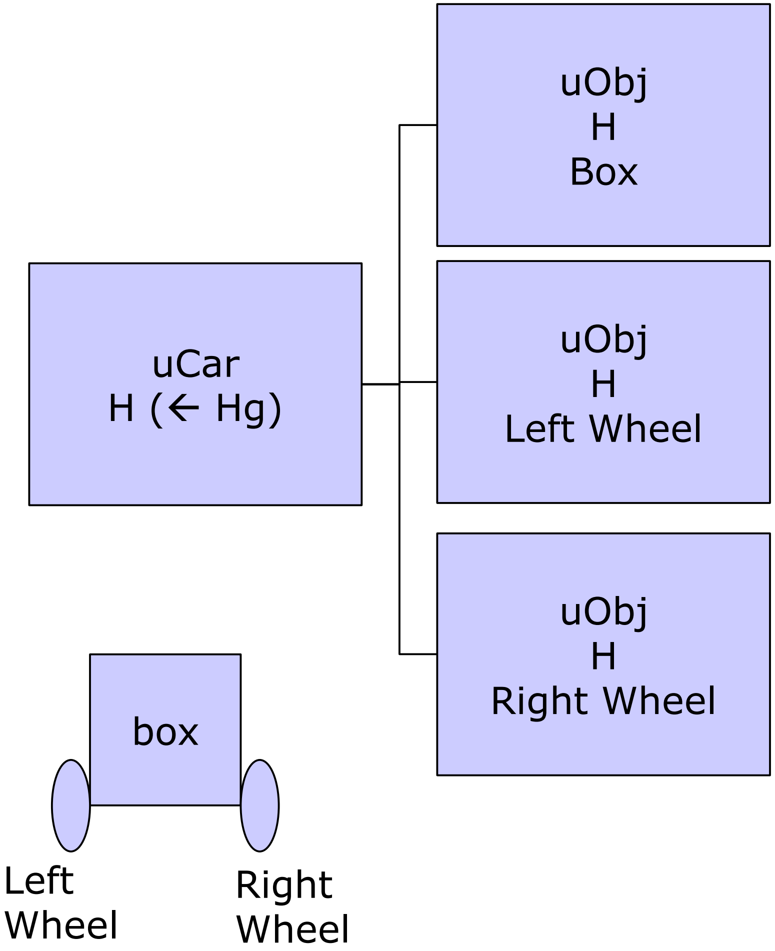
$$H_g = I$$
$$P_{current} = H_g P = P$$
$$C_{current} = H_g HC = HC$$

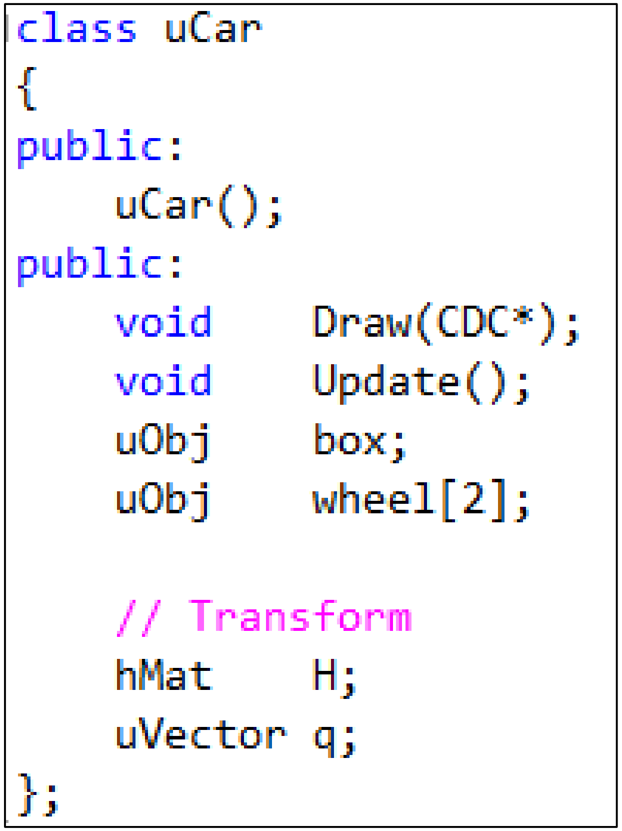
$$H_g = H_g'$$
$$P_{current} = H_g' P$$
$$C_{current} = H_g' HC$$

- Child's Relative Transform, H is constant

  → Child looks fixed on Parent.

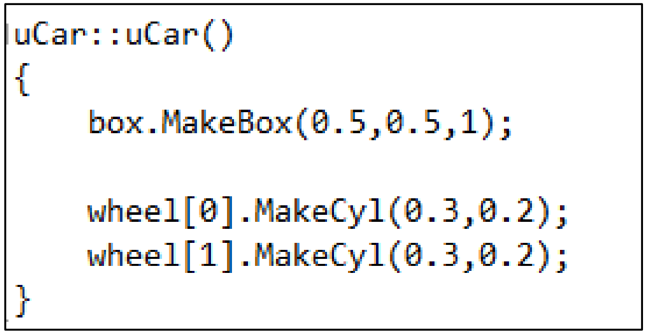
Extending uObj into Multiple Object, uCar Class



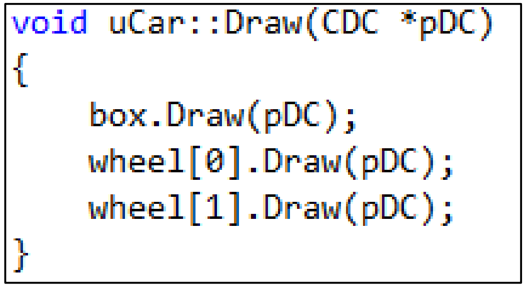


```
class uCar
{
public:
    uCar();
public:
    void    Draw(CDC*);
    void    Update();
    uObj    box;
    uObj    wheel[2];

    // Transform
    hMat    H;
    uVector q;
};
```

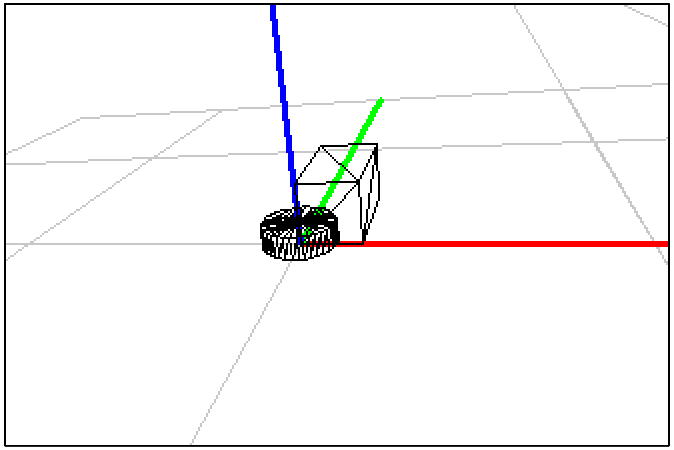
uCar Geometry Design

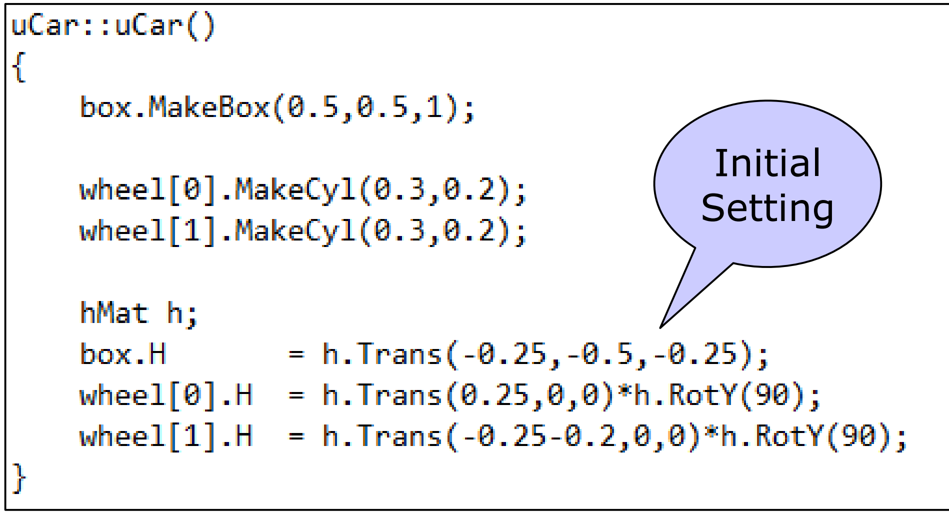


```
uCar::uCar()
{
    box.MakeBox(0.5,0.5,1);

    wheel[0].MakeCyl(0.3,0.2);
    wheel[1].MakeCyl(0.3,0.2);
}
```



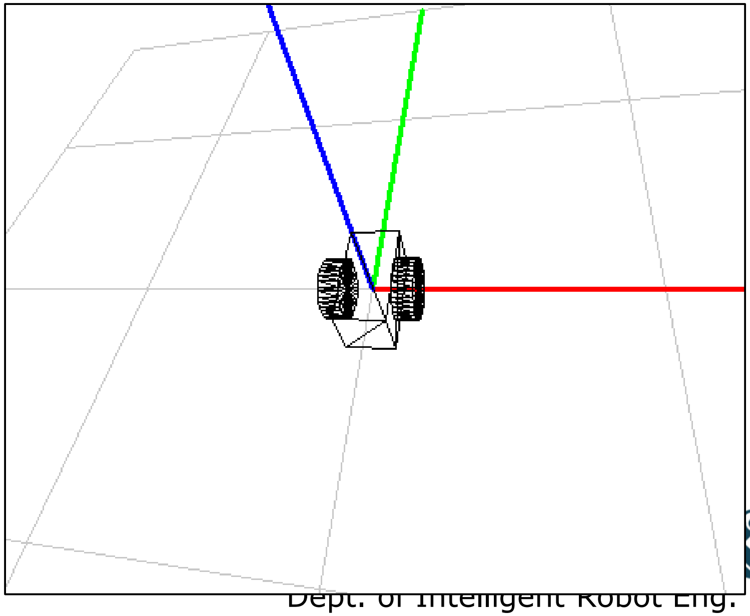
```
void uCar::Draw(CDC *pDC)
{
    box.Draw(pDC);
    wheel[0].Draw(pDC);
    wheel[1].Draw(pDC);
}
```



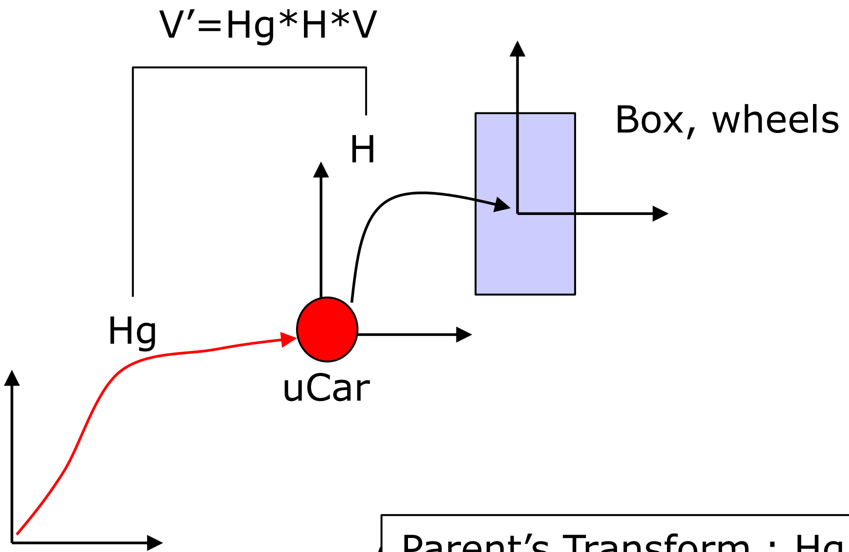


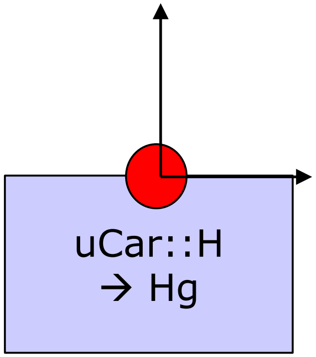
```
uCar::uCar()
{
    box.MakeBox(0.5,0.5,1);

    wheel[0].MakeCyl(0.3,0.2);
    wheel[1].MakeCyl(0.3,0.2);

    hMat h;
    box.H       = h.Trans(-0.25,-0.5,-0.25);
    wheel[0].H  = h.Trans(0.25,0,0)*h.RotY(90);
    wheel[1].H  = h.Trans(-0.25-0.2,0,0)*h.RotY(90);
}
```

Initial Setting



uObj::Update() Has Parent's Transform





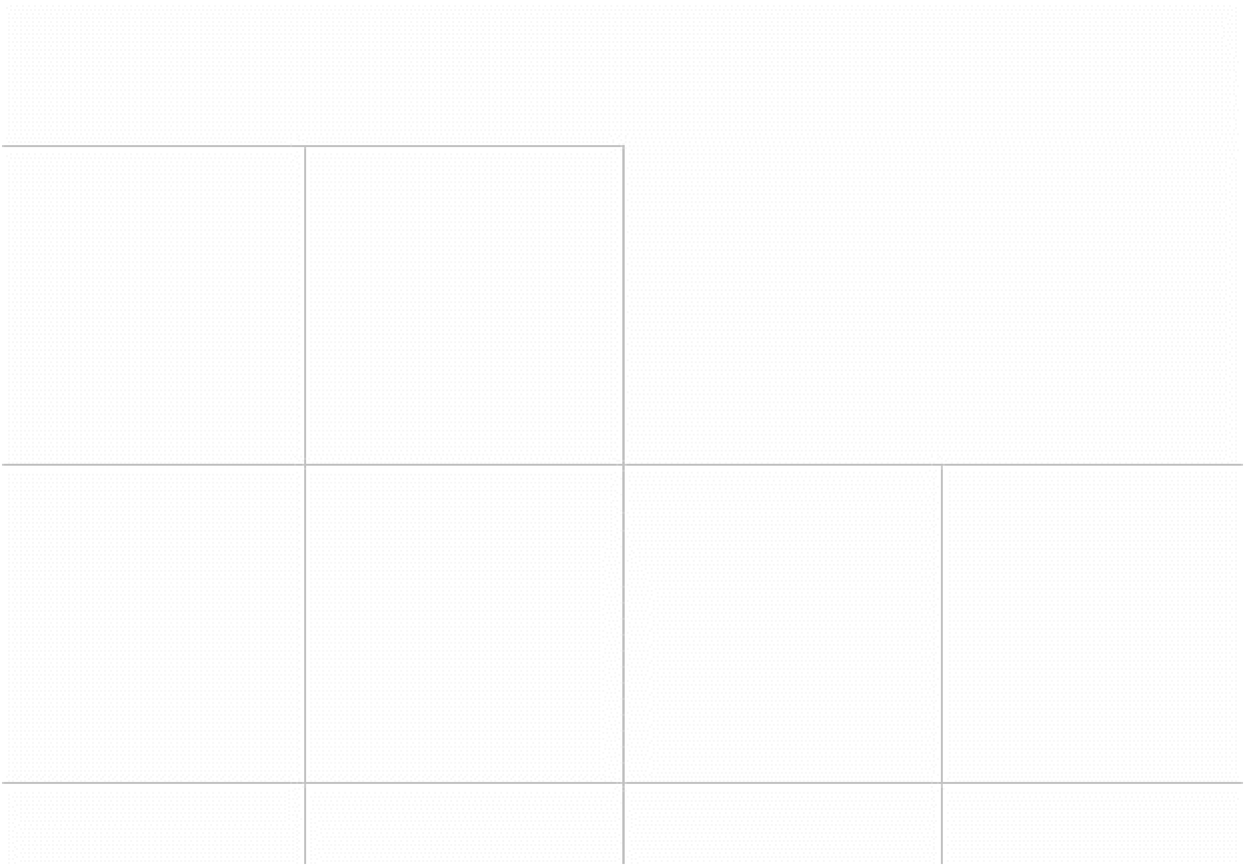
Parent's Transform : Hg

Self Transform : H (Hrel)

```
void uObj::Update()
{
    int i;

    // 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]);
    }
```

```
void uCar::Update()
{
    hMat h;

    box.Hg      = H;
    wheel[0].Hg = H;
    wheel[1].Hg = H;

    box.Update();
    wheel[0].Update();
    wheel[1].Update();
}
```

Example: uWnd-36-Car3

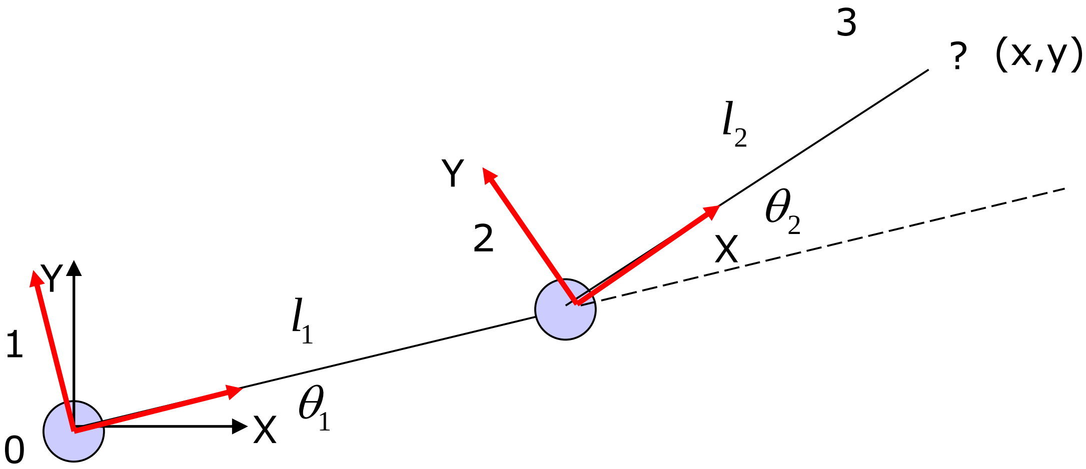


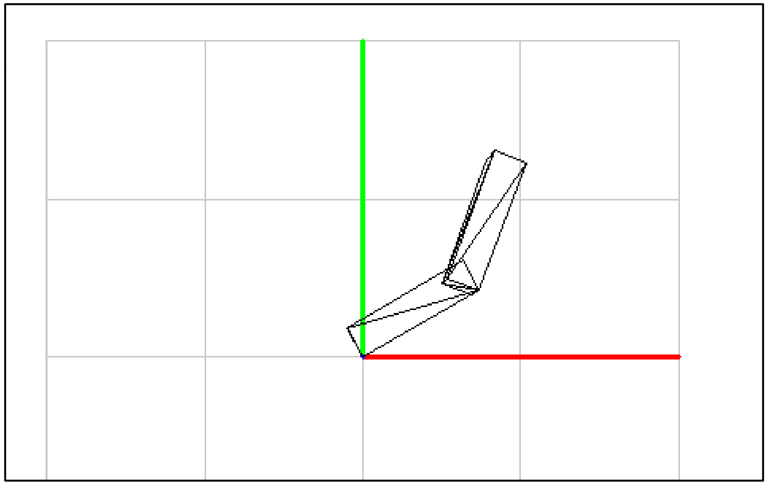
3

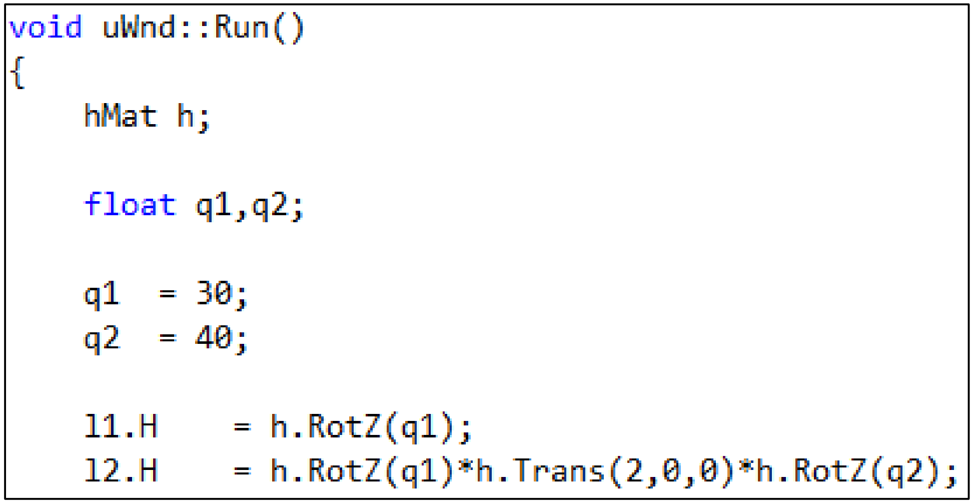
Object Skeleton from Multiple Object

Multiple Object for Robot Arm

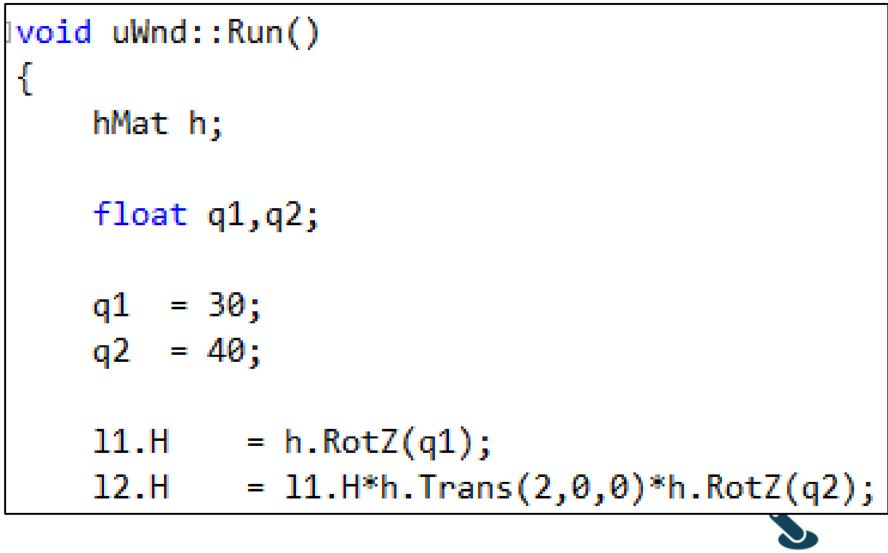
uWnd-37-Robot







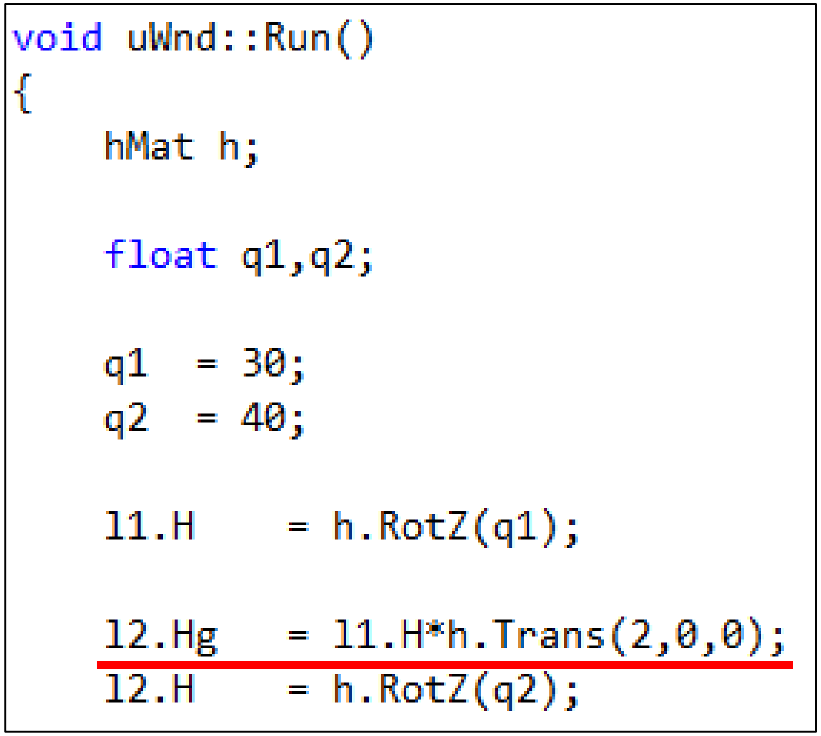
```
void uWnd::Run()
{
    hMat h;

    float q1,q2;

    q1  = 30;
    q2  = 40;

    l1.H    = h.RotZ(q1);
    l2.H    = h.RotZ(q1)*h.Trans(2,0,0)*h.RotZ(q2);
```



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

    float q1,q2;

    q1  = 30;
    q2  = 40;

    l1.H    = h.RotZ(q1);
    l2.H    = l1.H*h.Trans(2,0,0)*h.RotZ(q2);
```

Multi Object with uObj::Hg

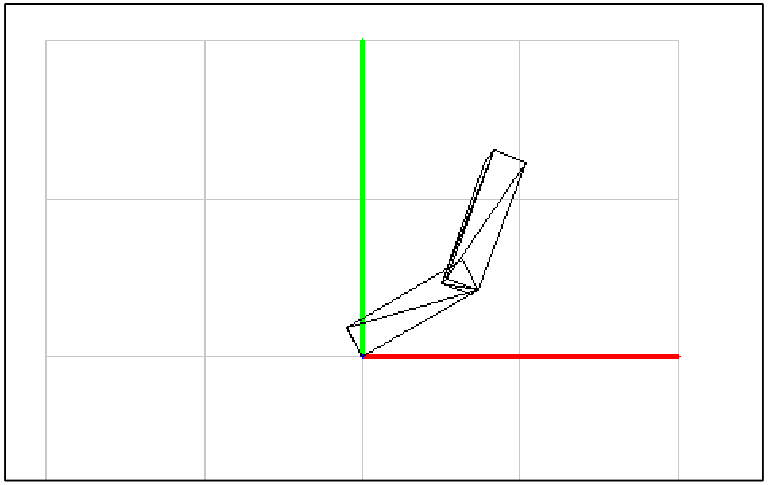
uWnd-38-Robot



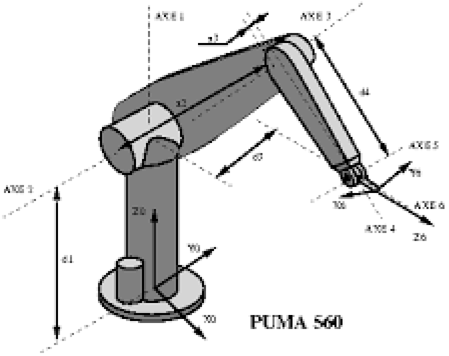
```
void uWnd::Run()
{
    hMat h;

    float q1,q2;

    q1  = 30;
    q2  = 40;

    l1.H     = h.RotZ(q1);

    l2.Hg    = l1.H*h.Trans(2,0,0);
    l2.H     = h.RotZ(q2);
```

- Hg is Parent Object’s Transform.

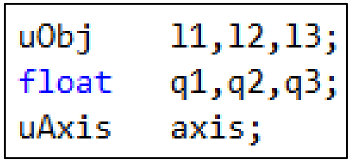
- l1.H and l2.H are regarded as Relative transforms



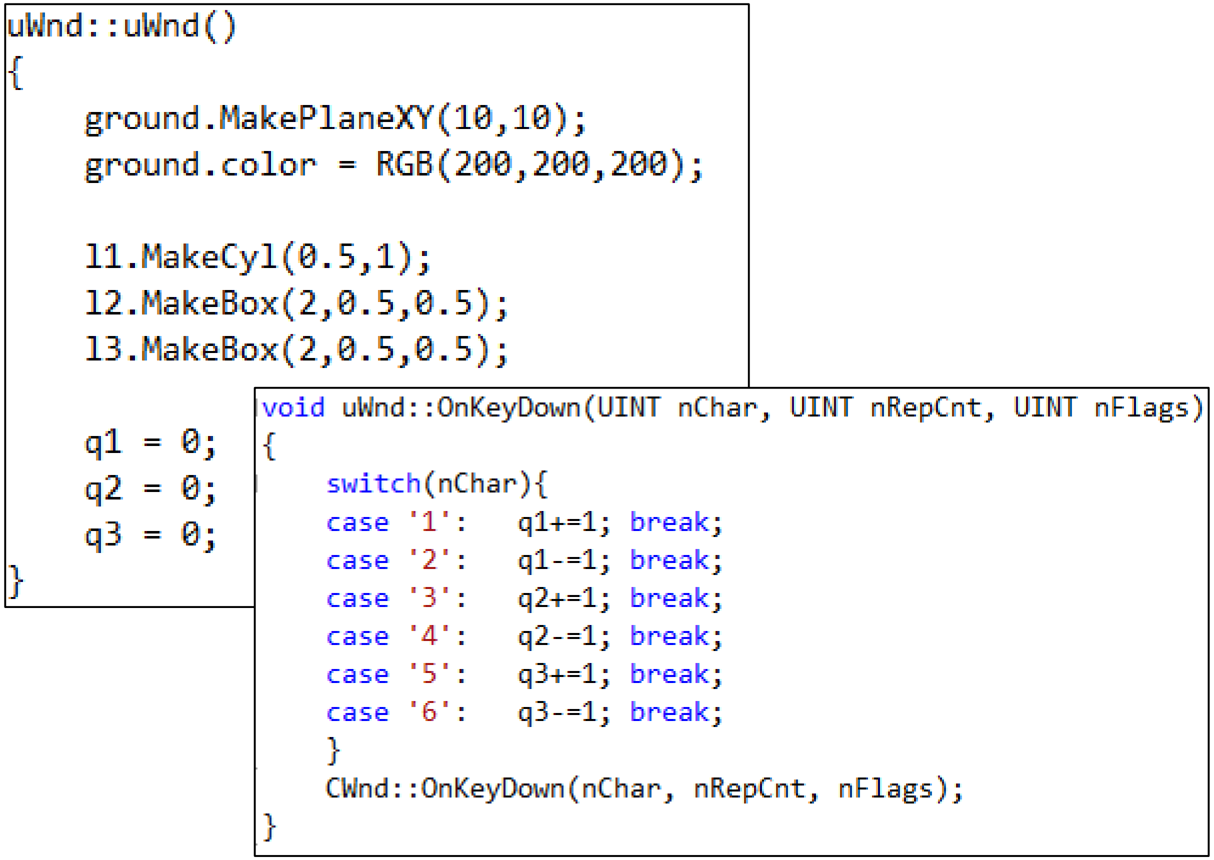
3 DOF PUMA example



uWnd.h



```
uObj    l1,l2,l3;
float   q1,q2,q3;
uAxis   axis;
```

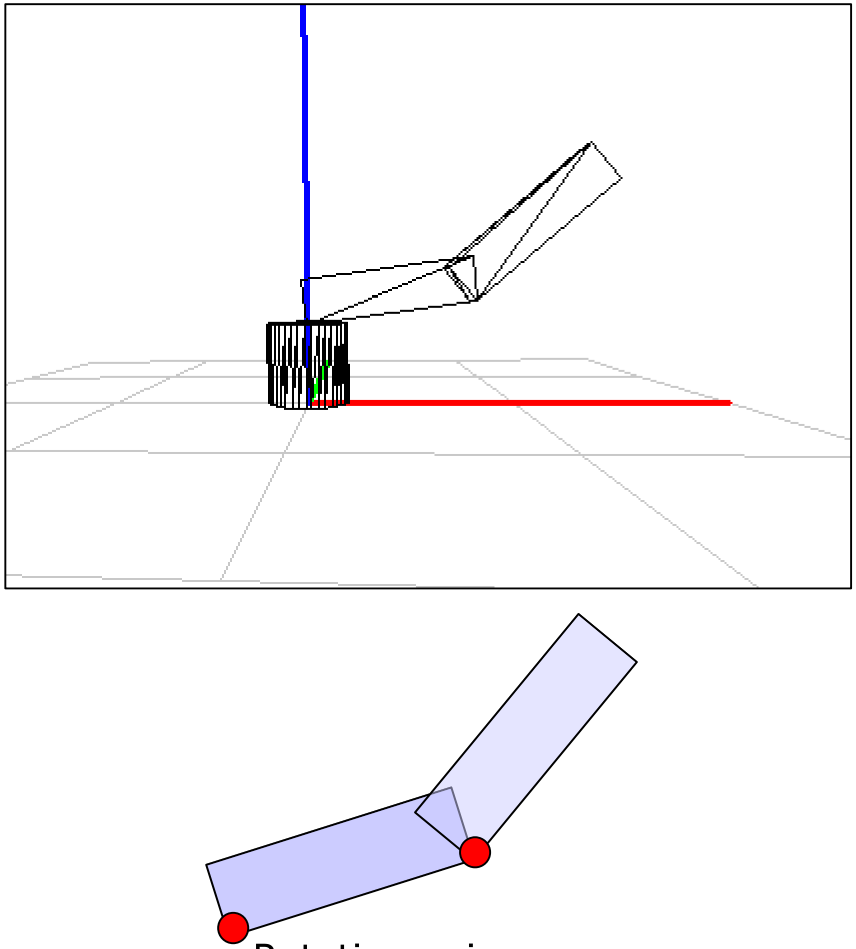


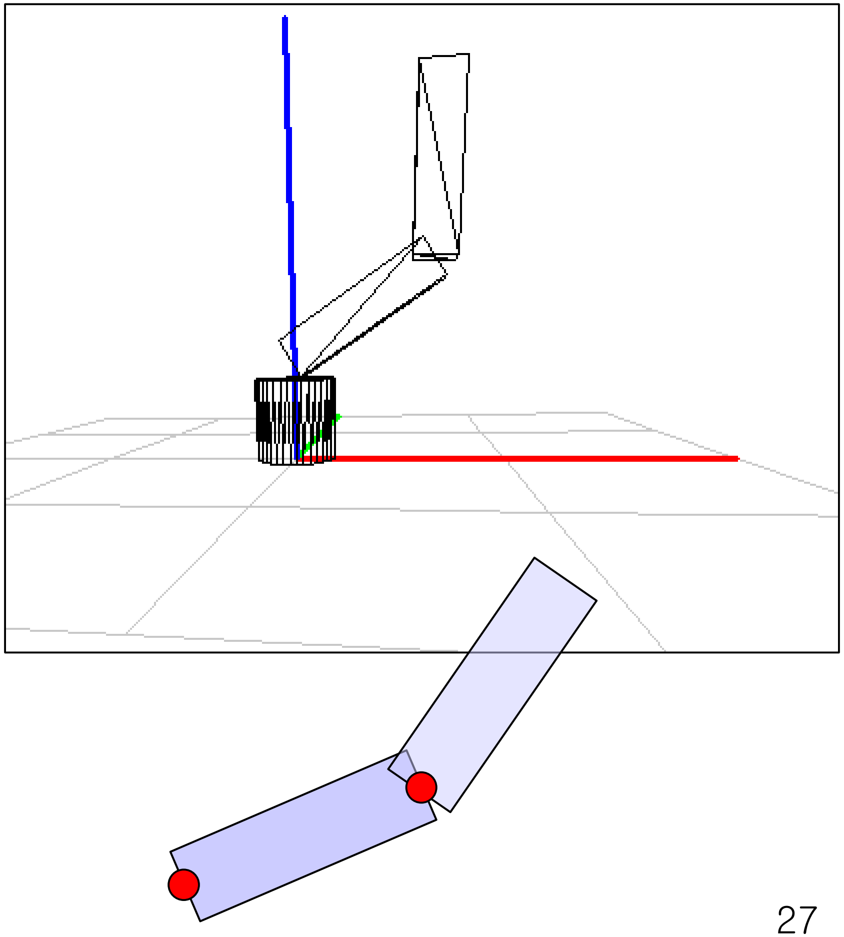
```
uWnd::uWnd()
{
    ground.MakePlaneXY(10,10);
    ground.color = RGB(200,200,200);

    l1.MakeCyl(0.5,1);
    l2.MakeBox(2,0.5,0.5);
    l3.MakeBox(2,0.5,0.5);

    q1 = 0;
    q2 = 0;
    q3 = 0;
}
```

```
void uWnd::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    switch(nChar){
    case '1':   q1+=1; break;
    case '2':   q1-=1; break;
    case '3':   q2+=1; break;
    case '4':   q2-=1; break;
    case '5':   q3+=1; break;
    case '6':   q3-=1; break;
    }
    CWnd::OnKeyDown(nChar, nRepCnt, nFlags);
}
```

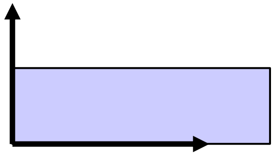
- One Cylinder and two boxes for 3 DOF PUMA.
- Key 1 & 2 for q1, 3 & 4 for q2, 5 & 6 for q3 rotation

Demo: puma.exe and puma2.exe

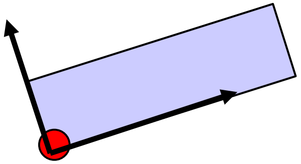




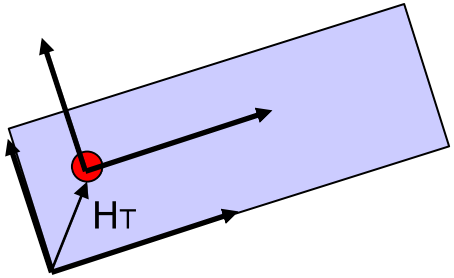
Pivotal Rotation
How we rotate object at Other Positions



Original
object

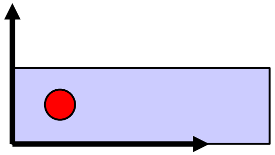


Rotation at an origin

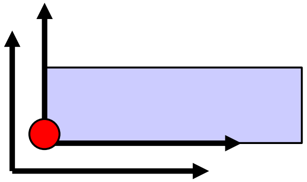


Rotation at the Pivot

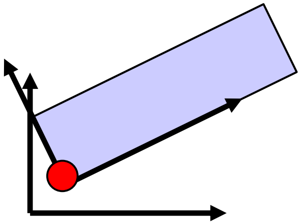
$$H = H_{Trans} H_{Rot} H_{-Trans}$$



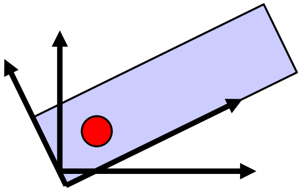
Original
object



Move to
Red point

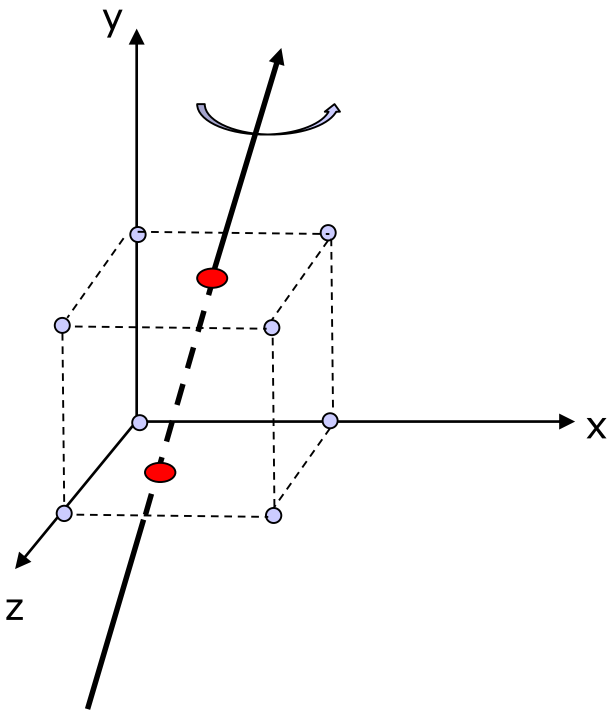


Rotate at
the red pivot



Move back
to origin

Complex Pivotal Rotation



1. Pivotal point has only translation

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

2. Pivotal point has translation and Rotation (very complex)

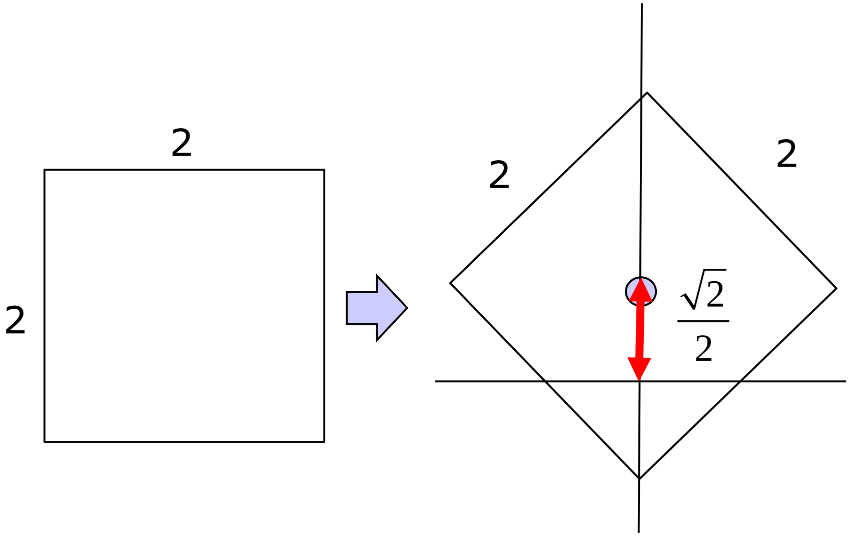
$$H = H_p H_{Rot} H_p^{-1}$$

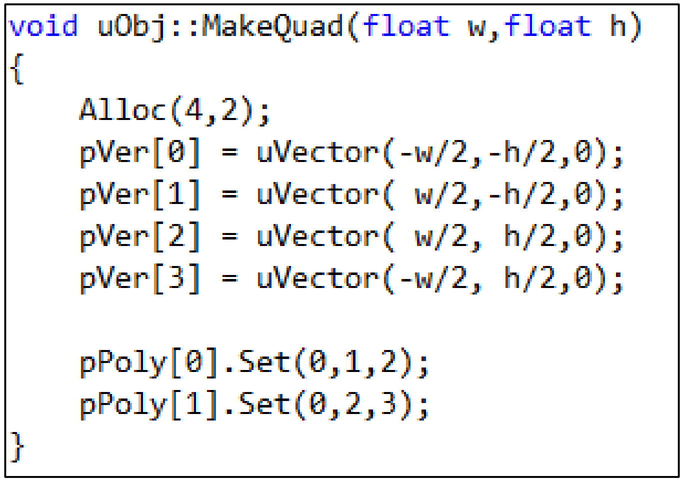
- When Pivotal point transform is very complex,

  → **We need another method, Quaternion**.

4 Example of Multiple Object

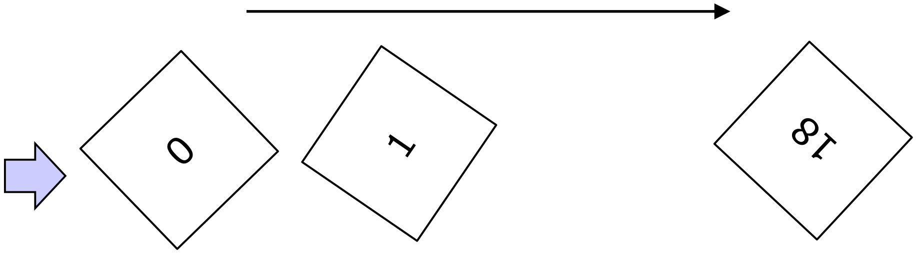
3Dim. Sculpture
Rotation of 19 Rectangles





```
void uObj::MakeQuad(float w,float h)
{
    Alloc(4,2);
    pVer[0] = uVector(-w/2,-h/2,0);
    pVer[1] = uVector( w/2,-h/2,0);
    pVer[2] = uVector( w/2, h/2,0);
    pVer[3] = uVector(-w/2, h/2,0);

    pPoly[0].Set(0,1,2);
    pPoly[1].Set(0,2,3);
}
```

90deg. rotation

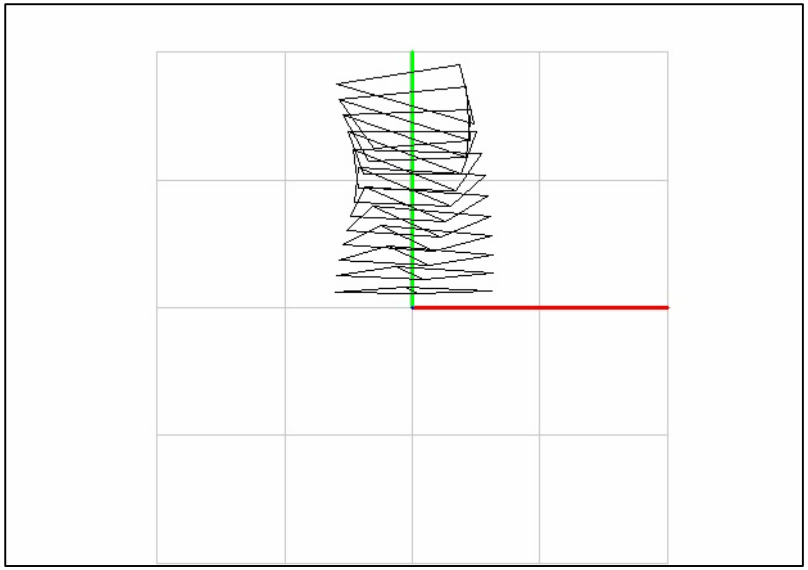


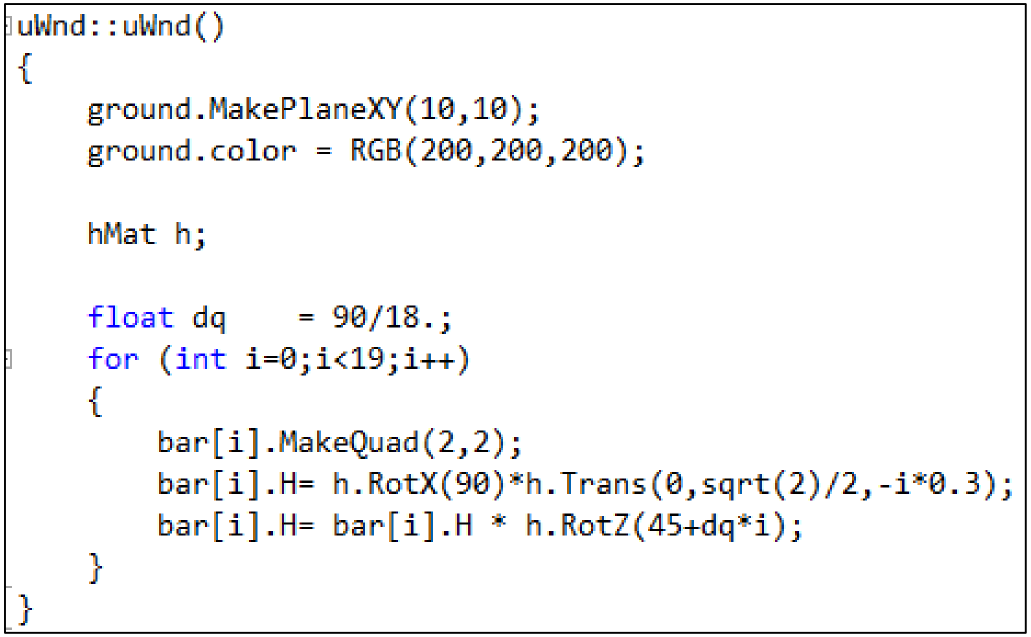
1st quad

$$\Delta\theta = \frac{90^\circ}{18}$$

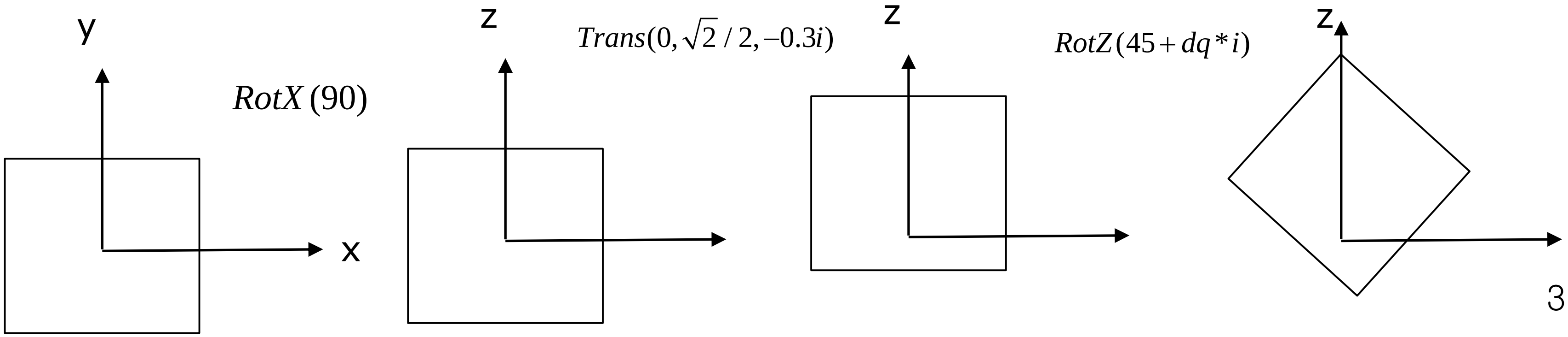
19 quad

Ex) uWnd-41-Sculp

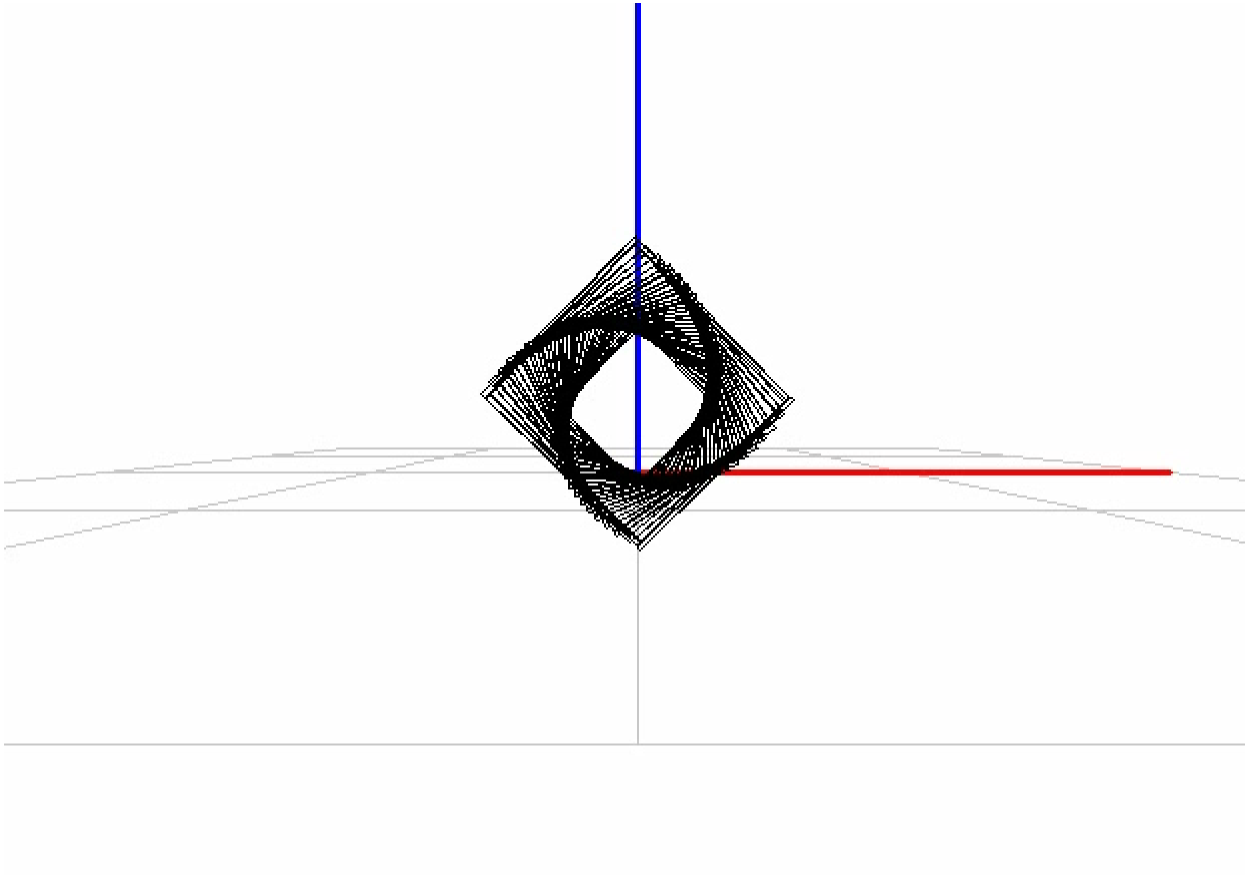




```
uWnd::uWnd()
{
    ground.MakePlaneXY(10,10);
    ground.color = RGB(200,200,200);

    hMat h;

    float dq    = 90/18.;
    for (int i=0;i<19;i++)
    {
        bar[i].MakeQuad(2,2);
        bar[i].H= h.RotX(90)*h.Trans(0,sqrt(2)/2,-i*0.3);
        bar[i].H= bar[i].H * h.RotZ(45+dq*i);
    }
}
```



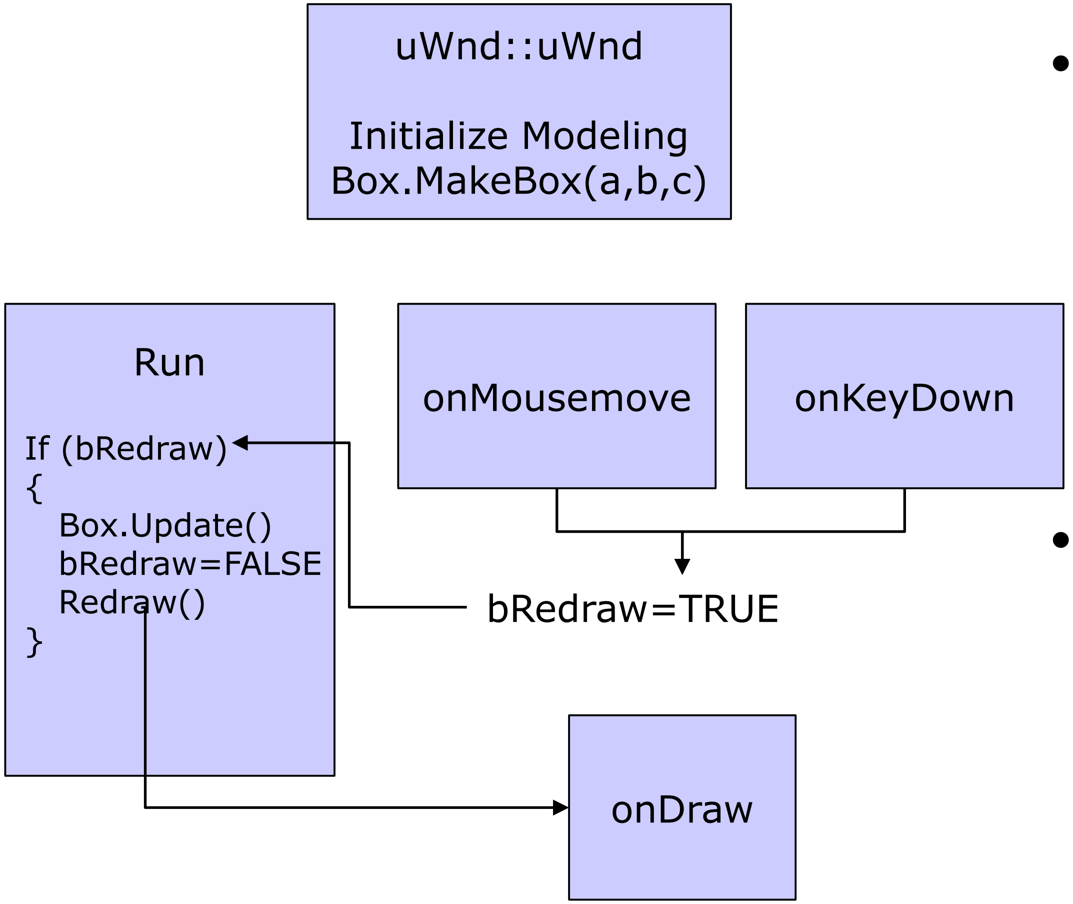
Example
uWnd-42-Sculp2-Ans



Clipping with Plane will be covered later

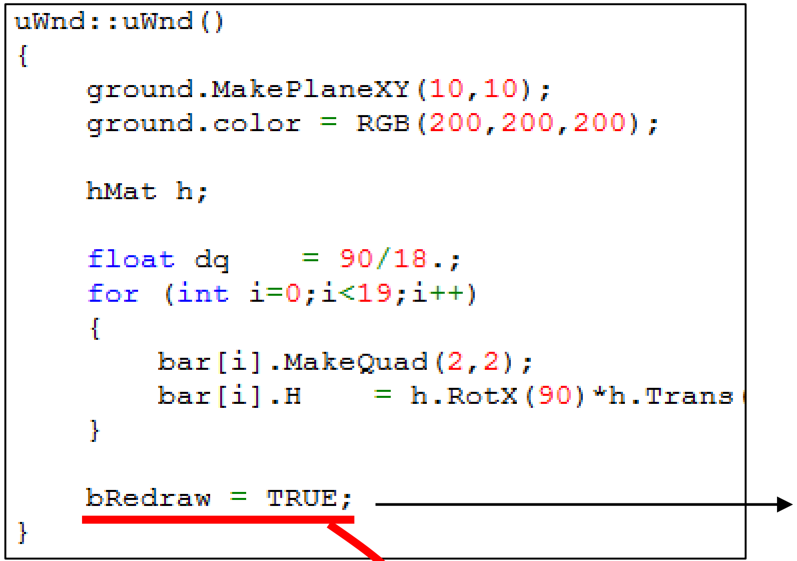
Event Programming for avoiding Flickering

uWnd-41-Sculp-Flickering



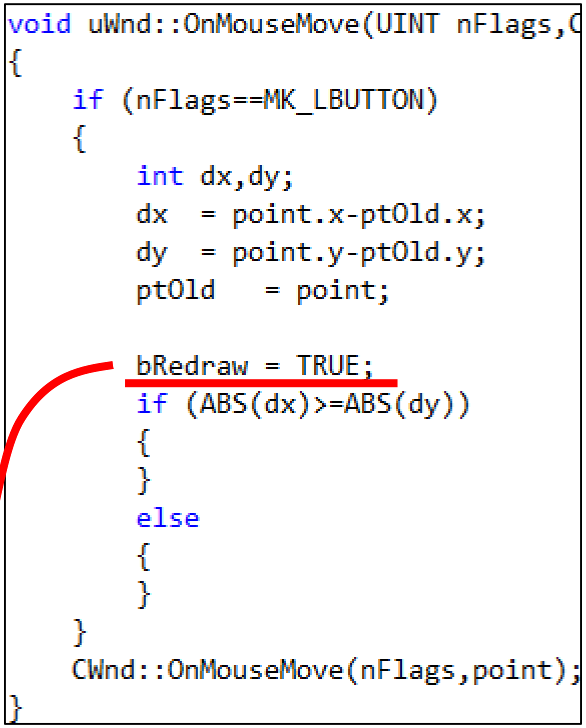
- Redraw() requires,
  - obj.update() for projection
  - Cam.R must be updated.

- Flag bRedraw is used
  - When mouse moves or key is pressed,
  - bRedraw = TRUE to wait for calling Redraw()

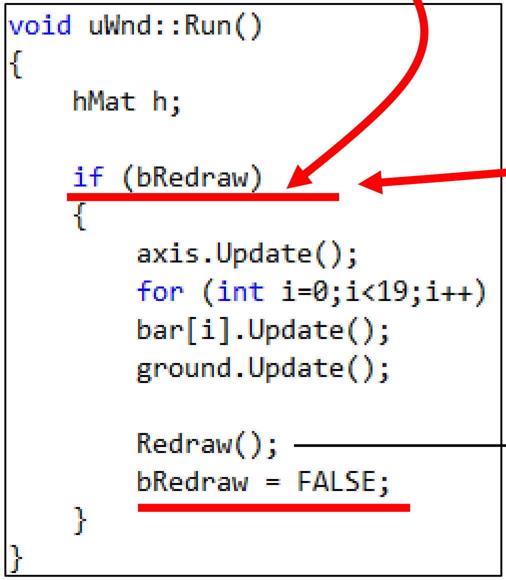


```
uWnd::uWnd()
{
    ground.MakePlaneXY(10,10);
    ground.color = RGB(200,200,200);

    hMat h;

    float dq    = 90/18.;
    for (int i=0;i<19;i++)
    {
        bar[i].MakeQuad(2,2);
        bar[i].H    = h.RotX(90)*h.Trans(
    }

    bRedraw = TRUE;
}
```

Run() will update objects



```
void uWnd::OnMouseMove(UINT nFlags,C
{
    if (nFlags==MK_LBUTTON)
    {
        int dx,dy;
        dx  = point.x-ptOld.x;
        dy  = point.y-ptOld.y;
        ptOld   = point;

        bRedraw = TRUE;
        if (ABS(dx)>=ABS(dy))
        {
        }
        else
        {
        }
    }
    CWnd::OnMouseMove(nFlags,point);
}
```



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

    if (bRedraw)
    {
        axis.Update();
        for (int i=0;i<19;i++)
        bar[i].Update();
        ground.Update();

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

Call onDraw()