Computer Graphics and Programming
Lecture 4

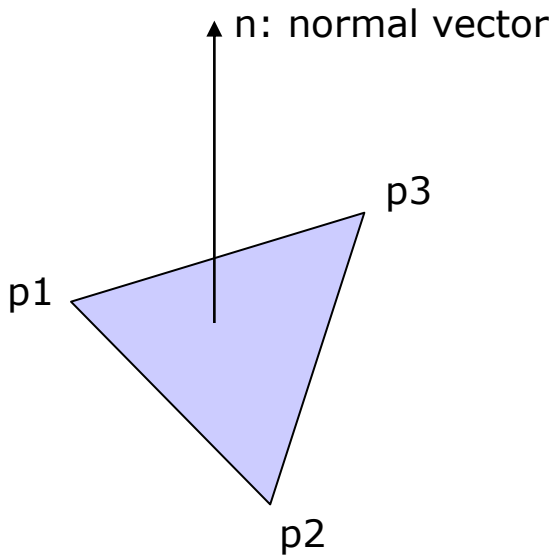
Jeong-Yean Yang

2020/10/22

1 Normal Vector

Normal Vector for
Hidden Surface Removal

- Three Basic Components in Graphics
  - Vertices
  - Polygon
  - Normal vector
- Normal vector is used for
  - Material and Light (95%)
  - Hidden Surface Removal (5%)
- Normal vector is very important.

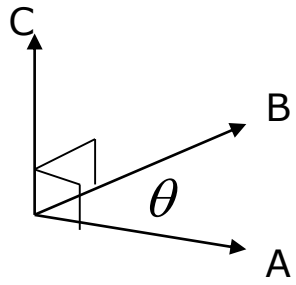


- Two types
  - 1) Calculation by three vertices
  - 2) Given for Light and Material

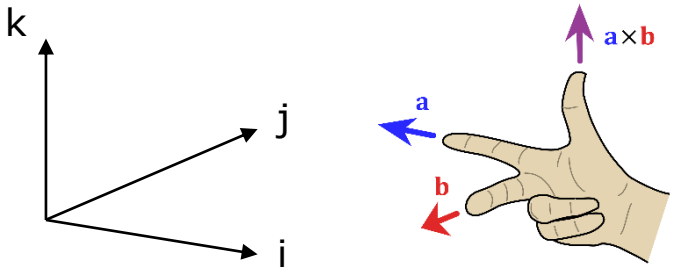
Ref.
Calculus, Spring

4. Vector Operation

- Cross Product (외적)



C=A X B



$$|C| = |A \times B| = |A||B|\sin\theta$$

|i|=1, |j|=1, |k|=1

k=ixj , i=jxk , j = kxi

-k=jxi , -i=kxj , -j = ixk

- Determinant of AxB

$$A \times B = \begin{vmatrix} i & j & k \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix} = \left(A_y B_z - A_z B_y\right) i + \left(A_z B_x - A_x B_z\right) j + \left(A_x B_y - A_y B_x\right) k$$

Ref.
Calculus, Spring

Determinant of A x B

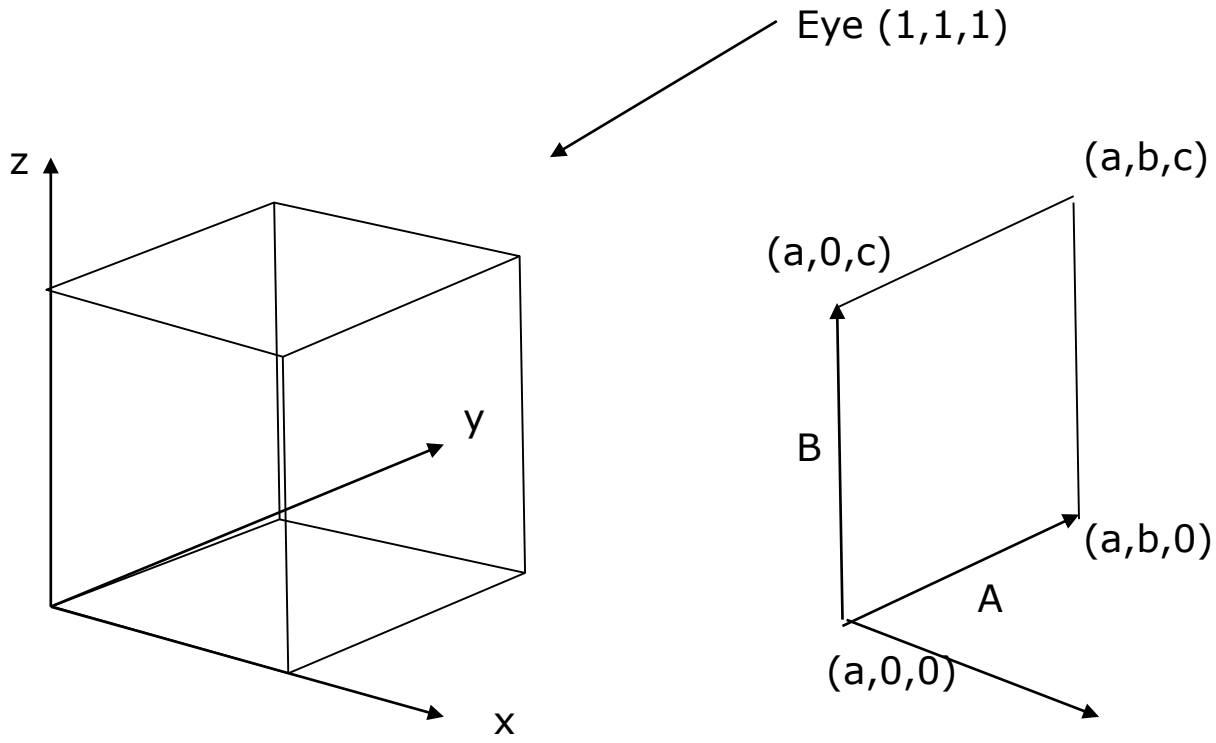
$$A \times B = \begin{vmatrix} i & j & k \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix} = i \begin{vmatrix} A_y & A_z \\ B_y & B_z \end{vmatrix} - j \begin{vmatrix} A_x & A_z \\ B_x & B_z \end{vmatrix} + k \begin{vmatrix} A_x & A_y \\ B_x & B_y \end{vmatrix}$$

$$= \left(A_y B_z - A_z B_y\right) i + \left(A_z B_x - A_x B_z\right) j + \left(A_x B_y - A_y B_x\right) k$$

- Cross Product, AxB
  - i x i = j x j = k x k = 0
- Why i x i =0 ?
  - |C| = |AxB| = |A||B| sin q
  - |C| = |AxA| = |A||A| sin 0 = 0

Cross Product for What?

- Example) Hidden surface removal



A = (a,b,0)-(a,0,0)
= (0,b,0)

B = (a,0,c)-(a,0,0)
= (0,0,c)

A x B = (bc,0,0)

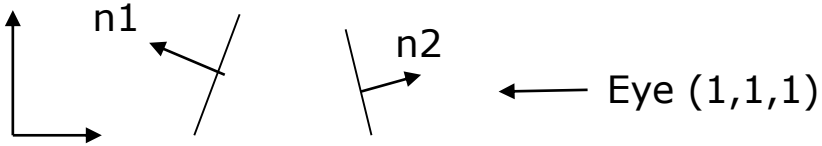
Eye position $= (1,1,1)$

$v = (0,0,0) - (1,1,1)$

$= (-1,-1,-1)$

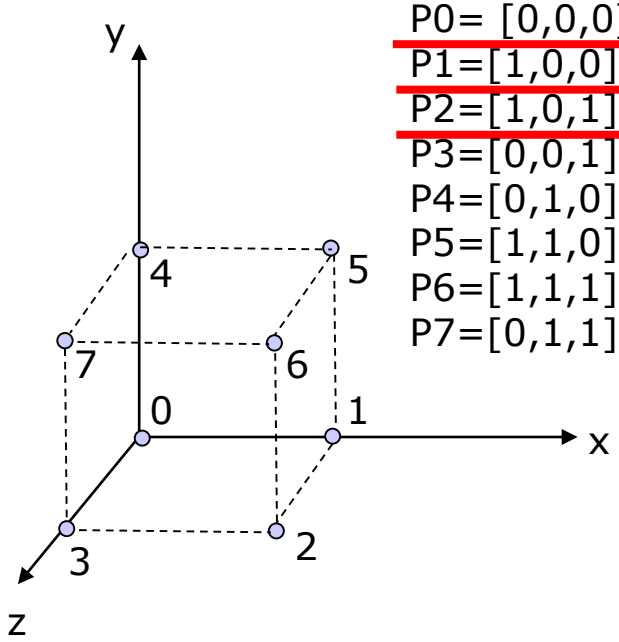
$v \cdot (A \times B) = -bc < 0$

show!



Cross Product of Cube's Polygon

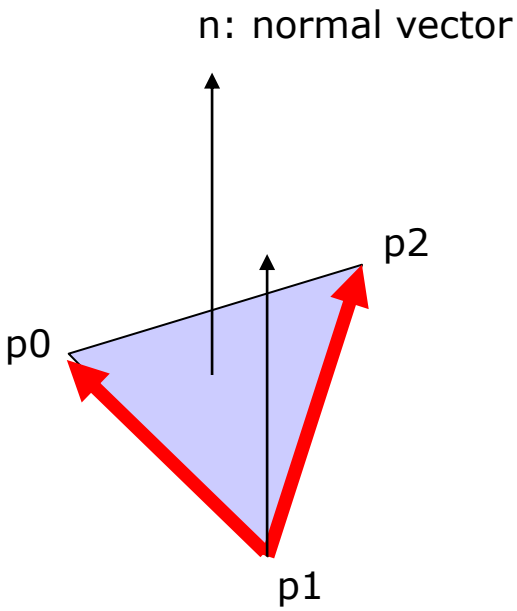
- Remind



P0= [0,0,0]
P1=[1,0,0]
P2=[1,0,1]
P3=[0,0,1]
P4=[0,1,0]
P5=[1,1,0]
P6=[1,1,1]
P7=[0,1,1]

Polygon 0=[0,1,2]
Polygon 1=[0,2,3]
Polygon 2=[6,2,1]
Polygon 3=[6,1,5]
Polygon 4=[4,0,3]
Polygon 5=[4,3,7]

Polygon 6=[7,3,2]
Polygon 7 =[7,2,6]
Polygon 8=[5,1,0]
Polygon 9 =[5,0,4]
Polygon 10=[4,7,6]
Polygon 11=[4,6,5]



$$\hat{B} = p_0 - p_1 \ , \ \hat{A} = p_2 - p_1$$

$$\therefore \hat{A} \times \hat{B} = \left(p_2 - p_1 \right) \times \left(p_0 - p_1 \right)$$

Cross Product in uVector

- Use operator *

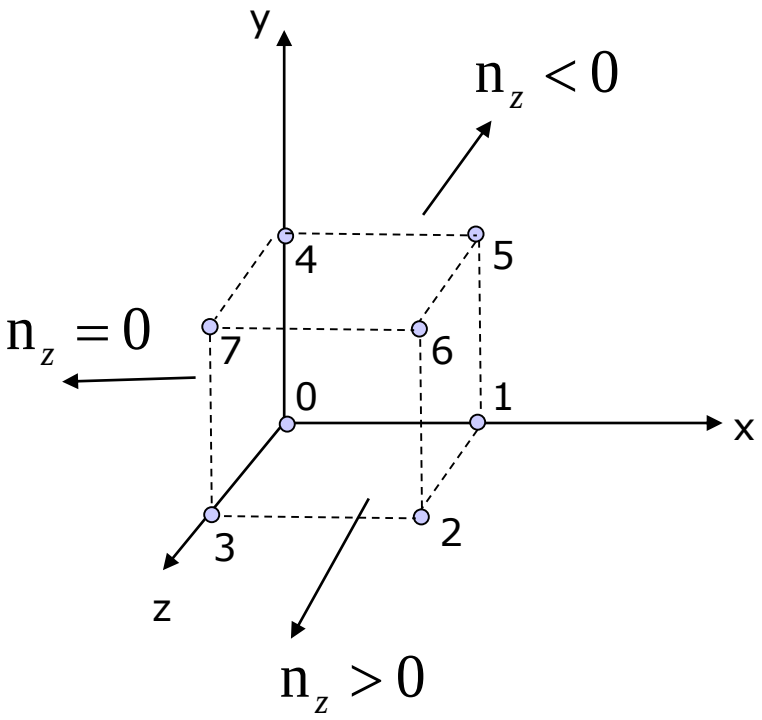
$$A \times B = \left(A_y B_z - A_z B_y\right) i + \left(A_z B_x - A_x B_z\right) j + \left(A_x B_y - A_y B_x\right) k$$

```
uVector uVector::operator*(uVector u)
{
    uVector ret;
    ret.x   = y*u.z-z*u.y;
    ret.y   = z*u.x-x*u.z;
    ret.z   = x*u.y-y*u.x;
    return ret;
}
```

- Example of Cross Product of uVector (HW6)

```
uVector f(1,2,3);
uVector s(2,3,4);

uVector c = f*s;
uVector d = s*f;
```

Example
uWnd-21-Hidden surface



- Compare

  Normal vector and Z vector

$$\hat{n} = \hat{A} \times \hat{B} = (p_2 - p_1) \times (p_0 - p_1)$$

$$\therefore \hat{n} = [n_x, n_y, n_z]$$

if $n_z > 0$:

Draw

else:

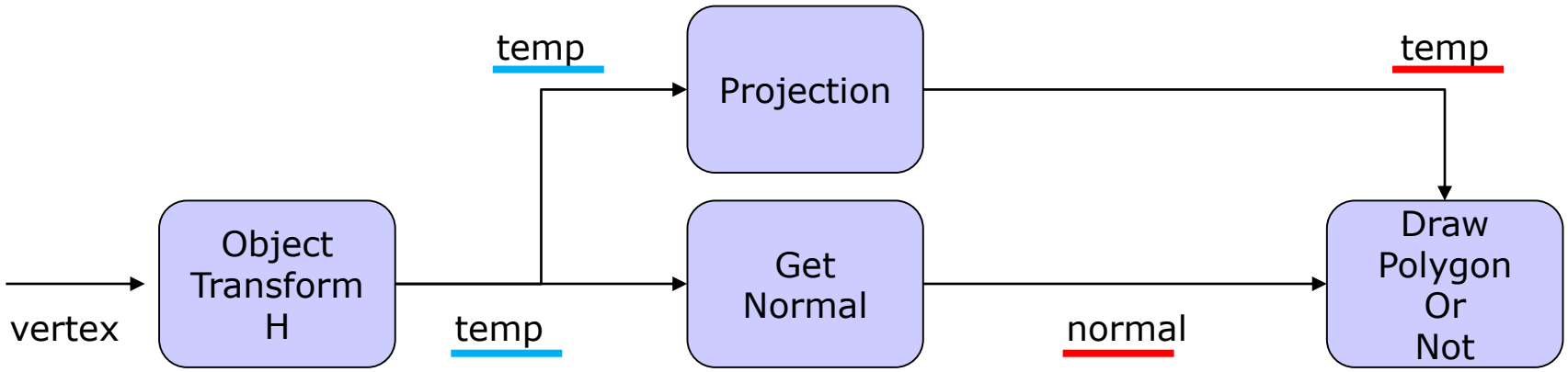
pass

2 Camera Work

uWnd-21-Hidden surface Operation Architecture

- uObj::Draw()

```
// projection
for (i=0;i<nMax;i++)
temp[i] = pCam->Projection(temp[i]);
```




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

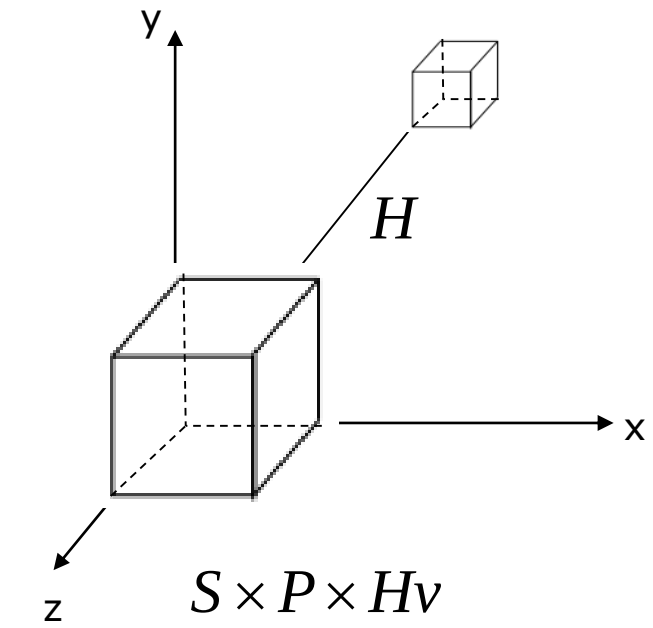
Object is rotating

```
// norma CCW
normal[0]   = (temp[2]-temp[1])*(temp[0]-temp[1]);
normal[1]   = (temp[3]-temp[2])*(temp[0]-temp[2]);
normal[2]   = (temp[1]-temp[2])*(temp[6]-temp[2]);
normal[3]   = (temp[5]-temp[1])*(temp[6]-temp[1]);
```

```
if (normal[0].z>0)
    DrawPolygon(pDC, temp[0],temp[1],temp[2]);
if (normal[1].z>0)
    DrawPolygon(pDC, temp[0],temp[2],temp[3]);
```


Object Transform Vs. Camera Work
ex) uWnd-22-Camera Work

- Object Transform



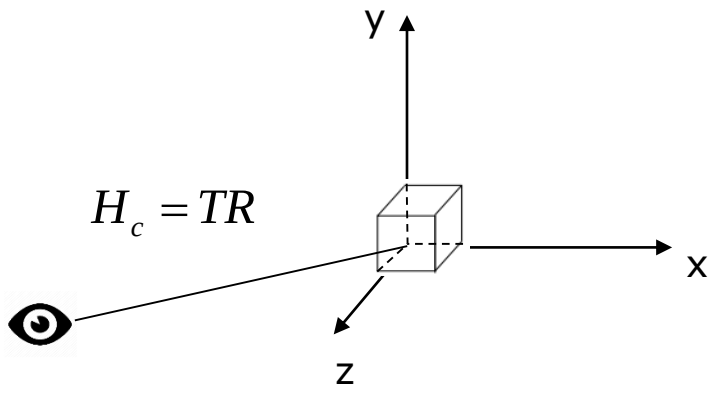
$$S \times P \times Hv$$

$$\equiv S \times P \times v'$$

Scale projection

Object transform

Camera Work



$$S \times P \times TR \times Hv$$

$$= S \times P \times H_c \times v'$$

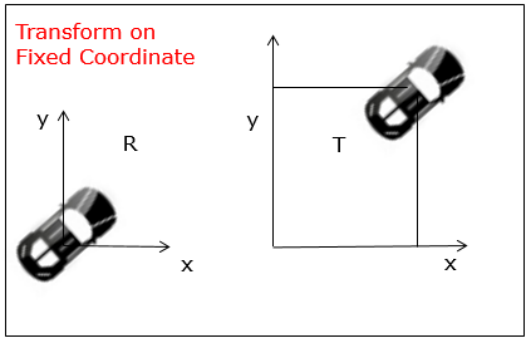
Camera Work: Rotation and Translation Hc=TR in uCam

```
uVector uCam::Projection(uVector t)
{
    // Camera Framework
    t   = R*t;
    t   = T*t;

    // Projection
    float z = t.z;
    t   = P*t;
    t   = t*(-1./z);
    t   = S*t;
    return t;
}
```

```
// T and R
T    = uVector(0,0,-10);
```

```
void uWnd::Run()
{
    hMat h,s;
    cam.R   = h.RotY(cam.q.y);
    cam.q.y+=5;
    Redraw();
}
```

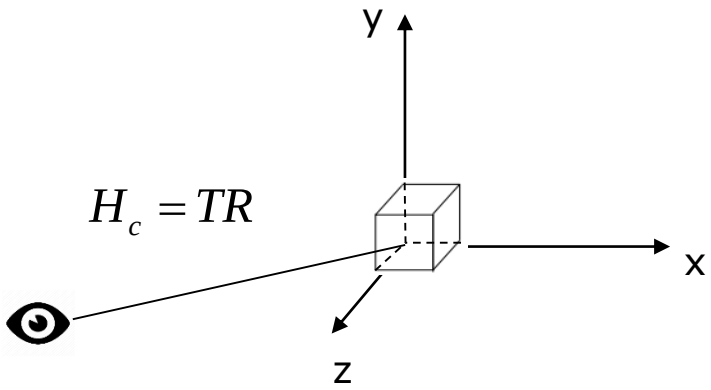
- Remind Fixed Coordinate
  - Fixed coordinate

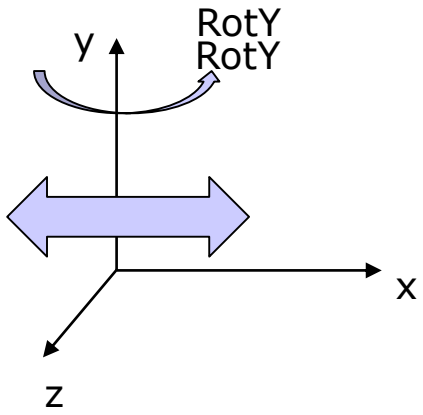


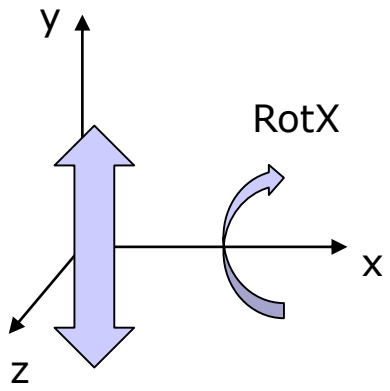
$$H_T \longrightarrow H_R$$

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

Example and HW 7







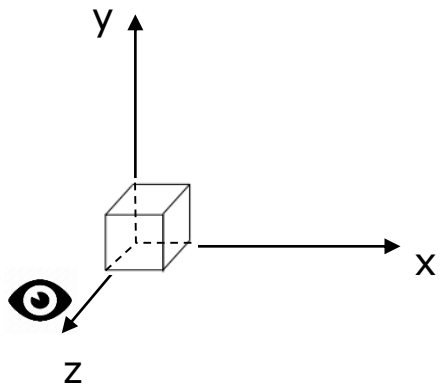
$$S \times P \times TR \times Hv$$
$$= S \times P \times H_c \times v'$$

$$H_c = TR = TR_Y R_X$$

- Click Left mouse button and drag.
  - Left and right direction rotates along Y-axis
  - Up and Down direction rotates along X-axis

Hidden Surface Removal with Camera Work Is it Good?

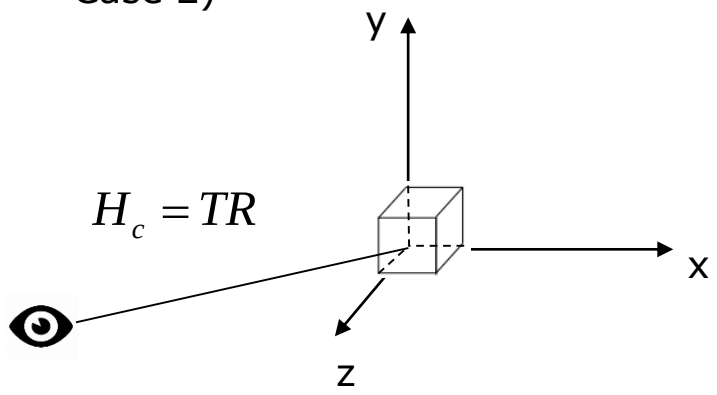
Case 1)



$n_z > 0$ for Draw

$\therefore \hat{z} \bullet \hat{n} > 0$

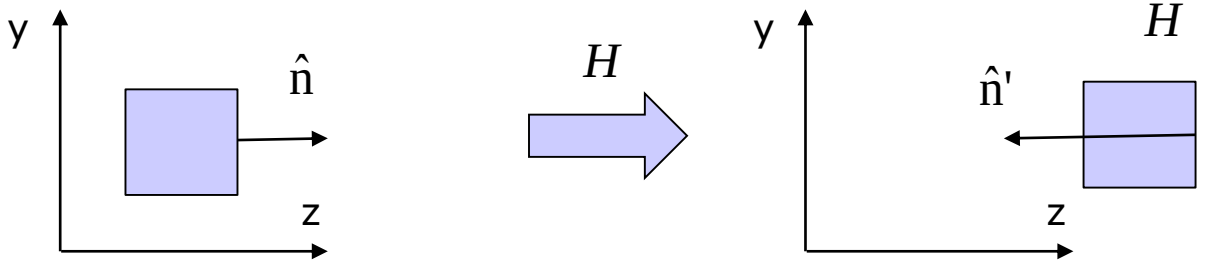
Case 2)



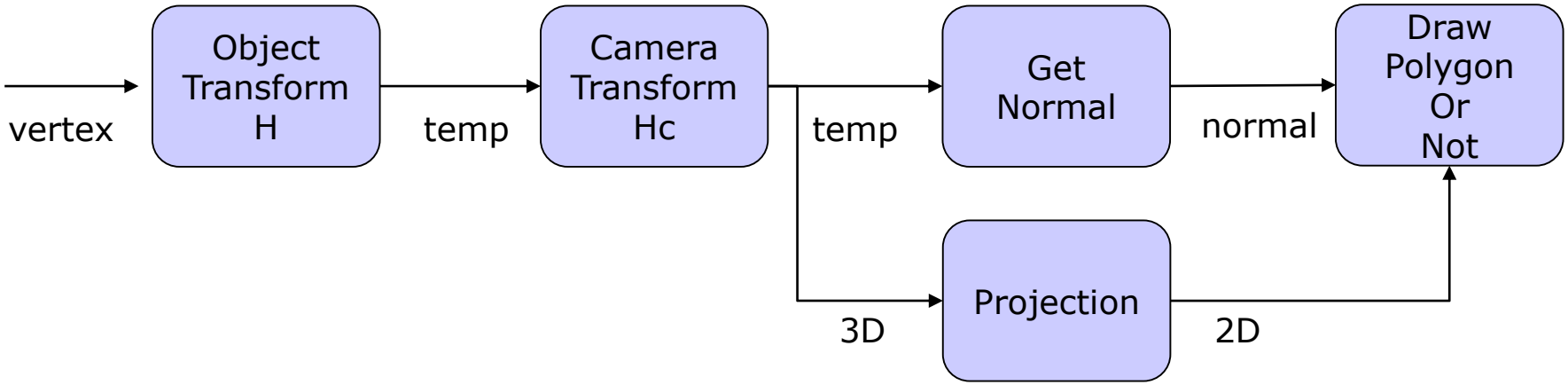
$H_c = TR$

$\hat{z} \bullet H_c \hat{n}$ (Bad)

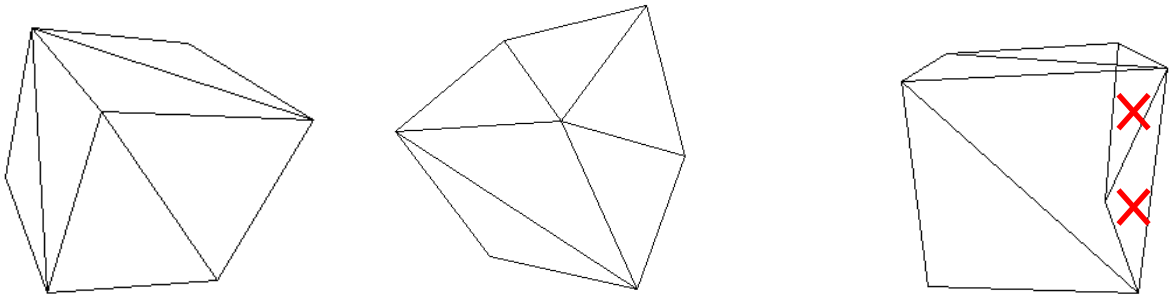
- Normal vector is a directional vector.
- **Transform of Direction vector is wrong.**



Hidden Surface Removal:
Bad case for Perspective Projection



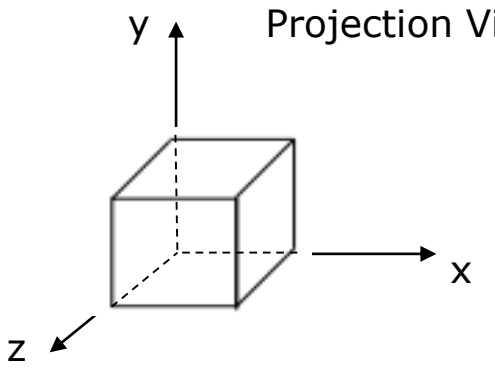
- Example) uWnd-25-Camera Walk2-Hidden

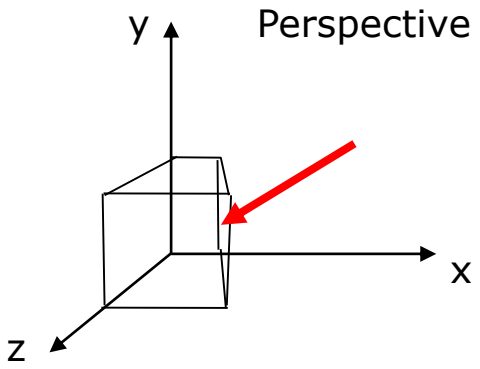


good

Bad

Why it is Wrong?



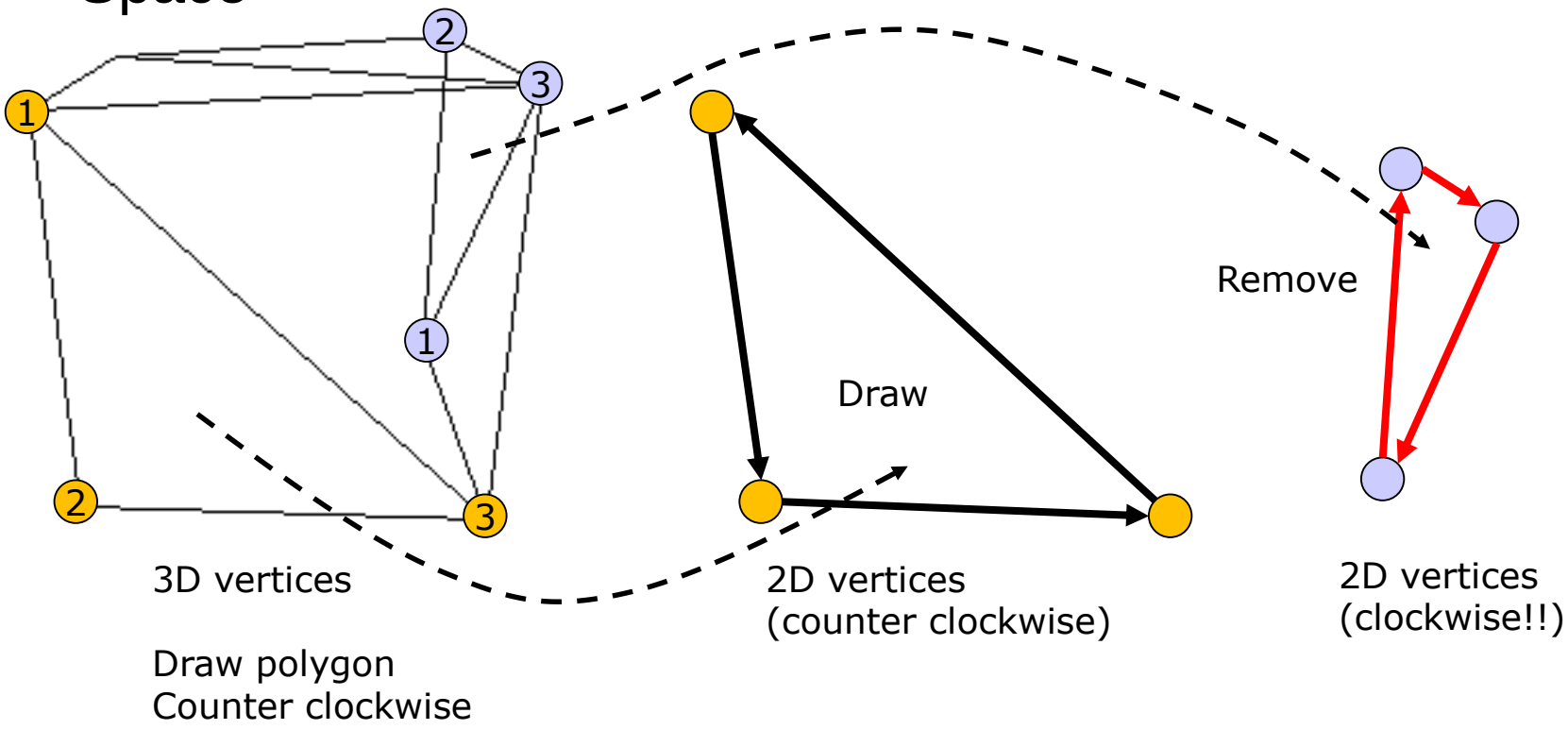


$$\hat{n}' \neq H_c \hat{n}$$
$$\neq H_c\left((p_0 - p_1)\times(p_2 - p_1)\right)$$
$$\rightarrow \hat{n}' = (H_c p_0 - H_c p_1)\times(H_c p_2 - H_c p_1)$$

- **Normal vectors are same in both cases**
- But, Perspective projection has skewed and warped plane.

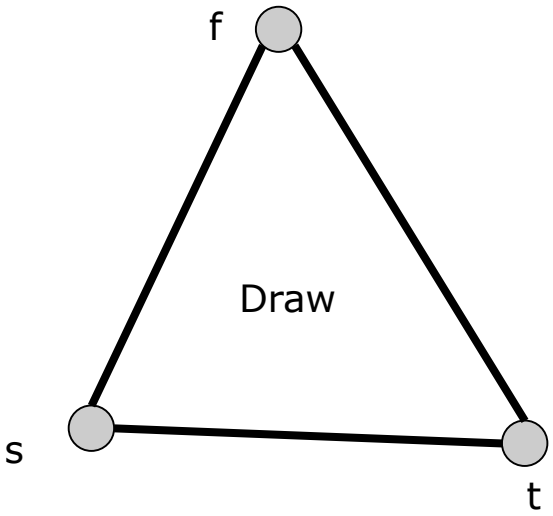
Hidden Surface Removal for Perspective Projection

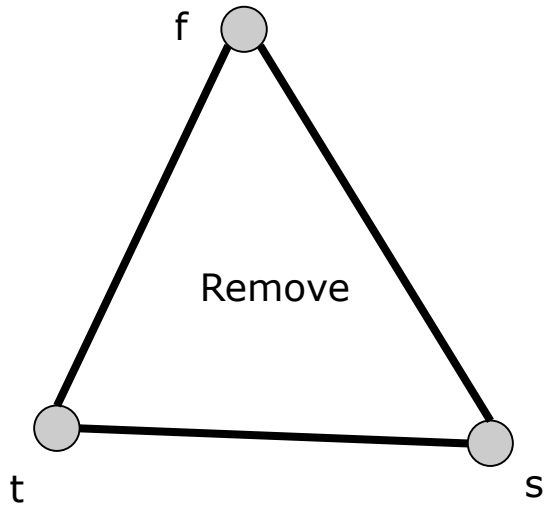
- Counter Clockwise Direction in 3D is changed in 2D Space



- Drawing Clockwise is removed

Normal Vector in Projected 2D Space Case of Drawing Counter Clockwise





$$\hat{B} = f - s \;,\; \hat{A} = t - s$$

$$\hat{A} \times \hat{B} = (t - s) \times (f - s)$$

$$\hat{z} \bullet \hat{A} \times \hat{B} > 0$$

$$\hat{B} = f - s \;,\; \hat{A} = t - s$$

$$\hat{A} \times \hat{B} = (t - s) \times (f - s)$$

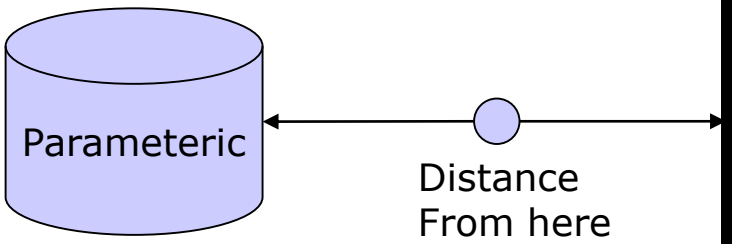
$$\hat{z} \bullet \hat{A} \times \hat{B} < 0$$

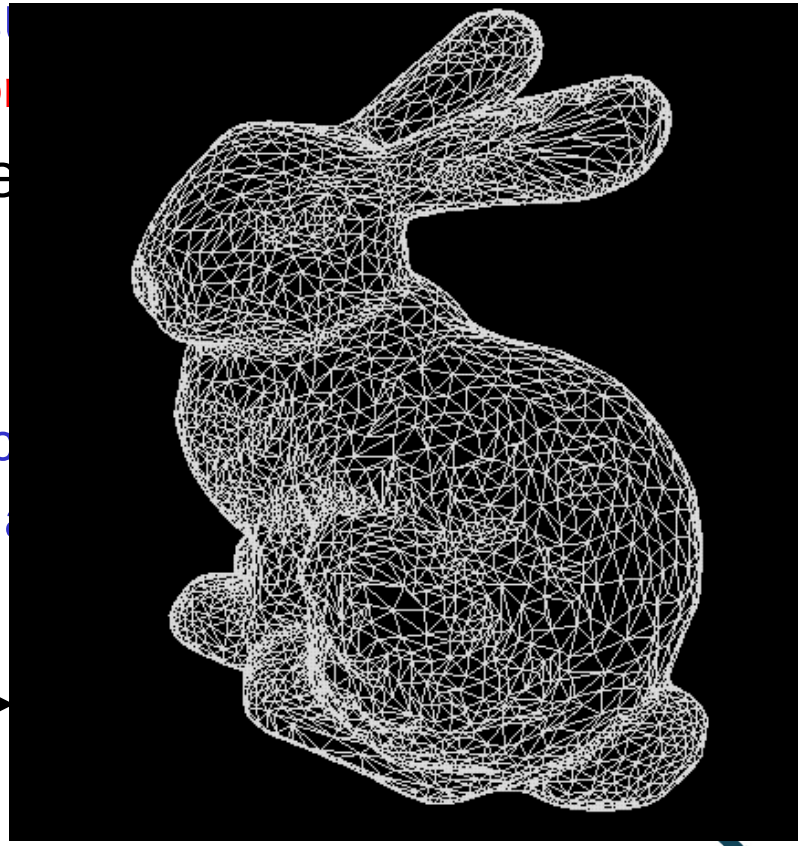
- Example) uWnd-26-Camera Walk3-Hidden

3 Object Designs

Graphic Primitives

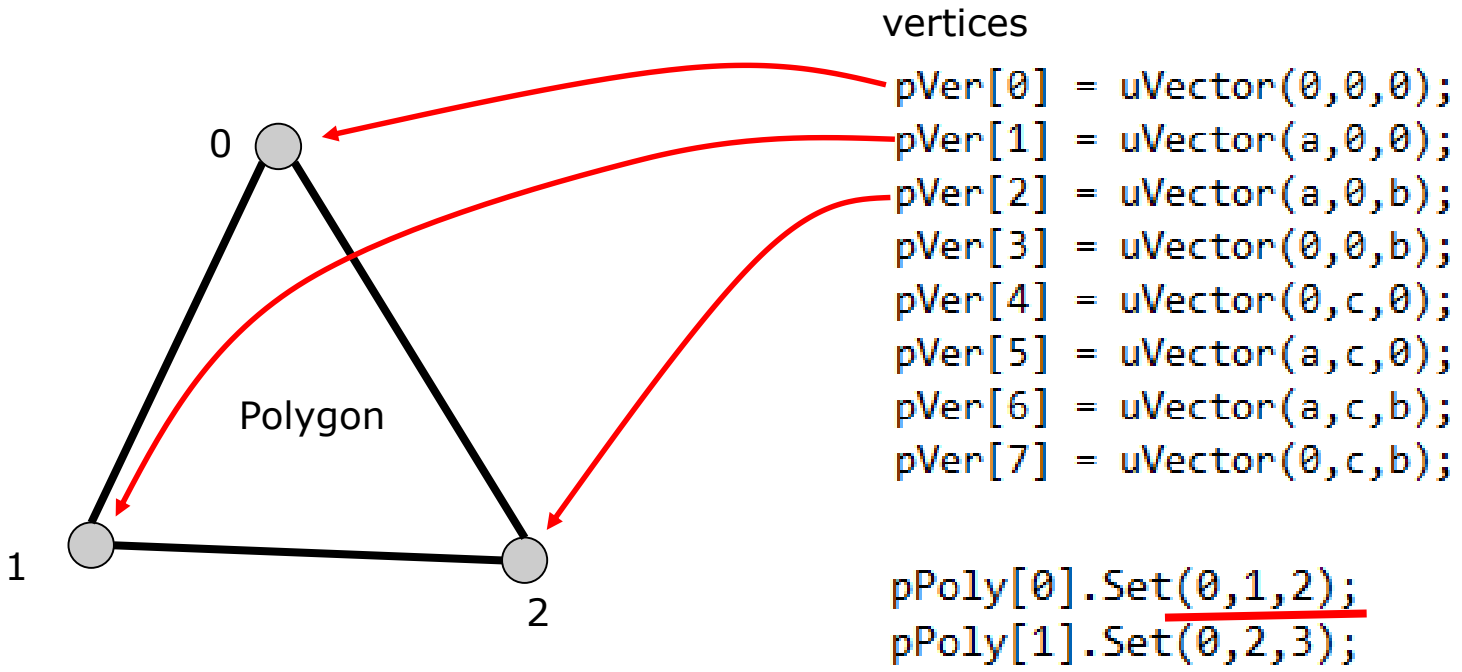
- Primitives
  - Box, Plane, Sphere, Donut, Tube, and so on
  - The shapes are NOT fantastic, B
    mathematical calculation of so
- Primitives = Parametric Space
- Parametric Vs Polygon-based
  - Parametric object is good for calc
  - Polygon takes long time of calcul





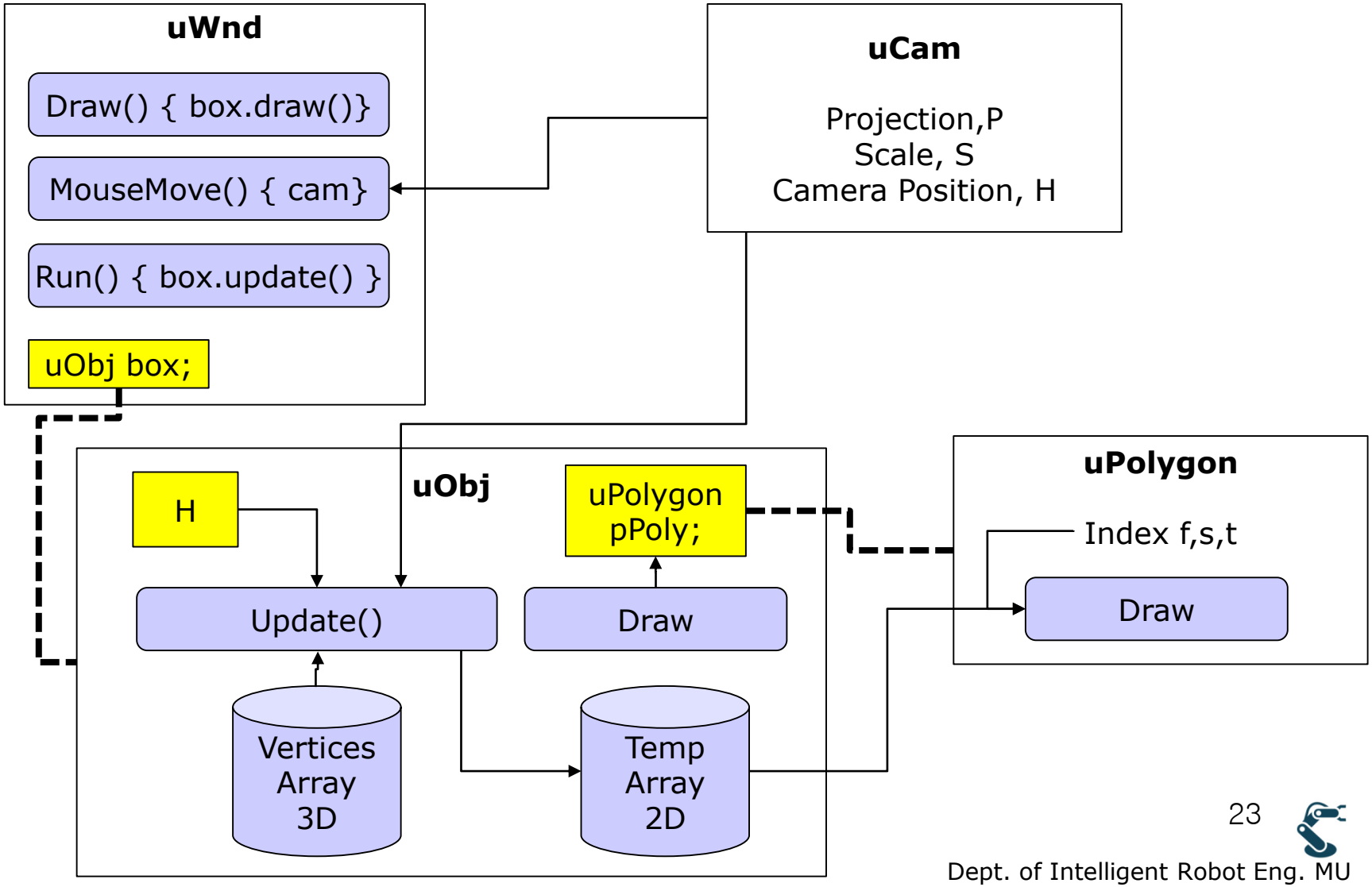
New Class : uPolygon

- uPolygon: Polygon class with three vertices index
  - What is INDEX?



  - Polygon has three indices that are the offsets of vertices buffer

Ex) uWnd-27-class-Polygon-Complete uWnd-uObj-uCam-uPolygon



uObj::Update()

- Update() function calculates
  - original 3D vertex(pVer) into 2D projected vertex(pTemp)

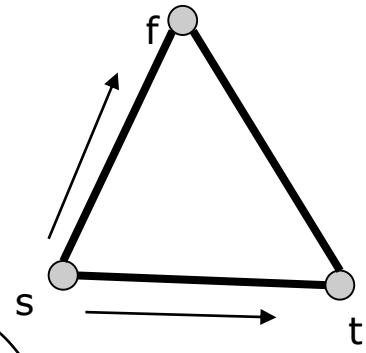
```
// original data
uVector      *pVer;
uVector      *pTemp;
uPolygon     *pPoly;
```

- uObj has Object Drawing and Vertex Calculation

uObj::Update()

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

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

    // calculate normal vector
    for (i=0;i<nPoly;i++)
    {
        int f,s,t;
        f   = pPoly[i].f;
        s   = pPoly[i].s;
        t   = pPoly[i].t;

        uVector A   = pTemp[t]-pTemp[s];
        uVector B   = pTemp[f]-pTemp[s];

        if ((A*B).z>0)  pPoly[i].bDraw = TRUE;
        else            pPoly[i].bDraw  = FALSE;
    }
}
```

$$t_2 = S_{cam} P_{cam} H_{cam} H_{object} v_3$$



$$\hat{n} = \hat{A} \times \hat{B} = (t - s) \times (f - s)$$

$$\hat{z} \bullet \hat{A} \times \hat{B} > 0$$

Dynamic Memory Allocation
new and delete from C++

```
uObj::uObj()
{
    pVer    = NULL;
    pTemp   = NULL;
    pPoly   = NULL;
}

uObj::~uObj()
{
    Close();
}
```

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

uObj.h

```
void uObj::Alloc(int nv,int np)
{
    Close();

    nVer    = nv;
    nPoly   = np;

    pVer    = new uVector[nv];
    pTemp   = new uVector[nv];
    pPoly   = new uPolygon[np];
}
```

```
void uObj::Close()
{
    if (pVer)   delete pVer;
    if (pPoly)  delete pPoly;
    if (pTemp)  delete pTemp;

    pVer    = NULL;
    pPoly   = NULL;
    pTemp   = NULL;
}
```

Cube has 8 vertices and 12 polygons.

Cylinder has 36x2 vertices and 72 polygons

Numbers of Vertices and Polygon are variable.

→ **Dynamic Memory Allocation**

Dynamic Memory Allocation
new and delete from C++

```
uObj::uObj()
{
    pVer    = NULL;
    pTemp   = NULL;
    pPoly   = NULL;
}

uObj::~uObj()
{
    Close();
}
```

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

uObj.h

```
void uObj::Alloc(int nv,int np)
{
    Close();

    nVer    = nv;
    nPoly   = np;

    pVer    = new uVector[nv];
    pTemp   = new uVector[nv];
    pPoly   = new uPolygon[np];
}
```

uObj box; → box.pVer = NULL

```
void uObj::Close()
{
    if (pVer)   delete pVer;
    if (pPoly)  delete pPoly;
    if (pTemp)  delete pTemp;

    pVer    = NULL;
    pPoly   = NULL;
    pTemp   = NULL;
}
```


Dynamic Memory Allocation
new and delete from C++

```
uObj::uObj()
{
    pVer    = NULL;
    pTemp   = NULL;
    pPoly   = NULL;
}

uObj::~uObj()
{
    Close();
}
```

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

uObj.h

```
void uObj::Alloc(int nv,int np)
{
    Close();

    nVer    = nv;
    nPoly   = np;

    pVer    = new uVector[nv];
    pTemp   = new uVector[nv];
    pPoly   = new uPolygon[np];
}
```

uObj box;

box.Alloc(8,12)

box.pVer = NULL

Nothing done

```
void uObj::Close()
{
    if (pVer)   delete pVer;
    if (pPoly)  delete pPoly;
    if (pTemp)  delete pTemp;

    pVer    = NULL;
    pPoly   = NULL;
    pTemp   = NULL;
}
```


Dynamic Memory Allocation
new and delete from C++

```
uObj::uObj()
{
    pVer    = NULL;
    pTemp   = NULL;
    pPoly   = NULL;
}

uObj::~uObj()
{
    Close();
}
```

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

uObj.h

```
void uObj::Alloc(int nv,int np)
{
    Close();

    nVer    = nv;
    nPoly   = np;

    pVer    = new uVector[nv];
    pTemp   = new uVector[nv];
    pPoly   = new uPolygon[np];
}
```

```
void uObj::Close()
{
    if (pVer)   delete pVer;
    if (pPoly)  delete pPoly;
    if (pTemp)  delete pTemp;

    pVer    = NULL;
    pPoly   = NULL;
    pTemp   = NULL;
}
```

uObj box;

box.Alloc(8,12)

box.pVer = NULL
box.pTemp=NULL
box.pPoly = NULL

box.pVer[8]

box.pTemp[8]

box.pPoly[12]

After using uObj,
Created Memory MUST be Deleted

```
uObj::uObj()
{
    pVer    = NULL;
    pTemp   = NULL;
    pPoly   = NULL;
}

uObj::~uObj()
{
    Close();
}
```

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

uObj.h

```
void uObj::Alloc(int nv,int np)
{
    Close();

    nVer    = nv;
    nPoly   = np;

    pVer    = new uVector[nv];
    pTemp   = new uVector[nv];
    pPoly   = new uPolygon[np];
}
```

```
void uObj::Close()
{
    if (pVer)   delete pVer;
    if (pPoly)  delete pPoly;
    if (pTemp)  delete pTemp;

    pVer    = NULL;
    pPoly   = NULL;
    pTemp   = NULL;
}
```

uObj box;

box.Alloc(8,12) ⟶ box.pVer[8]

C++ starts to destroy box

box.pVer[8]!=NULL

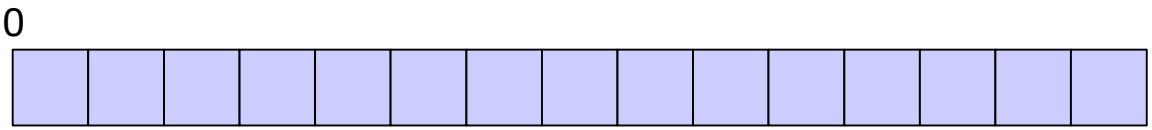
delete box.pVer
→ box.pVer[ ]
→ No 8 Storage

Allocation of Pointer Variable in C++

Initial state

uVector *p; → Pointer variable

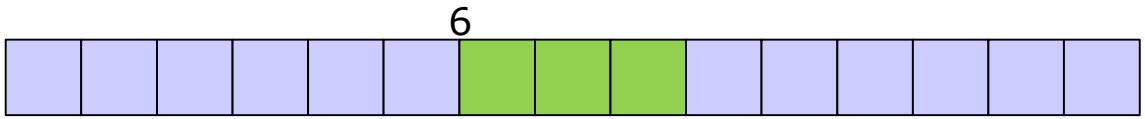
p=NULL; → P has NO storage



Application Heap memory

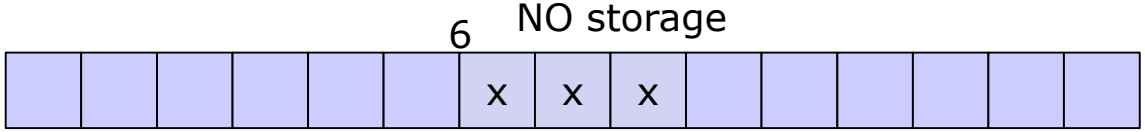
"new" in C++

p = new uVector [3]; → p=6; P has 3 storages



"delete" in C++

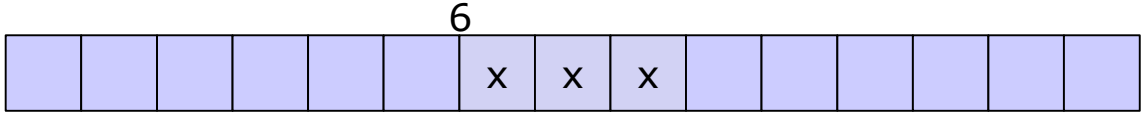
delete p; → p=6; **Confusion!!!**



Be careful!!
Don't use like p[0]=1

Remark p= NULL

p=NULL; → p=0; P has NO storage

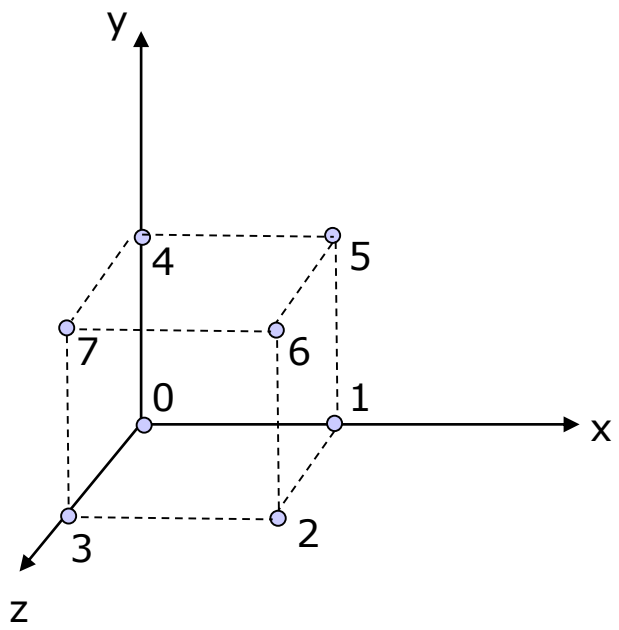


Ex 1) uObj::MakeBox(1,2,3)

```
void uObj::MakeBox(float a,float b,float c)
{
    Alloc(8,12);

    pVer[0] = uVector(0,0,0);
    pVer[1] = uVector(a,0,0);
    pVer[2] = uVector(a,0,b);
    pVer[3] = uVector(0,0,b);
    pVer[4] = uVector(0,c,0);
    pVer[5] = uVector(a,c,0);
    pVer[6] = uVector(a,c,b);
    pVer[7] = uVector(0,c,b);

    pPoly[0].Set(0,1,2);
    pPoly[1].Set(0,2,3);
    pPoly[2].Set(6,2,1);
    pPoly[3].Set(6,1,5);
    pPoly[4].Set(4,0,3);
    pPoly[5].Set(4,3,7);
    pPoly[6].Set(7,3,2);
    pPoly[7].Set(7,2,6);
    pPoly[8].Set(5,1,0);
    pPoly[9].Set(5,0,4);
    pPoly[10].Set(4,7,6);
    pPoly[11].Set(4,6,5);
}
```

8 vertices

12 polygons



Ex 2) Make Box by Arrays

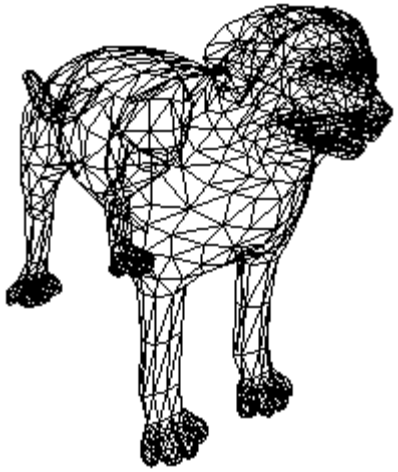
```
float vs[]=
{
    0,0,0,
    1,0,0,
    1,0,2,
    0,0,2,
    0,3,0,
    1,3,0,
    1,3,2,
    0,3,2
};

int ps[]=
{
    0,1,2,
    0,2,3,
    6,2,1,
    6,1,5,
    4,0,3,
    4,3,7,
    7,3,2,
    7,2,6,
    5,1,0,
    5,0,4,
    4,7,6,
    4,6,5
};
```

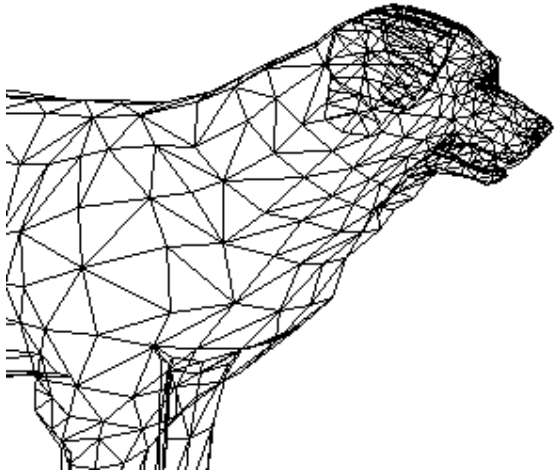
```
uWnd::uWnd()
{
    //box.MakeBox(1,2,3);

    int nv = sizeof(vs)/sizeof(float);
    int np = sizeof(ps)/sizeof(int);
    box.Alloc(nv/3,np/3);

    int i,n=0;

    // load vertices
    for (i=0;i<nv;i+=3)
    {
        float x = vs[i];
        float y = vs[i+1];
        float z = vs[i+2];
        box.pVer[n] = uVector(x,y,z);
        n++;
    }

    // load polygon
    n = 0;
    for (i=0;i<np;i+=3)
    {
        int f = ps[i];
        int s = ps[i+1];
        int t = ps[i+2];
        box.pPoly[n].f = f;
        box.pPoly[n].s = s;
        box.pPoly[n].t = t;
        n++;
    }
}
```

Remind that
sizeof(vs)=3*float*8=3*4*8 = 96
sizeof(ps)=3*int*8 = 3*4*12=144

Then, if we change vertices and polygons, Everything can be rendered in Graphics

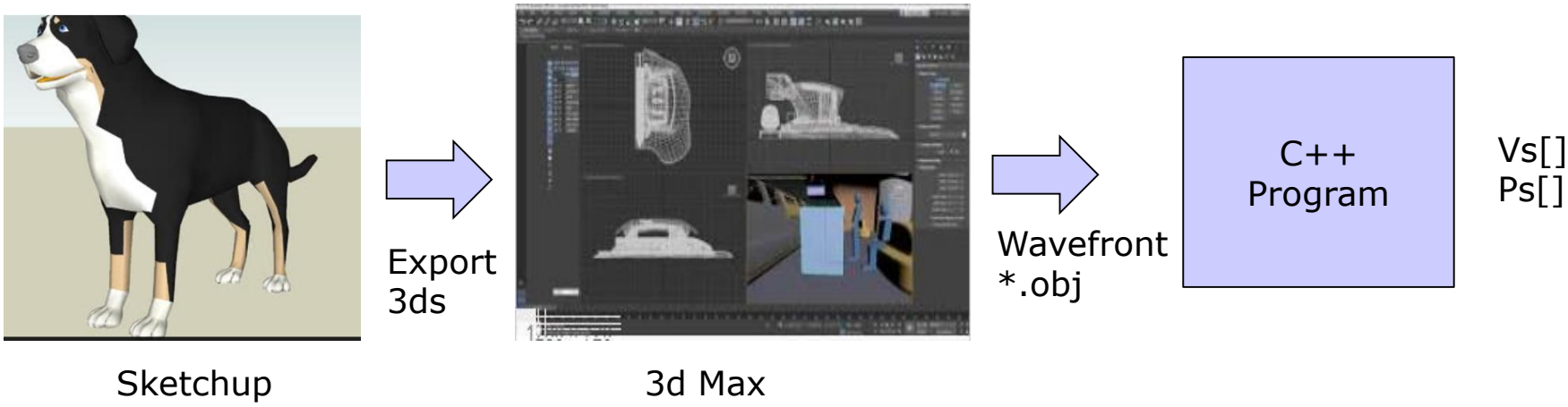
- Vertices = 1489, Polygons= 2974
- See uWnd-29-complex





Example) Draw a Dog

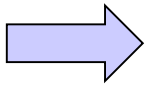
- https://3dwarehouse.sketchup.com/
- https://3dwarehouse.sketchup.com/model/u24f15eea-ce38-414e-ae9b-512ccbda2eb8/dog



- See dog.obj file in example

Wavefront Object file, *.obj

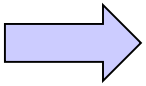
```
# 3ds Max Wavefront OBJ Exporter v0.97b - (c)2007 guruware
# File Created: 24.09.2019 18:10:54

#
# object Mesh01
#

v  595.2358 -1310.7905 -920.3279
v  666.1851 -1311.4379 -907.4728
v  595.2358 -1470.6831 -885.7084
v  595.2358 -1133.5787 -941.3796
v  665.4882 -1139.1316 -926.1271
```



Vertex vector

v X Y Z

```
g Mesh01
f 1 2 3
f 4 2 1
f 4 5 2
f 6 5 4
f 6 7 5
f 6 8 7
f 9 8 6
```



Polygon index

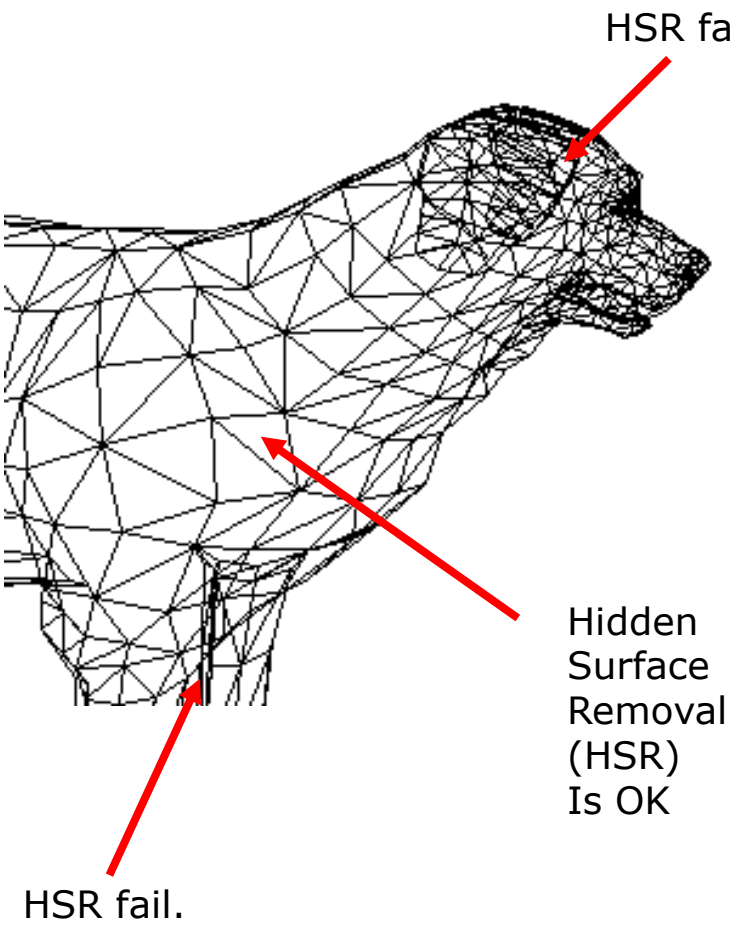
f f s t

Be careful!
It is NOT zero index. Start from 1.
But, general array has zero index that starts from 0.

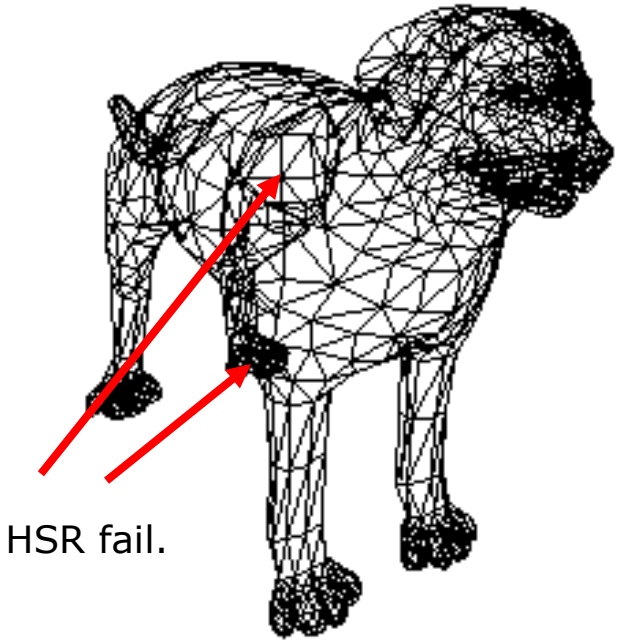
```
// load polygon
n = 0;
for (i=0;i<np;i+=3)
{
    int f = ps[i]-1;
    int s = ps[i+1]-1;
    int t = ps[i+2]-1;
    box.pPoly[n].f = f;
    box.pPoly[n].s = s;
    box.pPoly[n].t = t;
    n++;
}
```

-1 for zero index

But, Something is Wrong in Dog Drawing

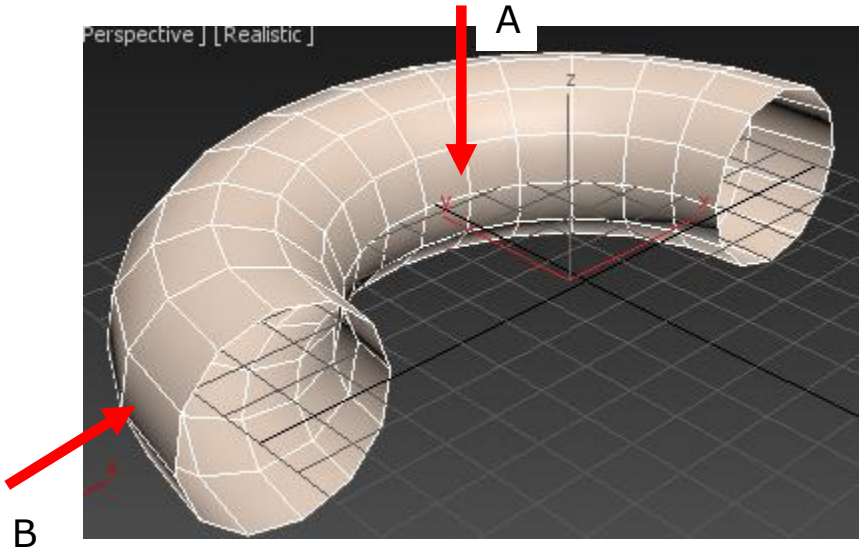


- Why Dog's Ear is drawn?

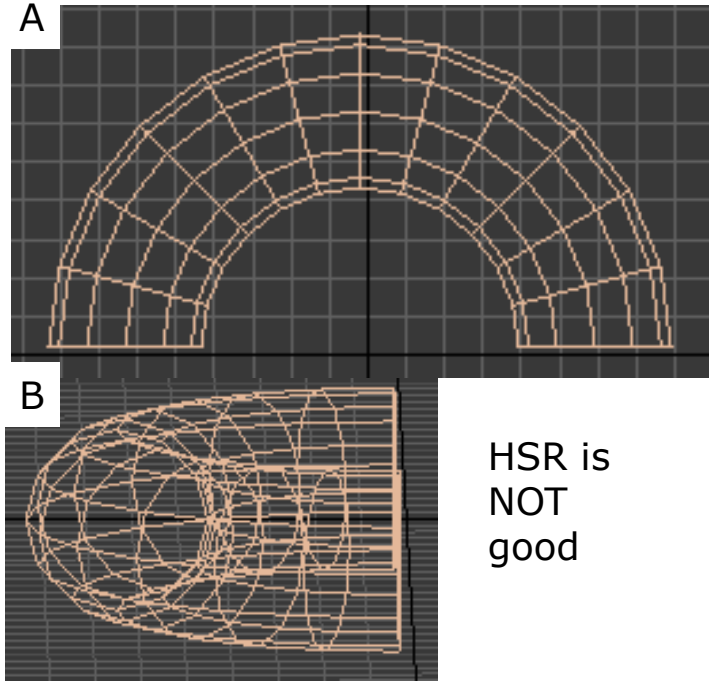


Final Problems of Graphics is Depth Problem

- Hidden Surface Removal(HSR) is NOT the sufficient condition for 3D Graphics



Torus

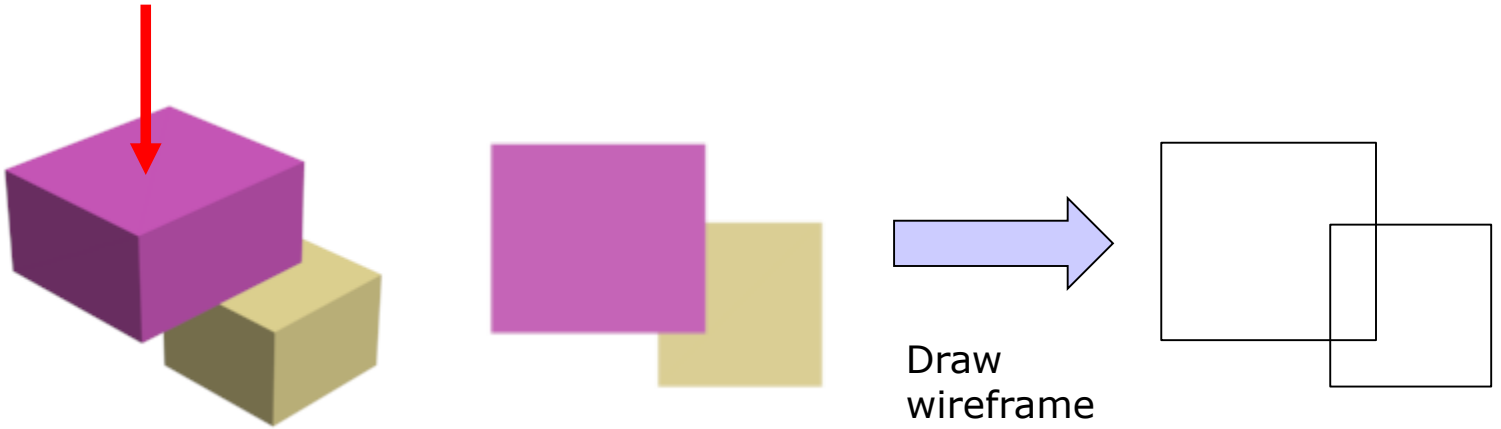


HSR is good

HSR is NOT good

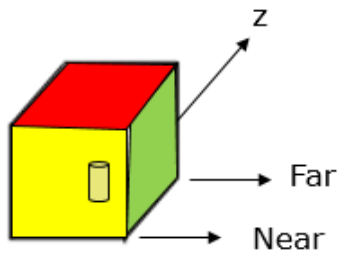
- HSR is that opposite polygons are Hidden.
  - It does NOT determine if far polygon is overlapped by closer polygon

Z direction Depth Problem



- HSR has no function of Which One is close or far

- How Graphics solve this problem?
  - Z buffering from Unit Space Mapping in Ch. 3
  - But, we do NOT implement it.
  - OpenGL or Direct X has Z-buffering, too.



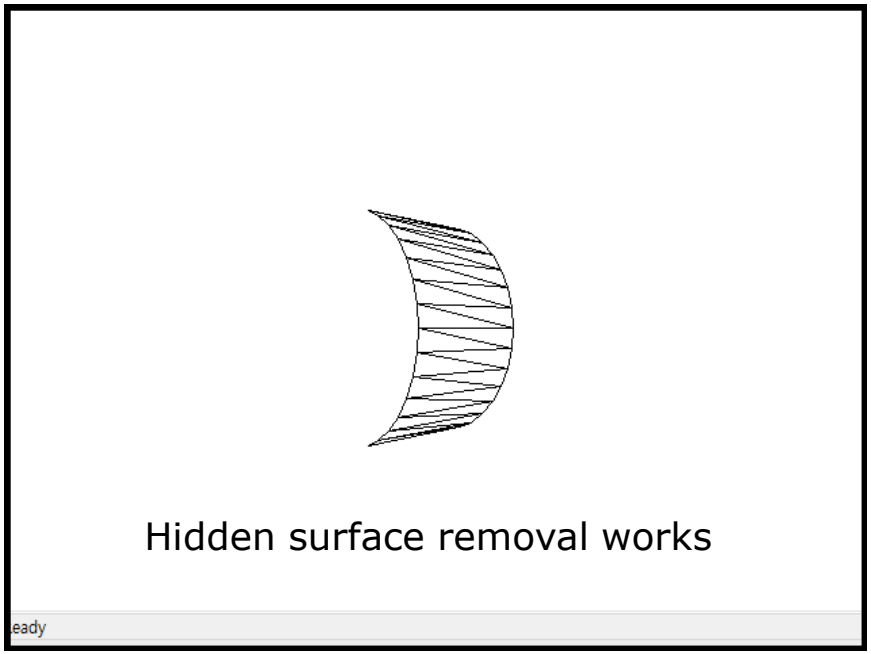
W=H=T=[-1,1]

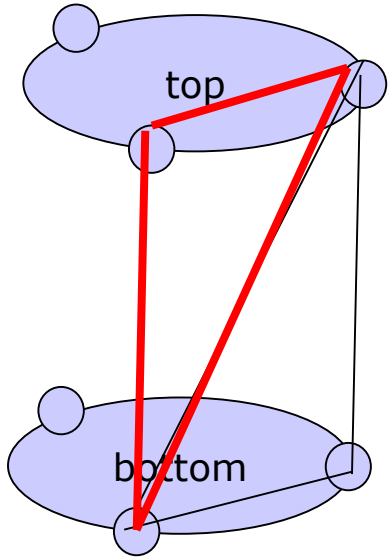
Unit Space

4 Example of Object Primitives

Make Cylinder (Radius, height, Resolution)

- uWnd-25-Cylinder before uPolygon Class
- uWnd-30-cylinder after uPolygon Class





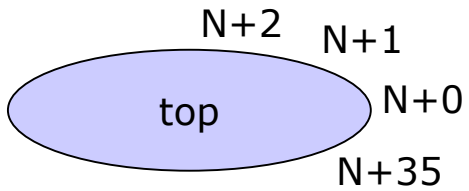
If resolution=3, angle gap is 360/3 = 120.

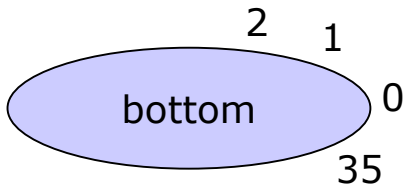
Example) uWnd-30-cylinder

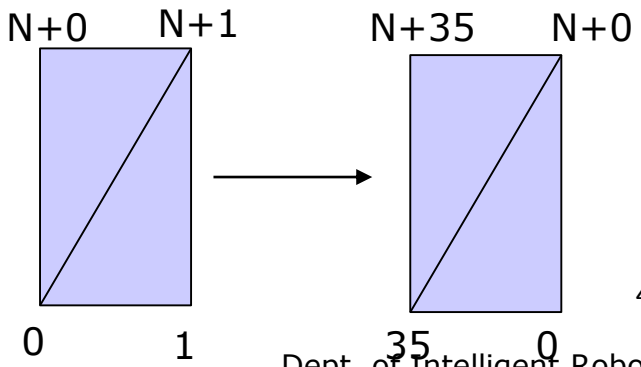
```
void uObj::MakeCyl(float r,float h,int n)
{
    Alloc(2*n, 2*n*2);

    float x,y;
    float dq = 360/((float)n);

    int i;
    // vertices
    for (i=0;i<n;i++)
    {
        x   = r*cos(RAD(i*dq));
        y   = r*sin(RAD(i*dq));

        pVer[i]     = uVector(x,y,0);
        pVer[i+n]   = uVector(x,y,h);
    }

    //Polygons
    int j=0;
    for (i=0;i<n;i++)
    {
        int next = i+1;
        if (next>=n)     next = 0;

        pPoly[j].Set(i,next,next+n);
        j++;
        pPoly[j].Set(i,next+n, i+n);
        j++;
    }
}
```

N(resolution)=36

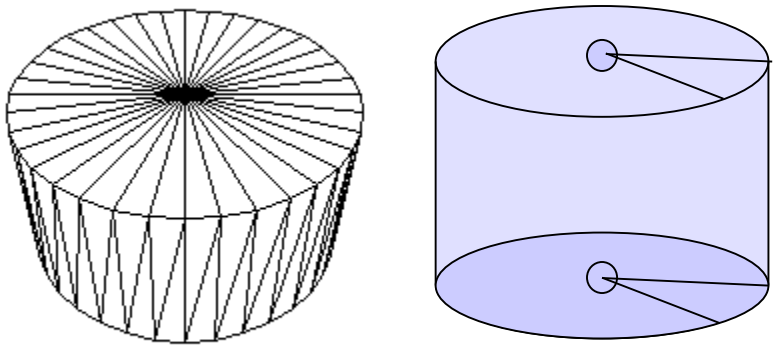
Vertex = N*2
Polygon= N*2*2 ← Two triangles







Top and Bottom of Cylinder



- Vertex = 2* N+ 2 (top+bottom)
- Polygon = 2N(side) + N(top) + N(bottom)

- Modify MakeCyl function for top and bottom.