Computer Graphics and Programming

Lecture 8
Basics of OpenGL

Jeong-Yean Yang

2020/12/8

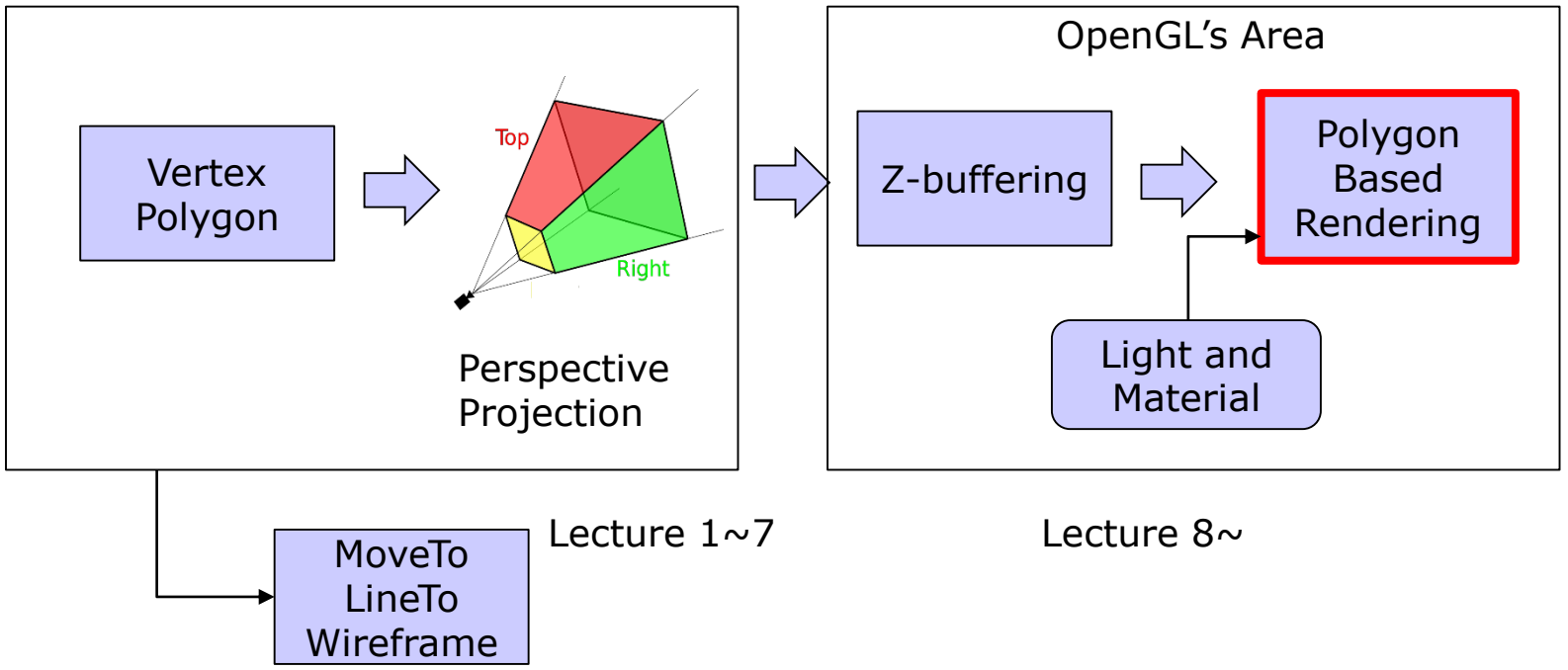
1 Introduction to OpenGL

Rendering API

- OpenGL
  - Open Graphics Library proposed by Silicon Graphics Inc. in 1992
  - Silicon Graphics Inc. produces Computer Graphics Workstation(The father of computer graphics)
  - OpenGL is the Strong Standard Platform for NVidia or AMD
  - OpenGL works on Windows, Linux, Android, iOS, and even any kinds of embedded machine.

- DirectX
  - Microsoft challenges to OpenGL World.
  - It works on Windows compatible devices, such as PC, Xbox, Sega, and Dreamcast

What is OpenGL's purpose?



- OpenGL fill polygons with colors by Lights and materials.
- OpenGL uses Hardware acceleration → Fast speed.

History of OpenGL

- 1.0~ 1.5 : glPushMatrix, glPopMatrix
  - Push and Pop structures are used
  - OpenGL ES ver. 1.x

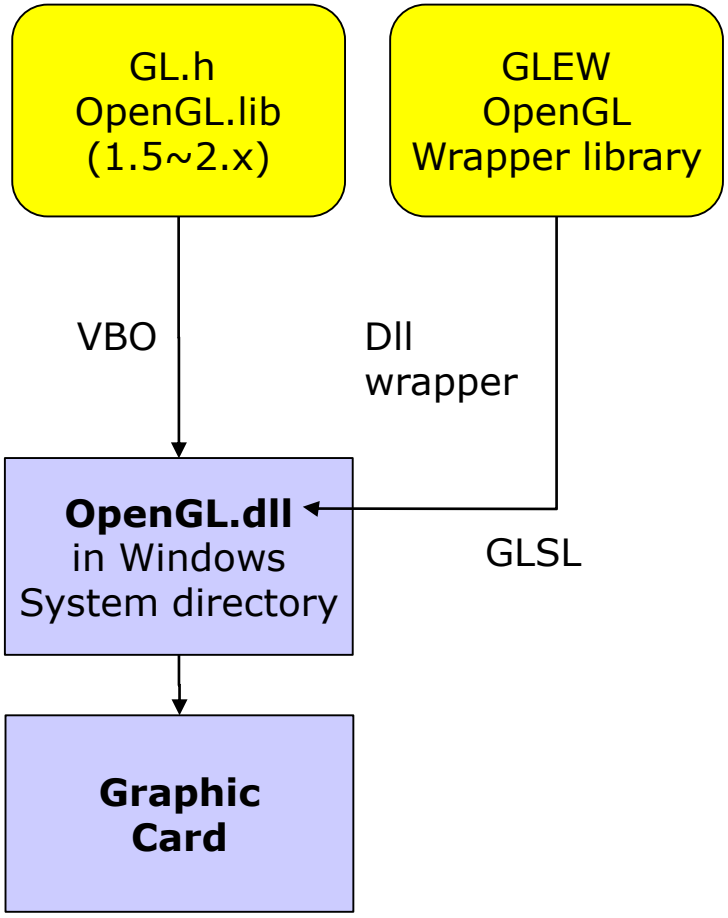
```
glPushMatrix();
    glRotatef(90,
    glEnable(GL_T
    glBindTexture
    gluQuadricTex
    gluQuadricDra
    gluDisk(obj,
    gluCylinder(o
    glDisable(GL_
glPopMatrix();
```

- 2.1: Vertex Buffer Object (VBO)
  - glGen**Buffer**, glBind**Buffer**
  - OpenGL ES ver. 2.x
  - Shading language(GLSL) appears.

Still VBO is more popular than VAO

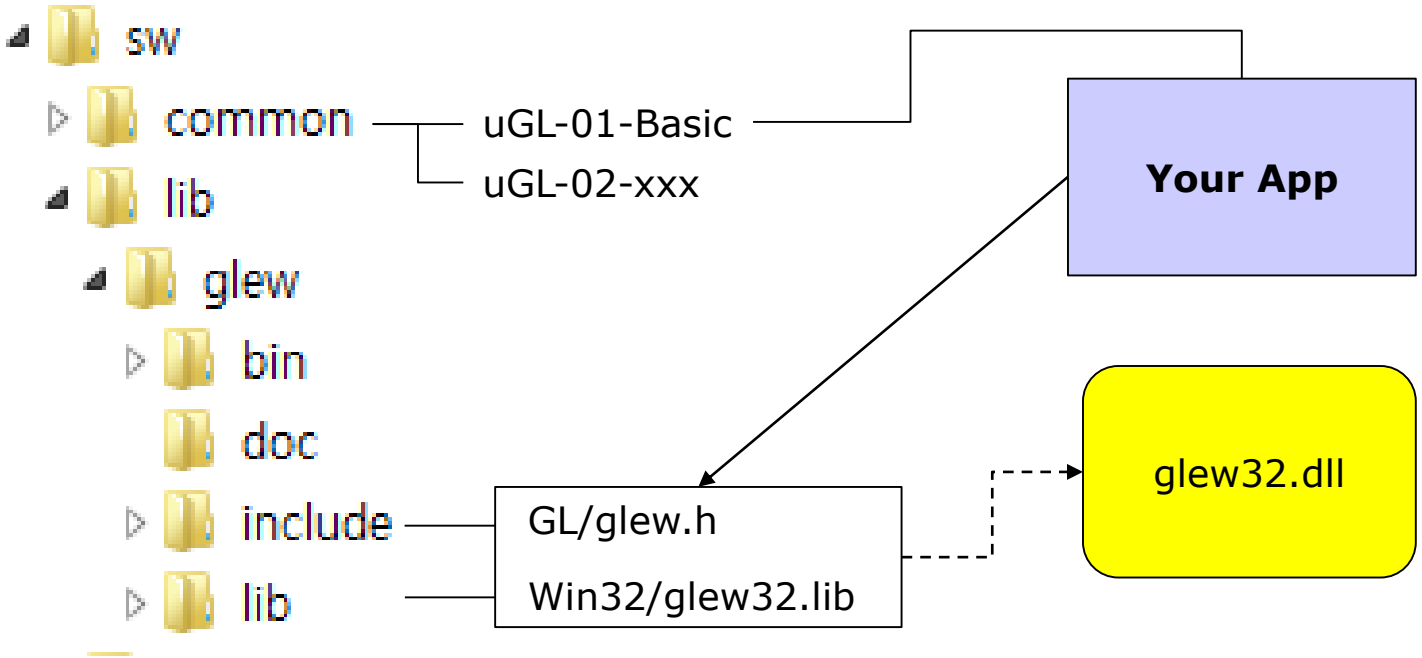
- 3.0: Vertex Array Object (VAO)
  - glBindVertex**Array**
  - OpenGL ES ver. 2.x

OpenGL Architecture in this Class



- GL.h + GLEW for our example
- GLSL works on Only GLEW library.

Basic Directory for OpenGL usages



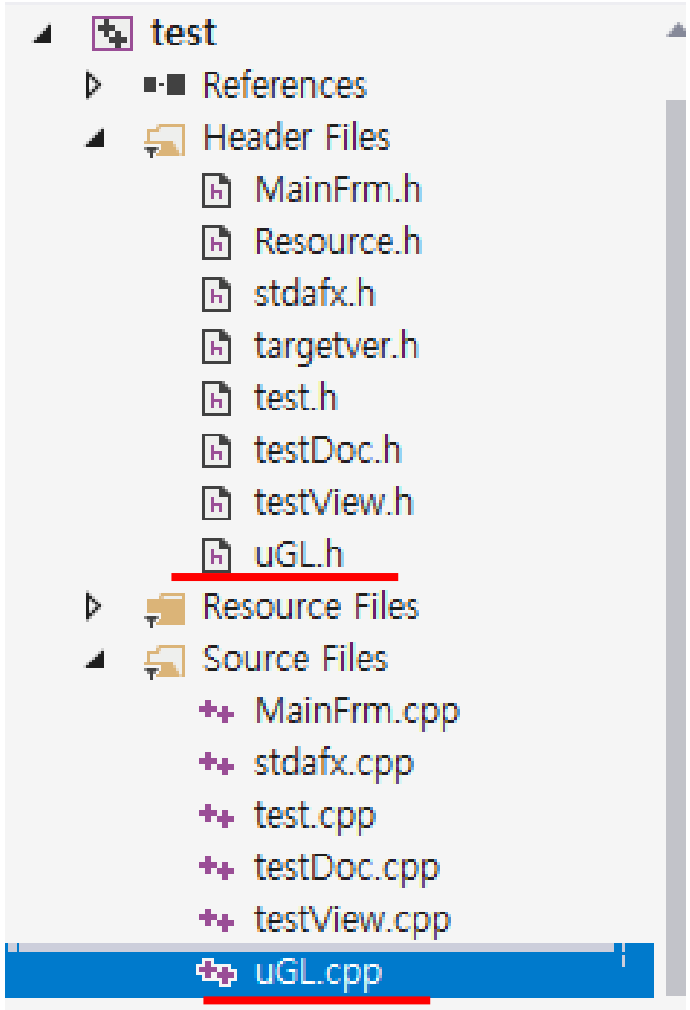
- Your App(uGL-01-Basic) uses “lib/glew/include”
- Glew.h and glew32.lib provide interfaces for glew32.dll

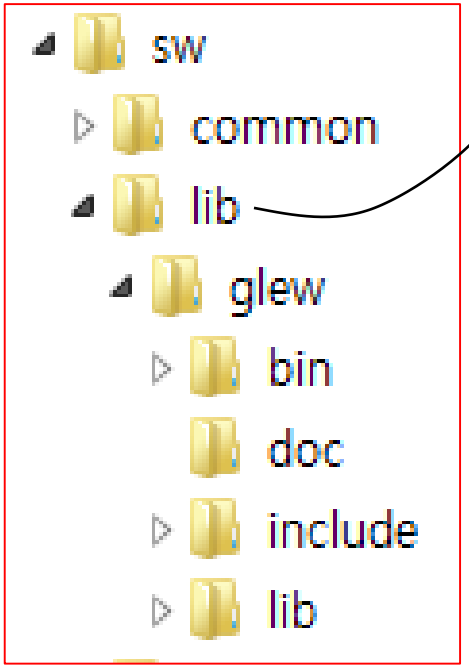
2

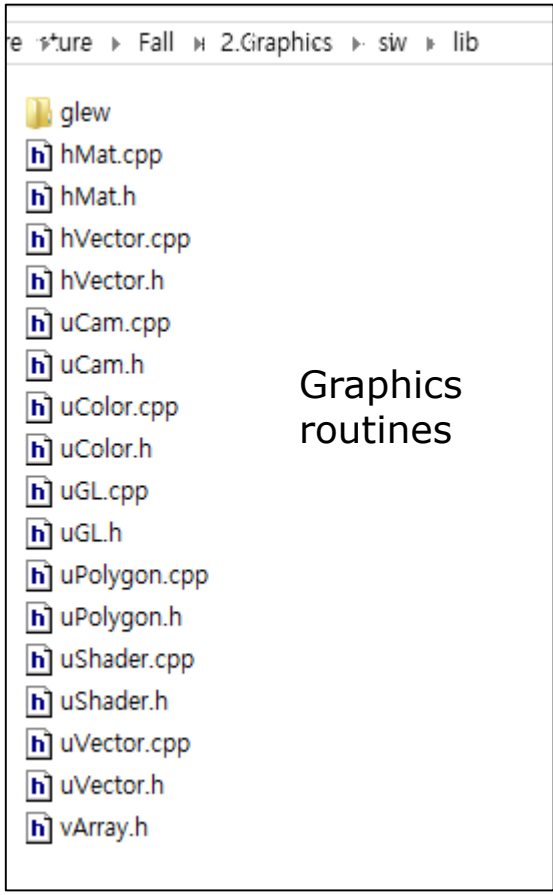
OpenGL in Windows Environment

Something Hard

uGL-01-Basic
New class, “uGL” for OpenGL





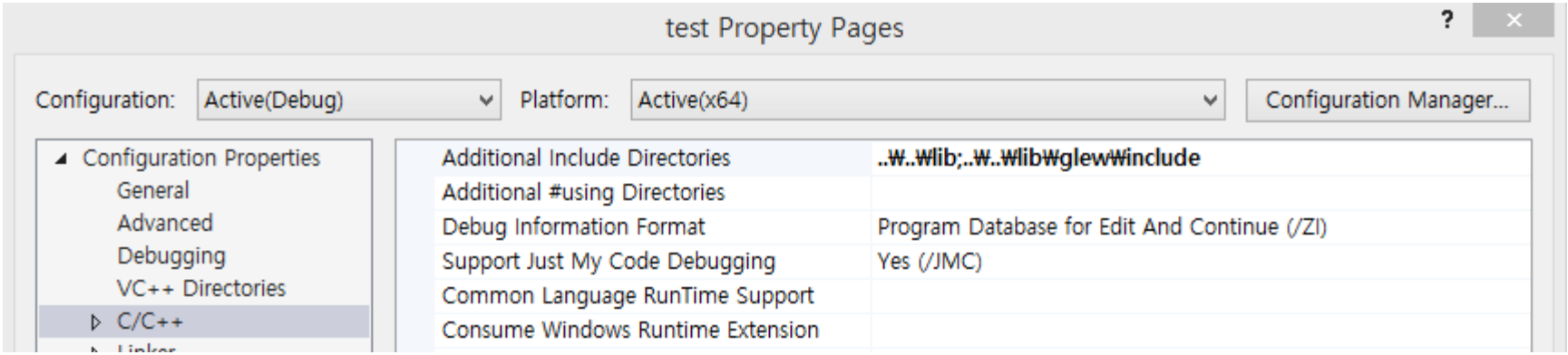


Graphics routines

Your Program uses
1. glew/include
2. Glew/lib

→ Project settings are changed.

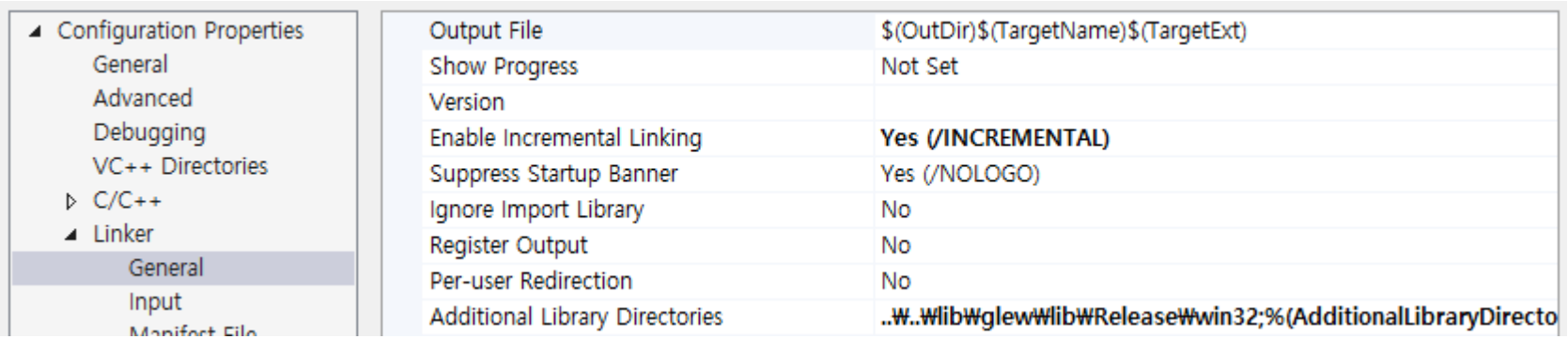
Use files under “lib”
Set Include directory



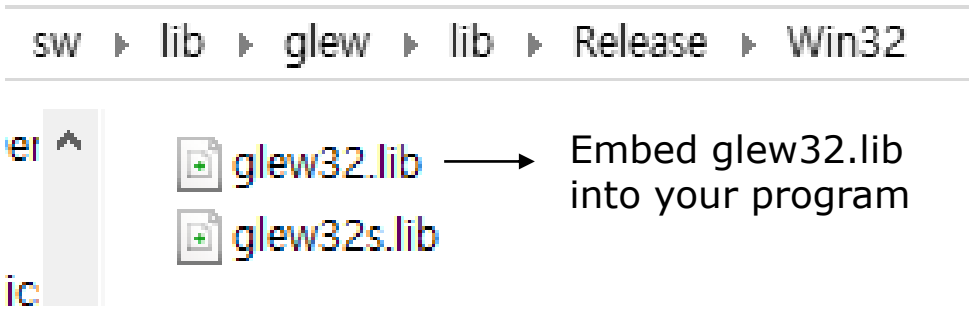
- Additional include directories
  - ..\..\lib; ..\..\lib\glew\include

- .\ = “sw\common\uGL-01-Basic”
- ..\ = “sw\common”
- ..\..\lib=“sw\lib”

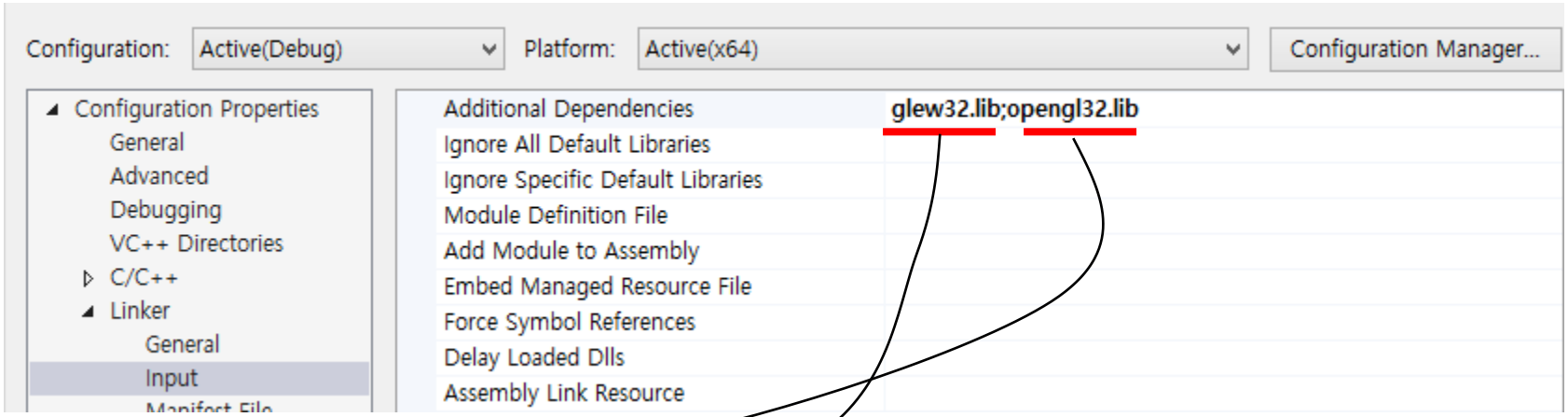
Use files under “lib/glew/lib”
Set Library Directory

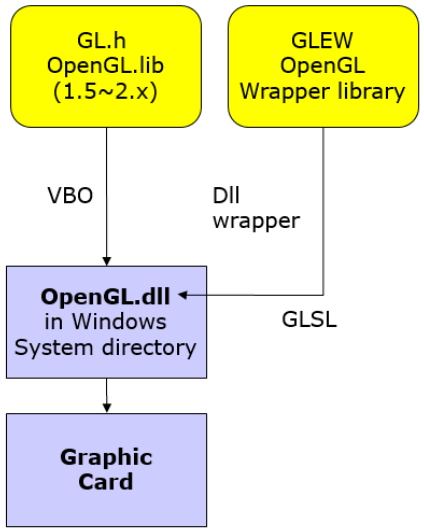


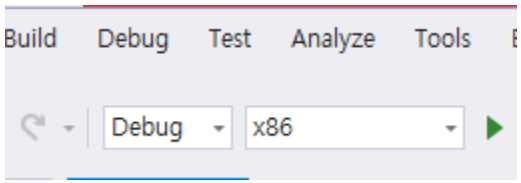
- ..\..\lib\glew\lib\release\win32



Link glew32.lib and OpenGL32.lib into your App







Warning!!
ONLY x86 (32bit)

uGL.h

```
#include <GL/glew.h>
#include <GL/GL.h>
#include <GL/wglew.h>
```

Header file

```
class uGL : public CWnd
{
public:
    uGL();
    virtual ~uGL();
public:
    virtual BOOL Create(CRect rect,CWnd *pParentWnd);
    void    PreInit();
    void    PostInit();
    virtual void Draw();
    virtual void Loading();
    virtual void Close();
    void    SetBK(COLORREF);

protected:
    HDC     m_hDC;
    HGLRC   m_hRC;

    struct _info
    {
        float       r,g,b,a;
        BOOL        bLoaded;
    } info;

protected:
    DECLARE_MESSAGE_MAP()
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnTimer(UINT_PTR);
    afx_msg void OnDestroy();
```

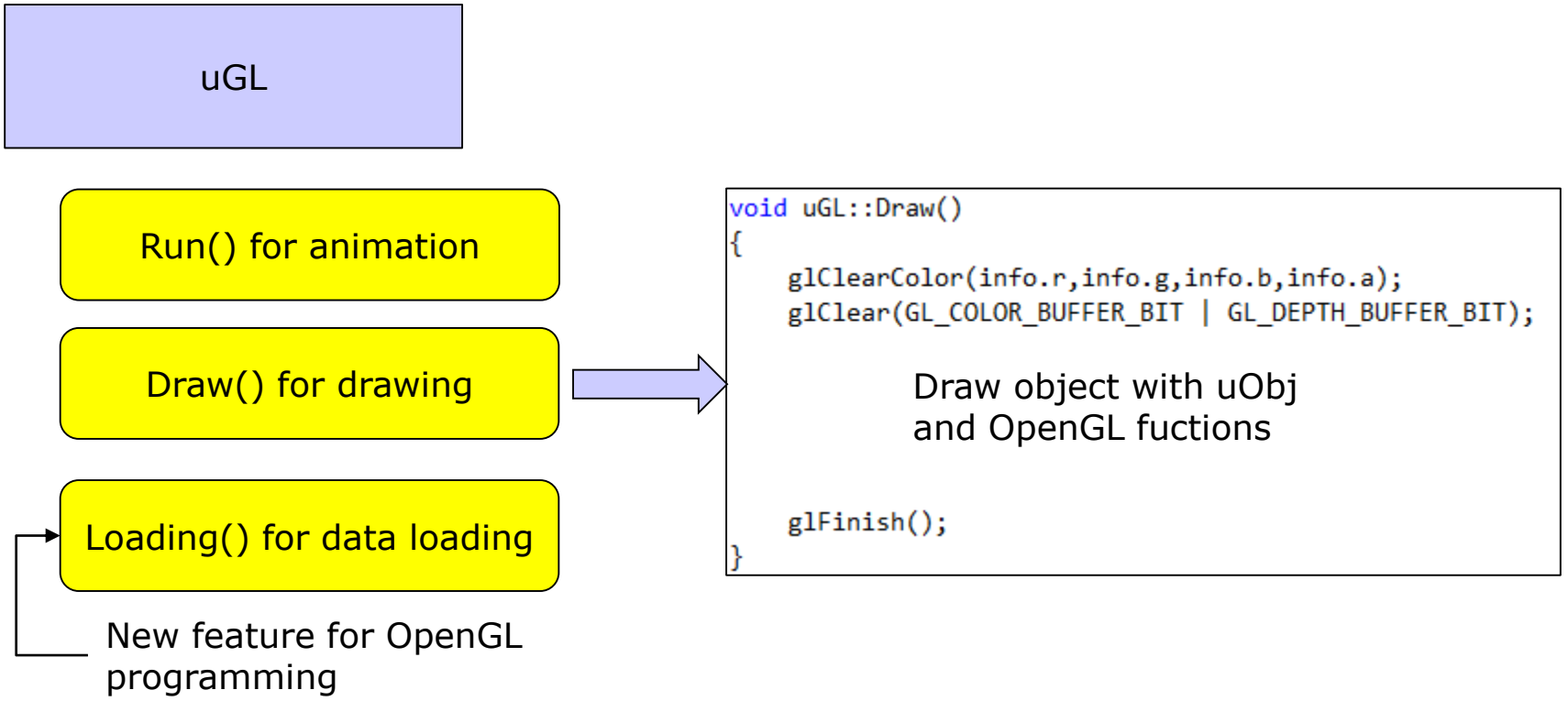
Device context for OpenGL

uGL.cpp

```
int uGL::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if(CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    SetTimer(1,10,NULL);

    m_hDC    = ::GetDC(m_hWnd);

    static PIXELFORMATDESCRIPTOR pfd ={
        sizeof(PIXELFORMATDESCRIPTOR),   // Size of this structure
        1,                               // Version of this structure
        PFD_DRAW_TO_WINDOW |             // Draw to Window (not to bitmap)
        PFD_SUPPORT_OPENGL |             // Support OpenGL calls in window
        PFD_DOUBLEBUFFER,                // Double buffered mode
        PFD_TYPE_RGBA,                   // RGBA Color mode
        32,                              // Want 24bit color
        8,16,8,8,8,0,                    // Not used to select mode
        0,0,                             // Not used to select mode
        64,16,16,16,0,                       // Not used to select mode
        32,                              // Size of depth buffer
        8,                               // Not used to select mode
        0,                               // Not used to select mode
        PFD_MAIN_PLANE,                  // Draw in main plane
        0,                               // Not used to select mode
        0,0,0};                        // Not used to select mode

    int nPixelFormat = ChoosePixelFormat(m_hDC,&pfd);
    VERIFY(SetPixelFormat(m_hDC,nPixelFormat,&pfd));

    m_hRC = wglCreateContext(m_hDC);
    PreInit();

    glewInit();
```

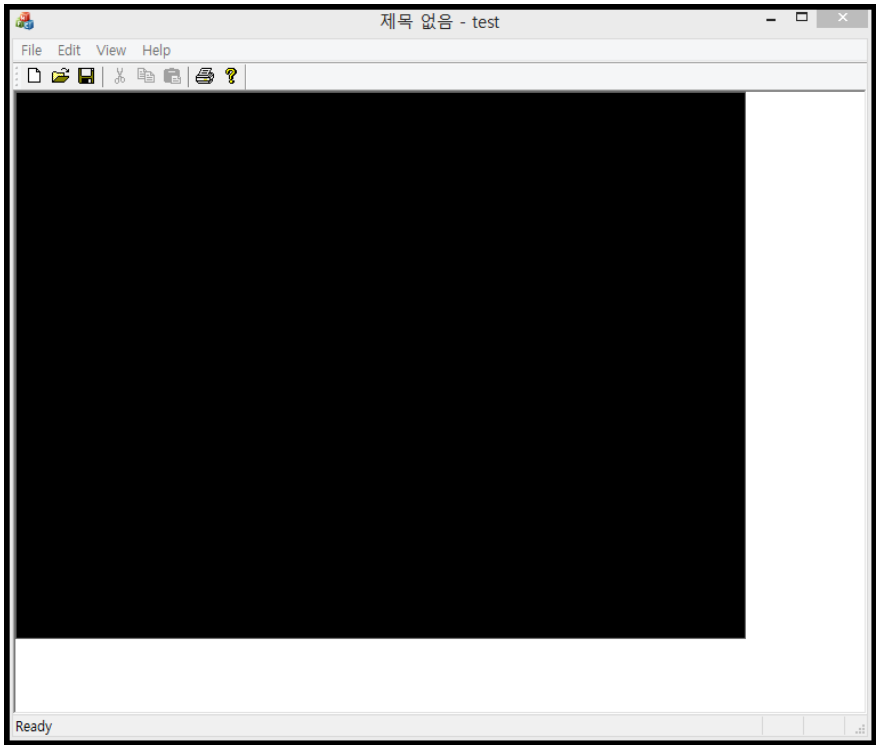

```
    int attribs[] =
    {
        WGL_CONTEXT_MAJOR_VERSION_ARB, 3,
        WGL_CONTEXT_MINOR_VERSION_ARB, 1,
        WGL_CONTEXT_FLAGS_ARB, 0,
        0
    };

    if(wglewIsSupported("WGL_ARB_create_context") == 1)
    {
        wglDeleteContext(m_hRC);
        m_hRC = wglCreateContextAttribsARB(m_hDC,0,attribs);
        wglMakeCurrent(NULL,NULL);
        wglMakeCurrent(m_hDC,m_hRC);
    }

    int nVersion[2];
    glGetIntegerv(GL_MAJOR_VERSION,&nVersion[0]);
    glGetIntegerv(GL_MINOR_VERSION,&nVersion[1]);

    if(nVersion[0]<3)    AfxMessageBox(L"At least OpenGL 3.1 is required.");
    {
        glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
        glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);
        glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
        glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);

        glDepthFunc(GL_LEQUAL);
        glEnable(GL_TEXTURE_2D);
        glEnable(GL_DEPTH_TEST);

        glAlphaFunc(GL_GREATER,0.01);
        glBlendFunc(GL_SRC_ALPHA,GL_ONE);
    }

     PostInit();
     return 0;
}
```

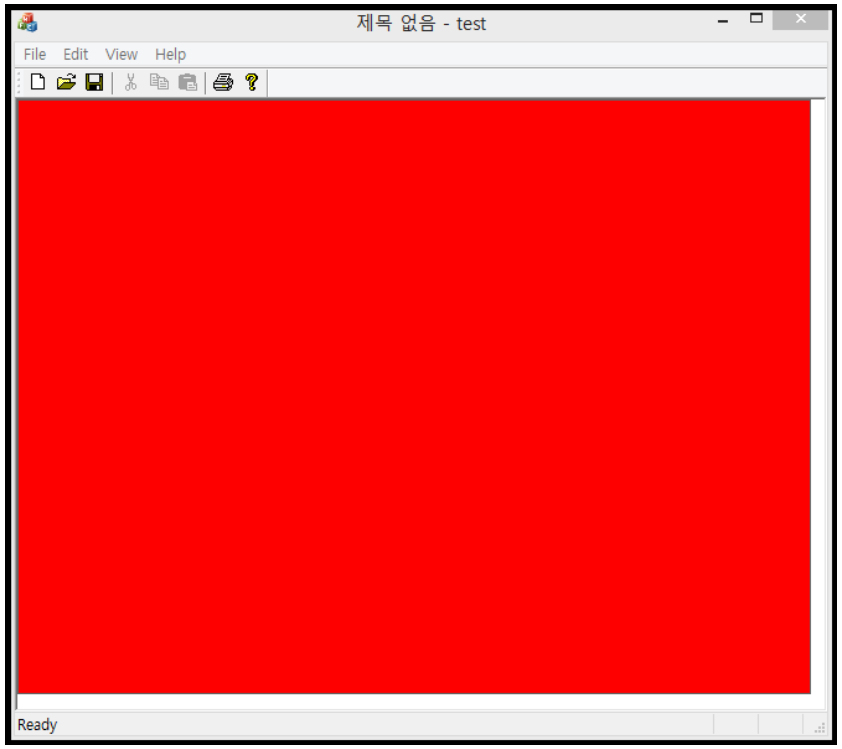

uGL structure



- glClearColor → fills background color (Erase all)
- glFinish → Rendering finishes here.

Ex) uGL-01-Basic





SetBK(RGB(0,0,0));

→

`glClearColor(info.r,info.g,info.b,info.a);`

SetBK(RGB(255,0,0));

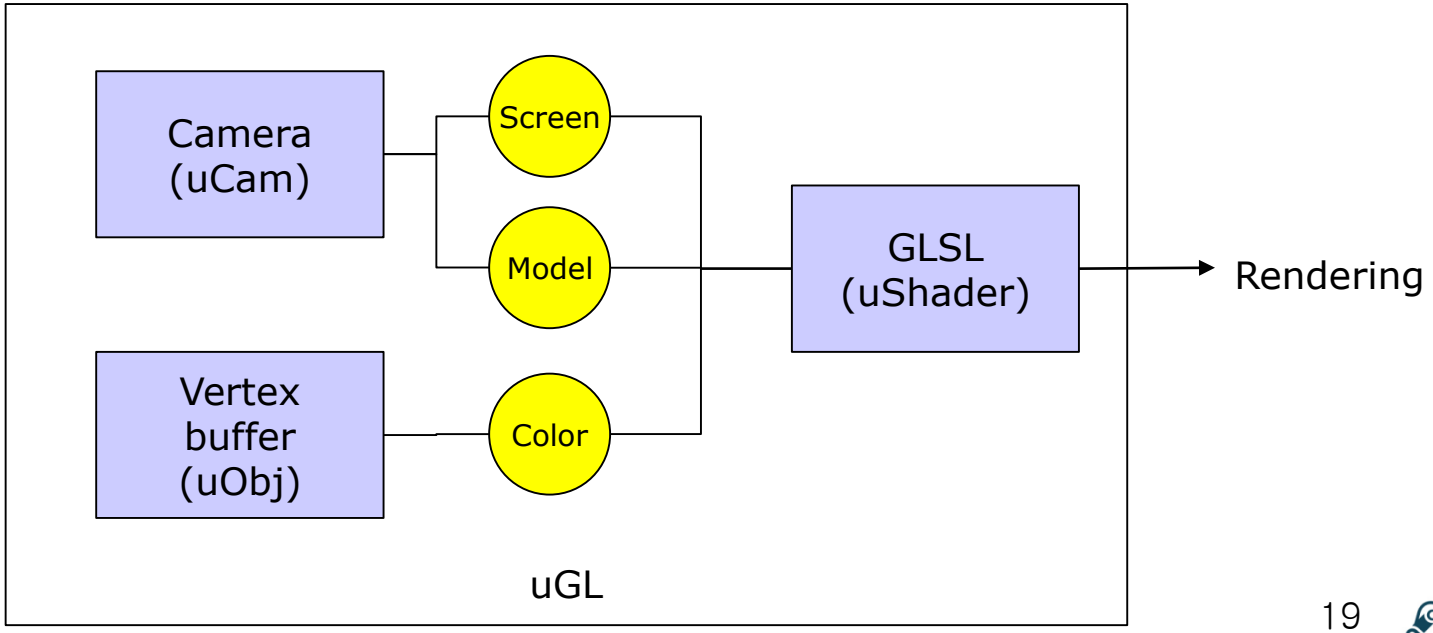
→

`glClearColor(info.r,info.g,info.b,info.a);`

3 Draw Polygon by Vertex Buffer Object (VBO)

Basic Structure in this Class

- uGL : OpenGL environment
- uObj: VBO-based Object Modeling
- uShader: Shader(GLSL)-based rendering
- uCam: perspective mapping(screen) and viewpoint transform (model)



Data Loading Must be in OpenGL Thread

- OpenGL has its own Thread.
- uGL::Draw() and uGL::Loading() work in OpenGL thread

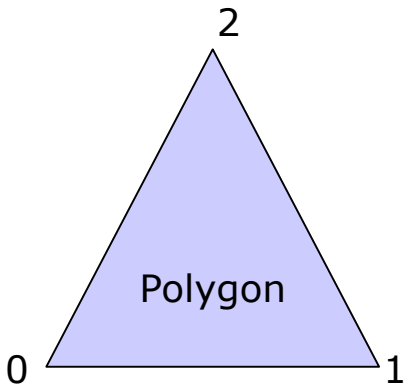
- **Keep it in your mind:**
- Loading out of OpenGL thread is NOT applied

- Any glxxxxx function, Shader, and VBO must be used in Loading() and Draw() functions(exactly in OpenGL thread)

Draw Polygon in OpenGL

Vertex has Position and Normal Vector

Case 1) MFC part



uVector pVer[3]

uPolygon poly.set(0,1,2)

Case 2) VBO part

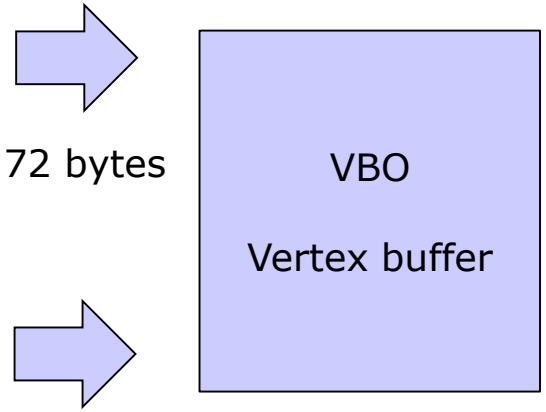
```
GLuint vs,fs;
float pVer[]={ 0,0,0,    // v0
               0,0,1,    // normal at v0
               1,0,0,    // v1
               0,0,1,    // normal at v1
               0,1,0,    // v2
               0,0,1};   // normal at v2

unsigned short face[3]  = { 0,1,2};
```

- A vertex
  - has the position, x, y, and z
  - Has the normal vector, nx, ny, nz
- vs is the handle for vertex buffer
- fs is the handle for polygon(face or element) buffer.

Copy Vertex Memory in CPU into Vertex Buffer (VBO) in GPU

```
GLuint vs,fs;
float pVer[]={ 0,0,0,    // v0
               0,0,1,    // normal at v0
               1,0,0,    // v1
               0,0,1,    // normal at v1
               0,1,0,    // v2
               0,0,1};   // normal at v2

unsigned short face[3]  = { 0,1,2};
```

Memory in CPU



- Float = 4byte, unsigned short = 2byte
- Each point in the polygon has 24 bytes
  - 3 float position variables, x, y, and z (12 bytes)
  - 3 float normal vector variables, nx, ny, and nz (12bytes)
- Polygon has 3 Points (24x3 =72bytes)
- Polygon Index has 3 indices (0,1,2) = (2x3 = 6bytes

1. Generate Vertex Buffer in “Loading” function

```
GLuint vs,fs;
float pVer[]={ 0,0,0,    // v0
               0,0,1,    // normal at v0
               1,0,0,    // v1
               0,0,1,    // normal at v1
               0,1,0,    // v2
               0,0,1};   // normal at v2

unsigned short face[3]  = { 0,1,2};
```

72 bytes

glGenBuffers(1,&vs);

6 bytes

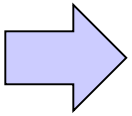
glGenBuffers(1,&fs);

- glGenBuffers creates storage(or array) in the GPU.
- 72 bytes for vertices and 6 bytes for index are not allocated yet.

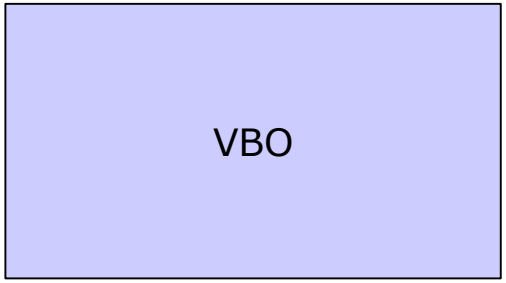
2. Copy Vertex into GPU

2.1 Open VBO memory in GPU

```
// global variable
GLuint vs,fs;
float pVer[]={ 0,0,0,    // x
               0,0,1,    // normal at x
               1,0,0,    // y
               0,0,1,    // normal at y
               0,1,0,    // z
               0,0,1};   // normal at z

unsigned short face[3]  = { 0,1,2};
```



72 bytes

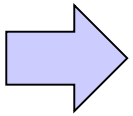



```
glBindBuffer(GL_ARRAY_BUFFER, vs);
```
Open VBO

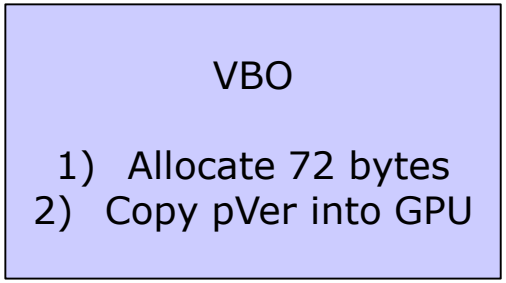
```
glBindBuffer(GL_ARRAY_BUFFER, 0);
```
Close VBO

2.2 Allocate and Copy 72bytes

```
// global variable
GLuint vs,fs;
float pVer[]={ 0,0,0,    // x
               0,0,1,    // normal at x
               1,0,0,    // y
               0,0,1,    // normal at y
               0,1,0,    // z
               0,0,1};   // normal at z

unsigned short face[3]  = { 0,1,2};
```



72 bytes




```
glBindBuffer(GL_ARRAY_BUFFER, vs);
```
Open VBO

```
glBufferData(GL_ARRAY_BUFFER, 72, pVer, GL_STATIC_DRAW);
```

```
glBindBuffer(GL_ARRAY_BUFFER, 0);
```
Close VBO

2. Copy Vertex into GPU

2.3 VBO has two fields (position and normal)

```
// global variable
GLuint vs,fs;
float pVer[]={ 0,0,0,    // x
               0,0,1,    // normal at x
               1,0,0,    // y
               0,0,1,    // normal at y
               0,1,0,    // z
               0,0,1};   // normal at z

unsigned short face[3]  = { 0,1,2};
```

12 bytes position

12 bytes normal

```
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 12,
(void*)0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 12,
(void*)12);

glDisableVertexAttribArray(0);
glDisableVertexAttribArray(1);
```

Float x,y,z = 3 EA

```
0: glVertexAttribPointer(0, 3, GL_FLOAT,
                GL_FALSE, 12, (void*)0);
```

Float x,y,z = 12bytes

Position has 12 bytes

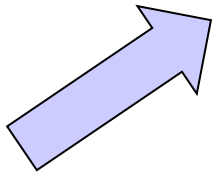
```
1: glVertexAttribPointer(1, 3, GL_FLOAT,
                GL_FALSE, 12, (void*)12);
```

Float nx,ny,nz

p	o	s	i	t	i	o	n	(x)	
		n	o	r	m	a	l	(n	x)
				p	o	s	i	t	o
n	(y)					n	o	r	m
a	l	(n	y)					p	o
s	i	t	i	o	(z)				
n	o	r	m	a	l	(n	z)		

3. Copy Face Index into GPU

```
// global variable
GLuint vs,fs;
float pVer[]={ 0,0,0,    // x
               0,0,1,    // normal at x
               1,0,0,    // y
               0,0,1,    // normal at y
               0,1,0,    // z
               0,0,1};   // normal at z

unsigned short face[3]  = { 0,1,2};
```



6 bytes

Open VBO

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);

glBufferData(GL_ELEMENT_ARRAY_BUFFER, 6, face,
GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

Close VBO

- glBufferData has two types
  - Vertex buffer (72 bytes)

```
glBufferData(GL_ARRAY_BUFFER, 72, pVer, GL_STATIC_DRAW);
```

  - Polygon index buffer (6 bytes)

```
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 6, face, GL_STATIC_DRAW);
```

- Now, loading is finished.

4. Draw Polygon
Remind: All data are uploaded into GPU

```
void uGL::Draw()
{
    glClearColor(info.r,info.g,info.b,info.a);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // call vertex buffer
    glBindBuffer(GL_ARRAY_BUFFER, vs);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0,3, GL_FLOAT,FALSE,sizeof(float)*6,(void*)0);     // vertex,v
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1,3, GL_FLOAT,FALSE,sizeof(float)*6,(void*)12);    // normal,n

    // call face buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);

    // draw
    {
        glUniform4f(m_sh.diffuse    ,1,0,0,1);
        glDrawElements(GL_TRIANGLES, 3*1, GL_UNSIGNED_SHORT, (void*)0);
    }

    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER,0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

    glFinish();
}
```

Open Vertex buffer

Open index buffer

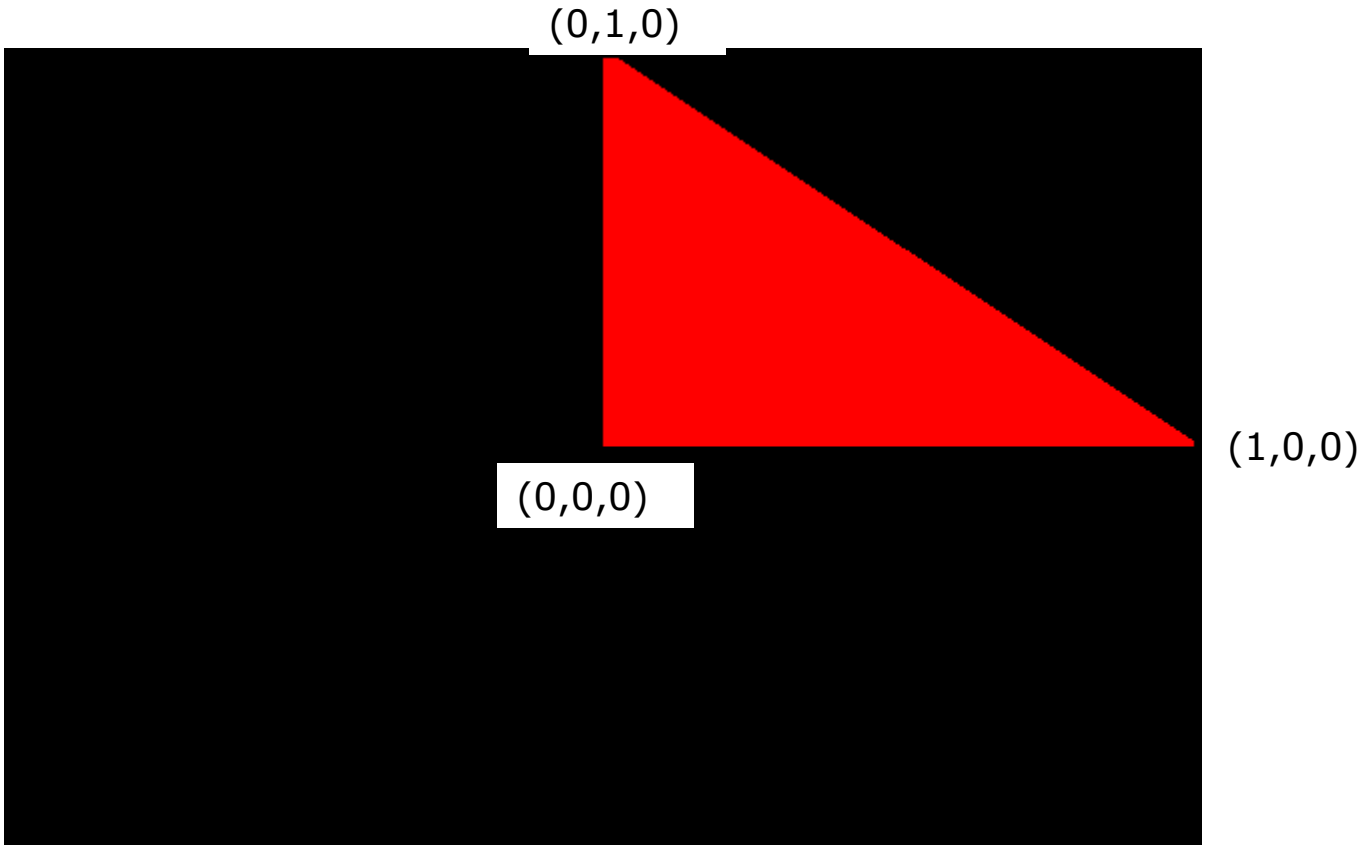
glUnivorm4f(ooo, r, g, b, a)

Draw Polygon 3 points by referring index buffer

Close vertex buffer

Close index buffer

Ex) uGL-02-Polygon-Color
Draw a Triangle Polygon

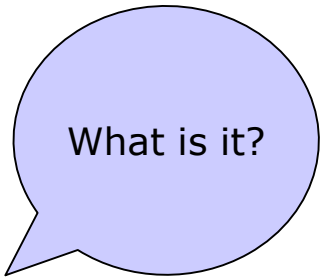


- Why Width is NOT same with Height?
  - It is NOT in space, [-320,320], but in space, [-1 1]

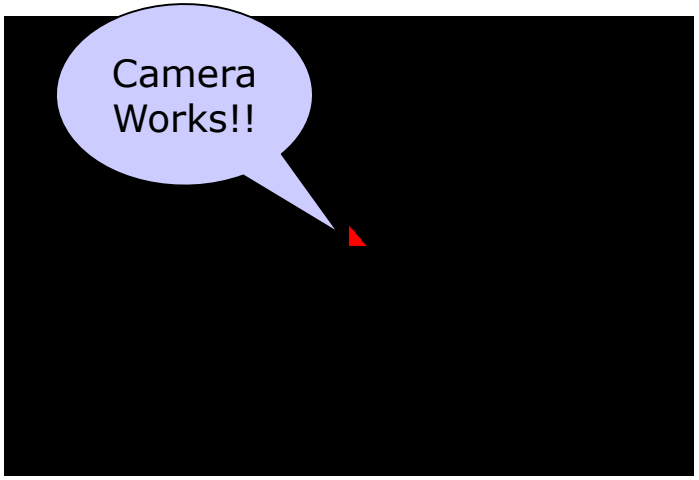
ex) uGL-03-Object-Camera

- The Unit Space, [-1, 1] is scaled by Projection.

```
void uGL::Draw()
{
    glClearColor(info.r,info.g,info.b,info.a);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glFrontFace(GL_CCW);

    // draw
    {
        hMat h  = cam.T*cam.R;
        glUniformMatrix4fv(pCurrentShader->screen, 1, 0, cam.P.v);
        glUniformMatrix4fv(pCurrentShader->model, 1, 0, h.v);

        // call vertex buffer
        glBindBuffer(GL_ARRAY_BUFFER, vs);
        glEnableVertexAttribArray(0);
        glVertexAttribPointer(0,3, GL_FLOAT,FALSE,sizeof(float)*6,(void*)0);
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(1,3, GL_FLOAT,FALSE,sizeof(float)*6,(void*)12);

        // call face buffer
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);

        glUniform4f(m_sh.diffuse    ,1,0,0,1);
        glDrawElements(GL_TRIANGLES, 3*1, GL_UNSIGNED_SHORT, (void*)0);

        glDisableVertexAttribArray(0);
        glDisableVertexAttribArray(1);
        glBindBuffer(GL_ARRAY_BUFFER,0);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
    }
    glFinish();
}
```



Ex) uGL-03-Object-Camera Result




```
void uGL::Loading()
{
    // load shader
    m_sh.Load("solid.vsh","solid.fsh");
    pCurrentShader = &m_sh;
```

Load shader

- Screen (perpective Mapping)
  - glUniformMatrix4fv(pCurrentShader->screen, 1, 0, cam.P.v);
- Model (Camera Transform)
  - hMat h = cam.T*cam.R;
  - glUniformMatrix4fv(pCurrentShader->model, 1, 0, h.v);

```
void glUniformMatrix4fv( GLint location,
                         GLsizei count,
                         GLboolean transpose,
                         const GLfloat *value);
```

- What is model and screen from pCurrentShader(uShader)?
  - We will learn GLSL in later parts

4 Objects in OpenGL

Build Objects in Class, uObj

- 1. MakeBox, MakeCyl as in the previous examples
- **2. Update() is modified for copying data into GPU**

```
void uObj::Update()
{
    // VBO
    glGenBuffers(1,&vs);
    glGenBuffers(1,&fs);

    // vertex
    glBindBuffer(GL_ARRAY_BUFFER, vs);
    glBufferData(GL_ARRAY_BUFFER, sizeof(uVertex)* nVer, pVer, GL_STATIC_DRAW);
        glEnableVertexAttribArray(0);
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(uVertex), (void*)0);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(uVertex), (void*)12);
        glDisableVertexAttribArray(0);
        glDisableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER,0);

    // face
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(uPolygon)*nPoly, pPoly, GL_STATIC_DRAW);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
}
```

Drawing in uGL

```
void uGL::Draw()
{
    glClearColor(info.r,info.g,info.b,info.a);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // set shader
    hMat h  = cam.T*cam.R;
    glUniformMatrix4fv(pCurrentShader->screen, 1, 0, cam.P.v);
    glUniformMatrix4fv(pCurrentShader->model, 1, 0, h.v);

    {
        box.Draw();
    }

    glFinish();
}
```

Set camera variable on shader

Draw object

Drawing in uObj (Set object's color)

```
void uObj::Draw()
{
    glUniform4f(pCurrentShader->diffuse, diffuse.r,diffuse.g,diffuse.b,diffuse.a);
    glUniform4f(pCurrentShader->ambient, ambient.r,ambient.g,ambient.b,ambient.a);
    glUniform4f(pCurrentShader->specular, specular.r,specular.g,specular.b,specular.a);

    // call vertex buffer
    glBindBuffer(GL_ARRAY_BUFFER, vs);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)0);     // vertex,v
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)12);    // normal,n

    // call face buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);

    glDrawElements(GL_TRIANGLES, 3*nPoly, GL_UNSIGNED_SHORT, (void*)0);

    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER,0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

}
```

Set three colors on shader

Drawing in uObj (call vertex VBO)

```
void uObj::Draw()
{
    glUniform4f(pCurrentShader->diffuse, diffuse.r,diffuse.g,diffuse.b,diffuse.a);
    glUniform4f(pCurrentShader->ambient, ambient.r,ambient.g,ambient.b,ambient.a);
    glUniform4f(pCurrentShader->specular, specular.r,specular.g,specular.b,specular.a);

    // call vertex buffer
    glBindBuffer(GL_ARRAY_BUFFER, vs);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)0);     // vertex,v
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)12);    // normal,n

    // call face buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);

    glDrawElements(GL_TRIANGLES, 3*nPoly, GL_UNSIGNED_SHORT, (void*)0);

    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER,0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

}
```

Open Vertex (Position and Norma) VBO

Drawing in uObj (call Polygon VBO)

```
void uObj::Draw()
{
    glUniform4f(pCurrentShader->diffuse, diffuse.r,diffuse.g,diffuse.b,diffuse.a);
    glUniform4f(pCurrentShader->ambient, ambient.r,ambient.g,ambient.b,ambient.a);
    glUniform4f(pCurrentShader->specular, specular.r,specular.g,specular.b,specular.a);

    // call vertex buffer
    glBindBuffer(GL_ARRAY_BUFFER, vs);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)0);     // vertex,v
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)12);    // normal,n

    // call face buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);

    glDrawElements(GL_TRIANGLES, 3*nPoly, GL_UNSIGNED_SHORT, (void*)0);

    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER,0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

}
```

Call Face(polygon) index VBO

Drawing in uObj

Draw Polygons with indexed vertices

```
void uObj::Draw()
{
    glUniform4f(pCurrentShader->diffuse, diffuse.r,diffuse.g,diffuse.b,diffuse.a);
    glUniform4f(pCurrentShader->ambient, ambient.r,ambient.g,ambient.b,ambient.a);
    glUniform4f(pCurrentShader->specular, specular.r,specular.g,specular.b,specular.a);

    // call vertex buffer
    glBindBuffer(GL_ARRAY_BUFFER, vs);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)0);     // vertex,v
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)12);    // normal,n

    // call face buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);

    glDrawElements(GL_TRIANGLES, 3*nPoly, GL_UNSIGNED_SHORT, (void*)0);

    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER,0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

}
```

Draw Polygons

Drawing in uObj (close VBO handle)

```
void uObj::Draw()
{
    glUniform4f(pCurrentShader->diffuse, diffuse.r,diffuse.g,diffuse.b,diffuse.a);
    glUniform4f(pCurrentShader->ambient, ambient.r,ambient.g,ambient.b,ambient.a);
    glUniform4f(pCurrentShader->specular, specular.r,specular.g,specular.b,specular.a);

    // call vertex buffer
    glBindBuffer(GL_ARRAY_BUFFER, vs);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)0);     // vertex,v
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1,3, GL_FLOAT,FALSE,sizeof(uVertex),(void*)12);    // normal,n

    // call face buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, fs);

    glDrawElements(GL_TRIANGLES, 3*nPoly, GL_UNSIGNED_SHORT, (void*)0);

    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER,0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
}
```

Close VBO handles

uVector is Replaced by uVertex
(uVertex is defined in uVector.h)

- uVertex has
  - uVector position
  - uVector normal

```
class uVertex
{
public:
        uVertex(){}
public:
        uVector v;
        uVector n;
};
```

- uObj has uVertex (instead of uVector)

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

  - pVer[0].v → Position
  - pVer[0].n → Normal

Ex) uGL-04-uGL-box (normal vectors are (0,0,1)

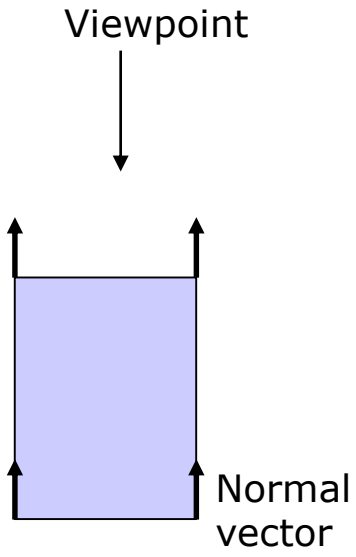
```
void uGL::Loading()
{
    // load shader
    m_sh.Load("solid.vsh","solid.fsh");
    box.MakeBox(1,2,3);
    box.Update();
}
```

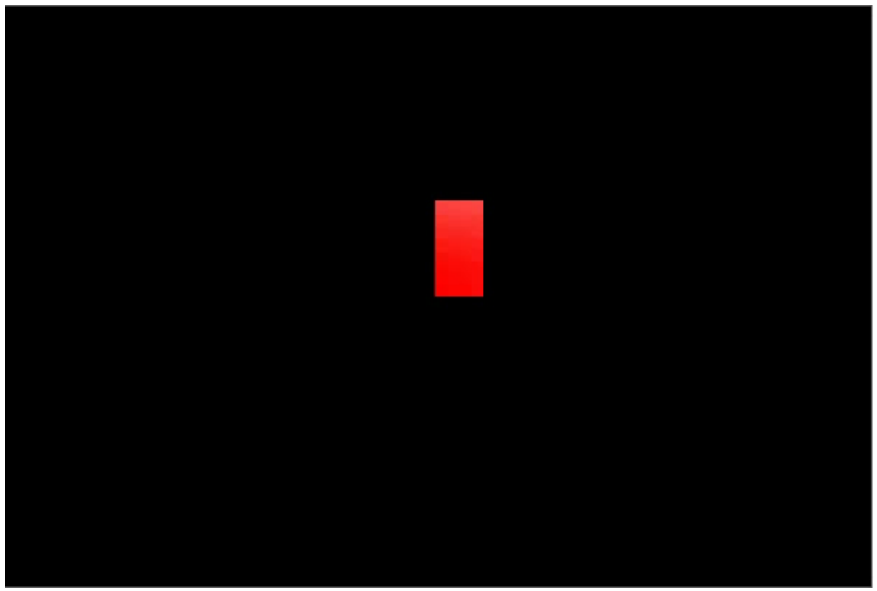
```
void uGL::Draw()
{
    glClearColor(info.r,info.g,info.b,info.a);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // set shader
    hMat h  = cam.T*cam.R;
    glUniformMatrix4fv(pCurrentShader->screen, 1, 0, cam.P.v);
    glUniformMatrix4fv(pCurrentShader->model, 1, 0, h.v);

    {
        box.Draw();
    }

    glFinish();
}
```

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

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

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

    nVer    = nv;
    nPoly   = np;

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

    for (int i=0;i<nv;i++)
    pVer[i].n   = uVector(0,0,1);
}
```


Ex) uGL-04-uGL-box Result



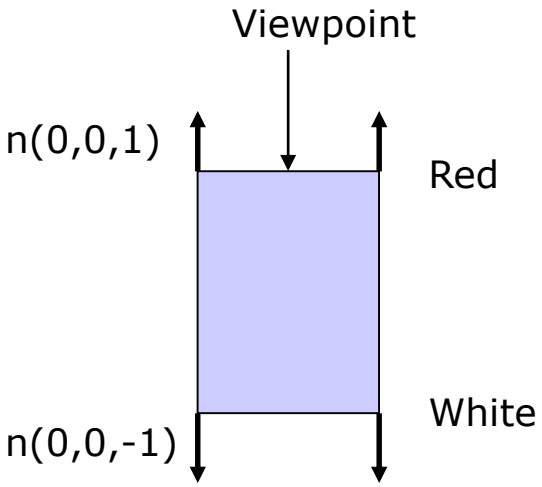


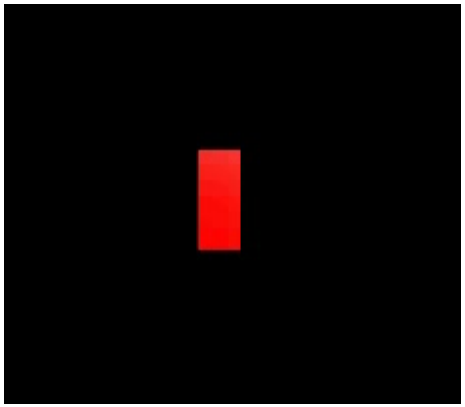
- Set Read at Normal vector direction
- Set White at Negative Normal vector direction
- NORMAL Vector is very important for color display

Ex) uGL-05-uGL-box-normal (Red and White)

```
void uGL::Loading()
{
    // load shader
    m_sh.Load("solid.vsh","solid.fsh");
    box.MakeBox(1,2,3);

    box.pVer[0].n   = uVector(0,0,-1);
    box.pVer[1].n   = uVector(0,0,-1);
    box.pVer[2].n   = uVector(0,0,-1);
    box.pVer[3].n   = uVector(0,0,-1);
    box.pVer[4].n   = uVector(0,0,1);
    box.pVer[5].n   = uVector(0,0,1);
    box.pVer[6].n   = uVector(0,0,1);
    box.pVer[7].n   = uVector(0,0,1);

    box.Update();
}
```

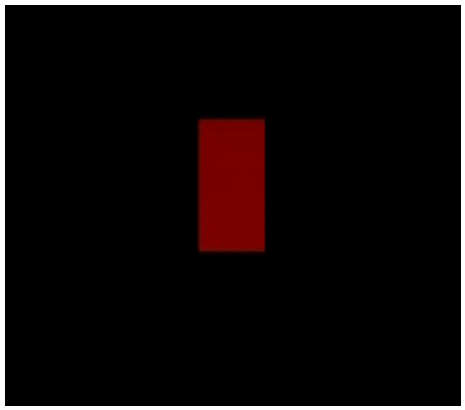
- At the Top, normal vector are (0,0,1)
- At the bottom, normal vectors are (0,0,-1)





Ex) uGL-05-uGL-box-normal2

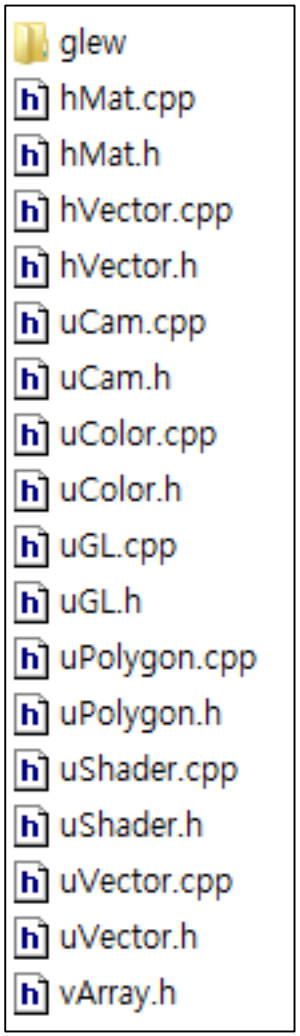
- Shader color is Red and Black



- Normal Vector is crucial point for Color Expression
- You also think that
  - GLSL based Shader can control light and color

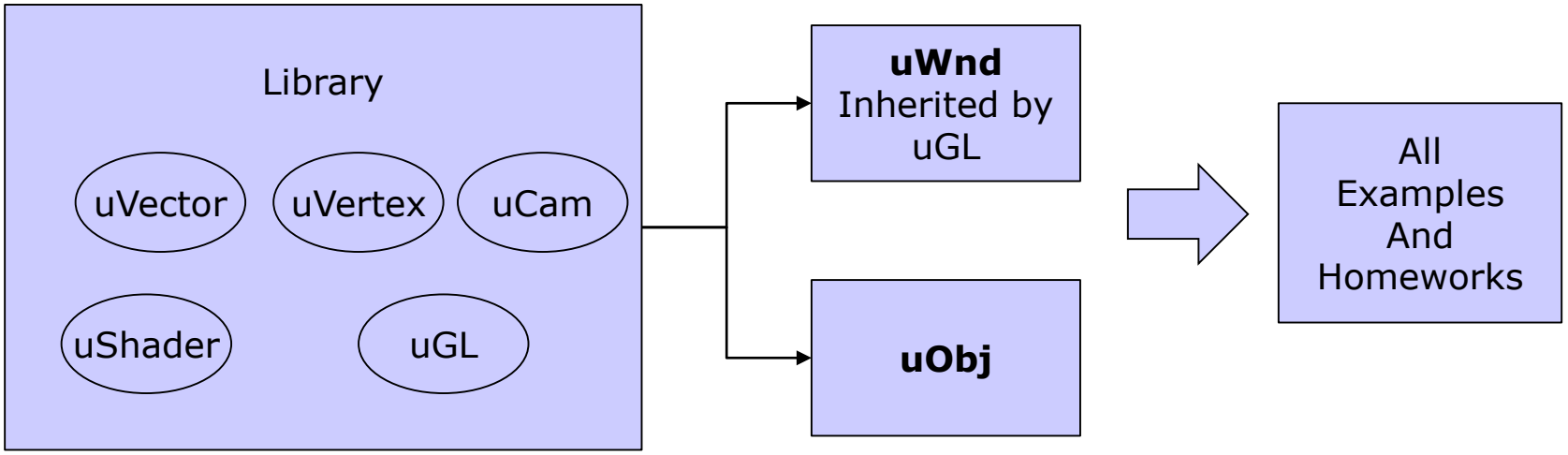
Complete Example in OpenGL
ex) uGL-**06**-uGL-uWnd

Library directory



- Every Examples and Assignments are based on uGL-06-uGL-uWnd example
- Library has uGL, uShader, etc.
- uObj and uWnd will be modified for a variety of demo.

Basic Structure in Examples



- OpenGL interfaces are defined in uGL
- uWnd is inherited by uGL
  - uWnd is for your own purposes
- uObj is for your own object building

Color in OpenGL

5

Normal Vector and
Three Colors
(Diffuse, Ambient, Specular, …)

Illumination: Energy of Physics

- Illumination in Physical world
  - Radiance: the flux of light energy in a given direction
  - Visibility: Light energy falls upon a surface
    - (Remind theHidden surface removal)
  - Energy balance: local balance of energy in a scene
    - The sum of light energy is equal to energy sources.
- Approximation of Colors and Light in graphics
  - **Ambient**: approximation of the global energy
  - Lambertian: approximation of **diffuse** interaction between materials and lights
  - Phong: approximation of **specular** effects.

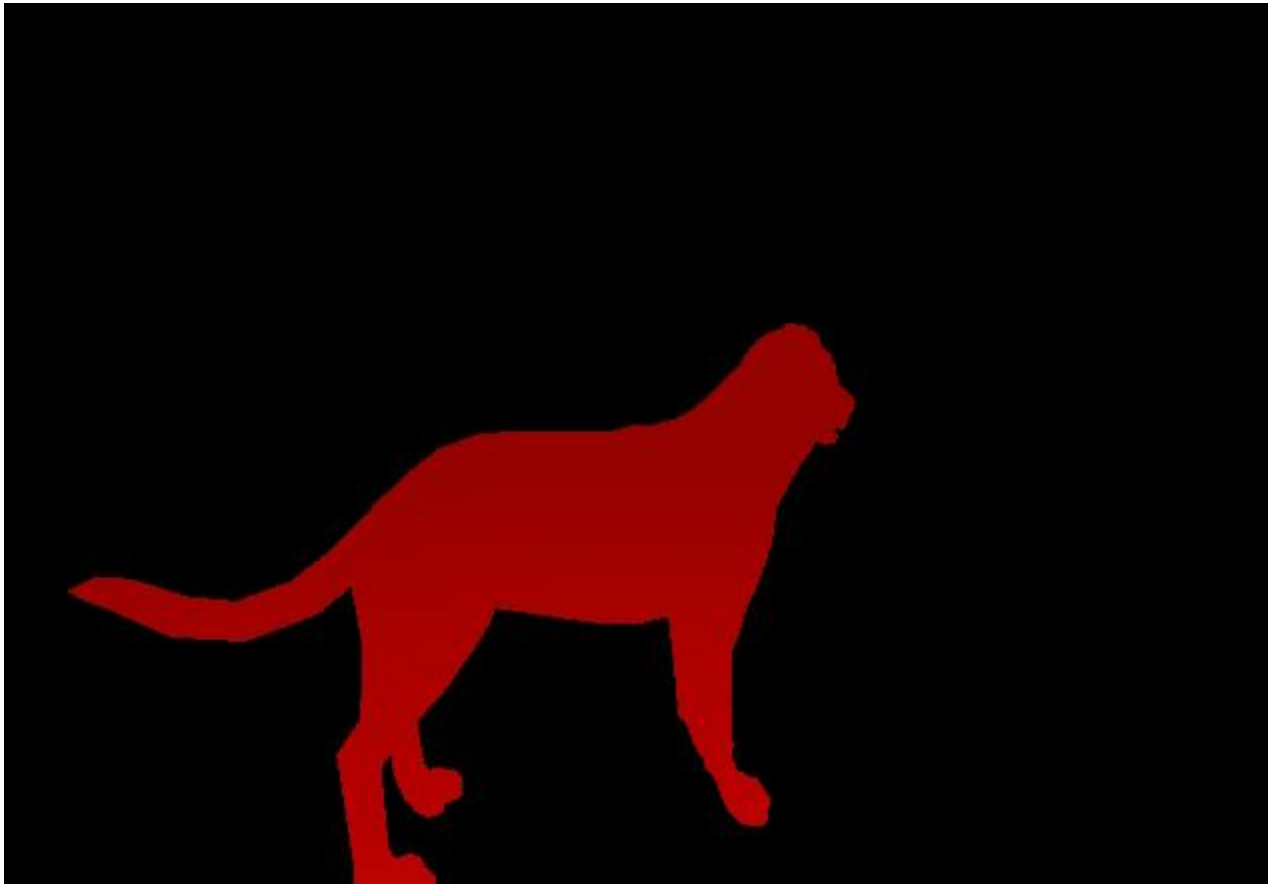
Computer Graphics has Three Color Types

- Ambient : the color of an object in shadow
  - Without no additional Light source
- Diffuse: the color of an object surface
  - Apple is Red. (Diffuse color is red)
- Specular: shiny color on the surface by light source.

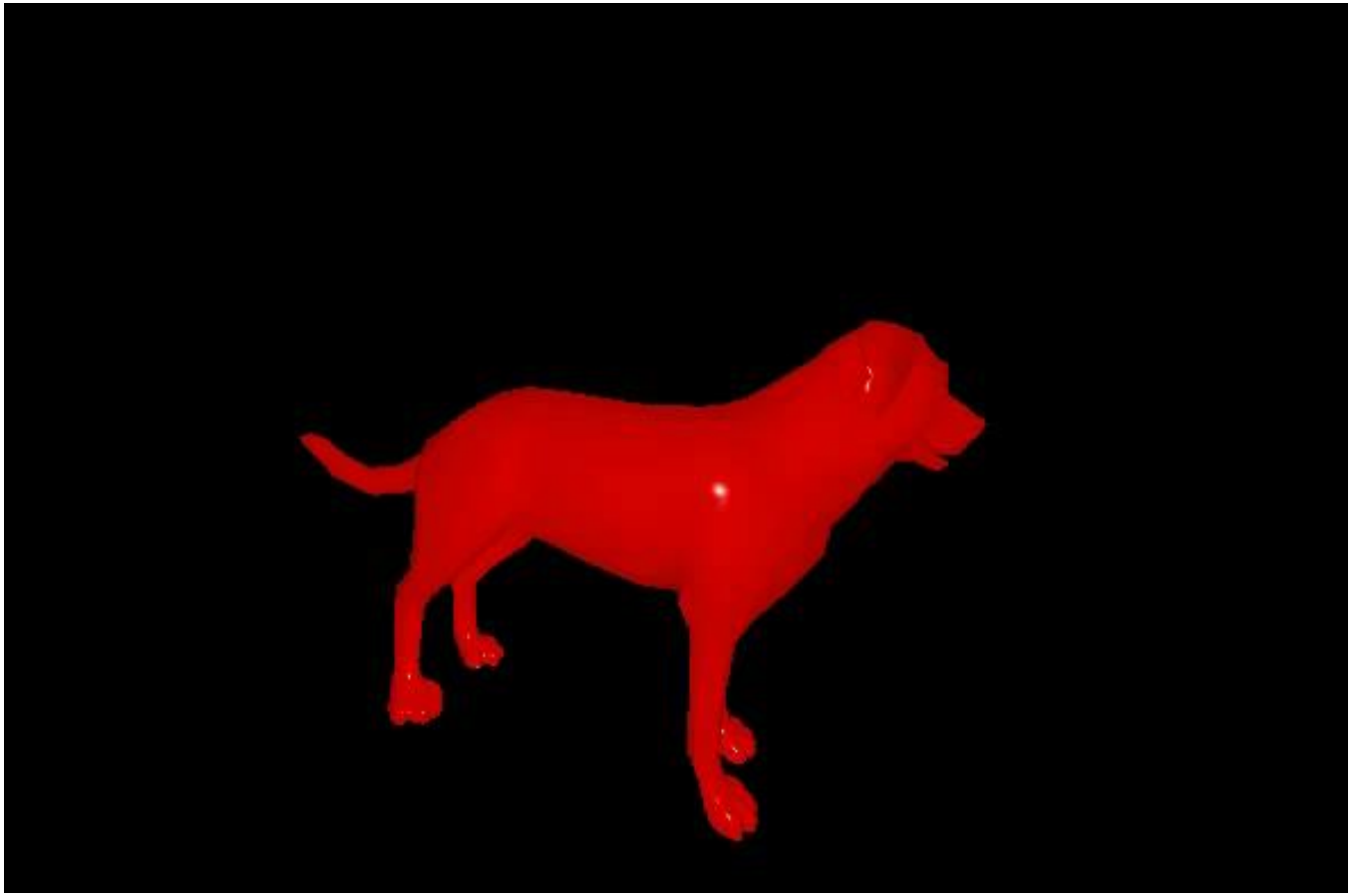
All Normal Vectors are Same

ex) normal vectors are (0,0,1) against viewpoint direction

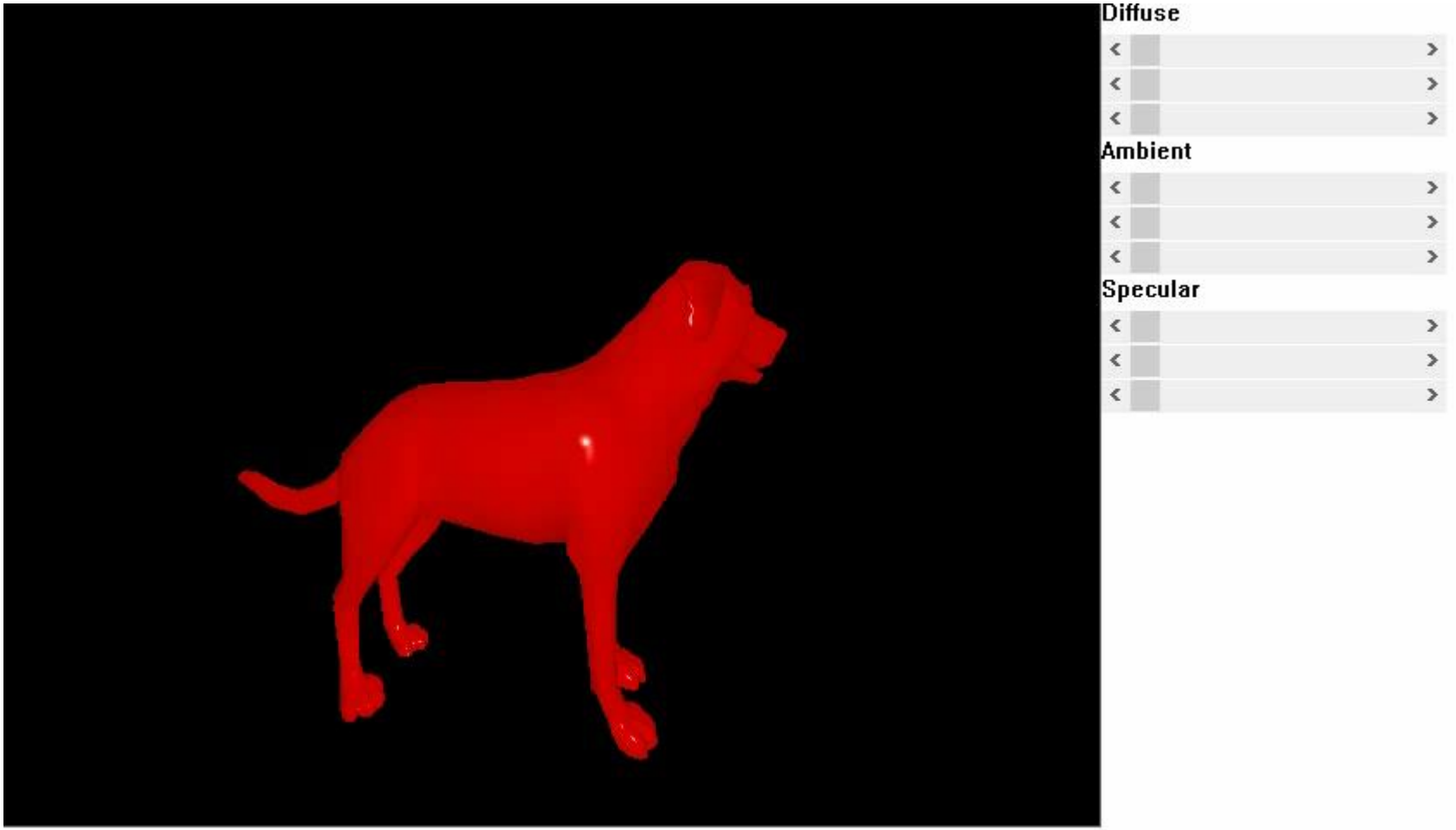
- testZ.exe



All Normal Vectors are Calculated for Each Polygon



Ambient, Diffuse, and Specular



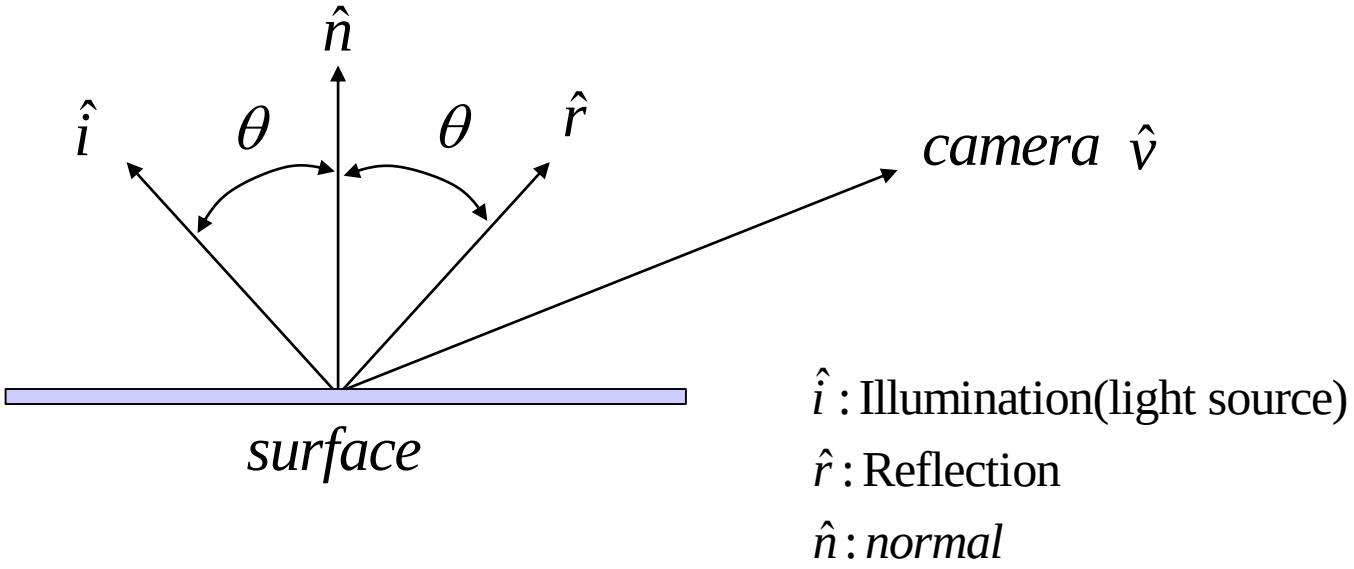
OpenGL Vs. Ray Tracing
Calculation of Light Energy in a Scene

- OpenGL
  - Uses diffuse, ambient, and specular → Approximation
  - Reflection is NOT real..
  - Light sources are limited

- Ray tracing
  - Calculating colors by following a ray.
  - The colors of each pixel is determined by calculating geometries and light sources → Higher computation

  - You will do it at the latter part of the semester.

Lambertian Reflection Model

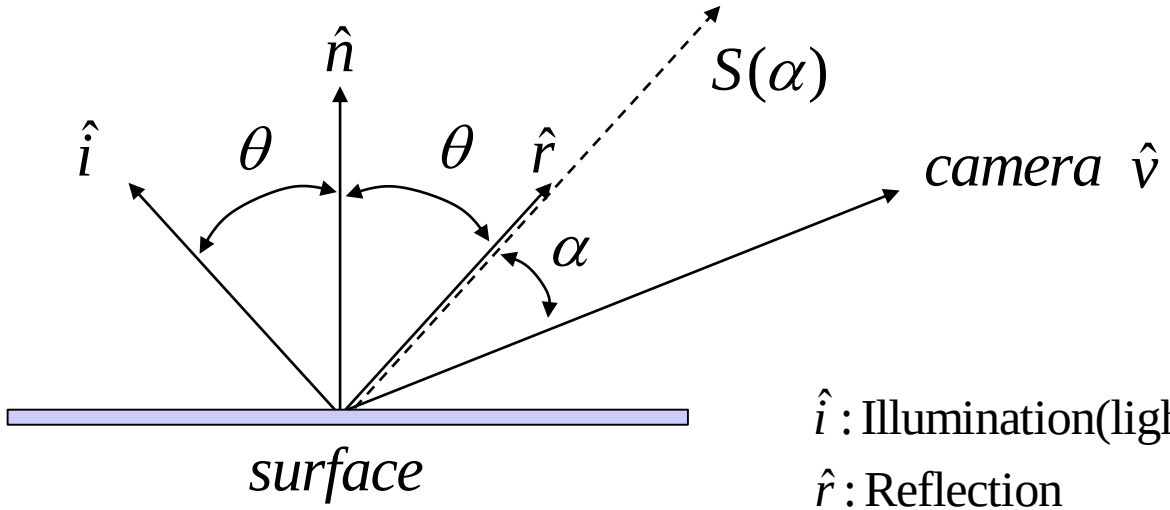


- Lambertian model defines Diffuse color
  - by Only Normal vector

$$\cos\theta = \hat{i} \circ \hat{n}$$

- OpenGL rendering calculates cosine for diffuse color

Phong's Reflection Model
(Phong Shading for Specular color)



$\hat{i}$: Illumination(light source)

$\hat{r}$: Reflection

$\hat{n}$: *normal*

- Phong model determines colors:

Reflection vector

$$\frac{\hat{i}+\hat{r}}{2}=\left(\hat{i}\circ\hat{n}\right)\hat{n}$$

$$\therefore \hat{r}=2\left(\hat{i}\circ\hat{n}\right)\hat{n}-\hat{i}$$

Cosine for specular color

$$\therefore \cos\alpha=\hat{r}\circ\hat{v}$$

Specular parameter of surface (Glass: high, wood: low)

$$\mathrm{S}(\alpha)=\cos\alpha^{s}$$

Phong's Specular in Your Example

- Open solid.fsh (GLSL fragment shader)

```
float specAngle = max(dot(reflectDir, viewDir), 0);
specular = pow(specAngle, 100.0);
```

$$\frac{\hat{i}+\hat{r}}{2}=\left(\hat{i}\circ\hat{n}\right)\hat{n}$$

$$\therefore \hat{r}=2\left(\hat{i}\circ\hat{n}\right)\hat{n}-\hat{i}$$

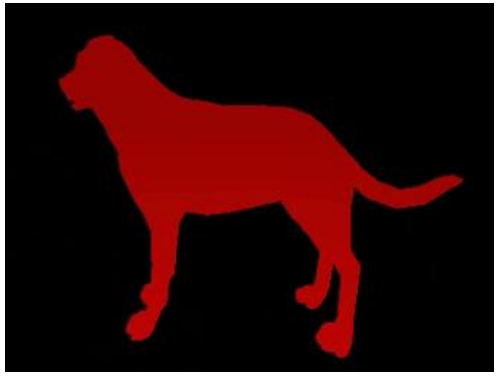
$$\therefore \cos\alpha=\hat{r}\circ\hat{v}$$

$$S(\alpha)=\cos\alpha^{s}$$

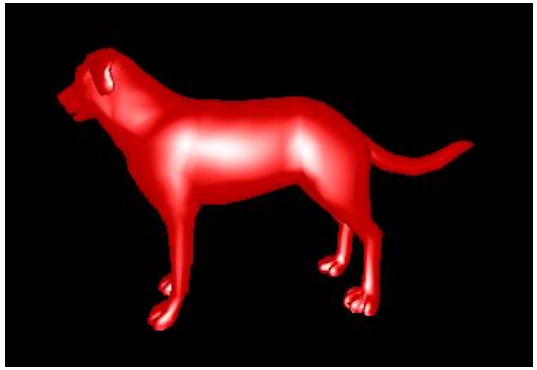
- GLSL programming uses
  - the basic concept of Ray Tracing Method.

History behind OpenGL

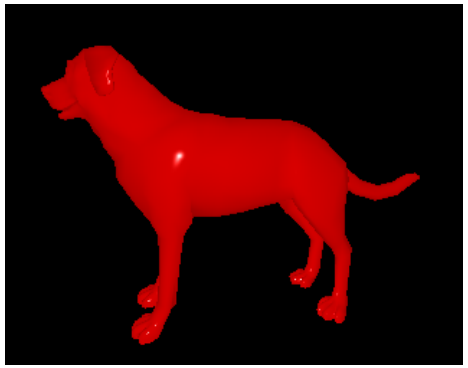
- 1st period(~2003) : Color by texture mapping. No light.
- 2nd period(~2010): Ambient, Diffuse, and Specular
- 3rd period(~2018): GLSL based Phong Shading



No light effect



Specular



Phong shading

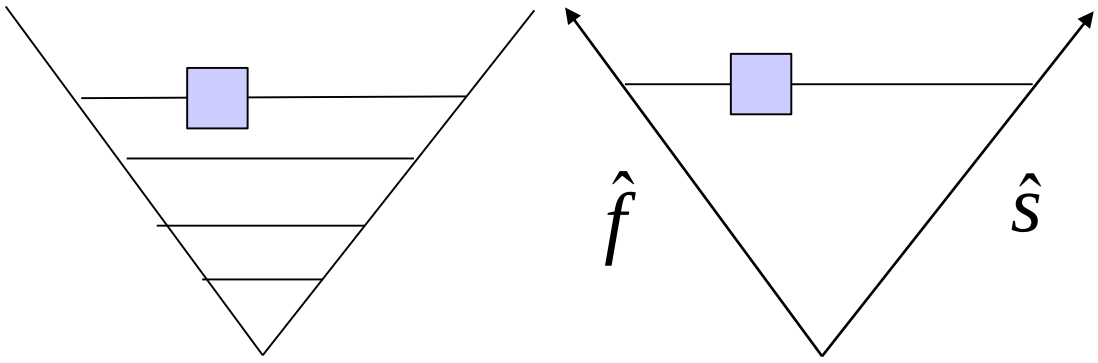
- GLSL is a simple tool for mimicking Ray Tracing
  - Such as Phong shading, Shadow, Cartoon Rendering

Shading with Normal Vector

- Lambertian model (diffuse color)
  - Uses cosine function between surface normal and light.

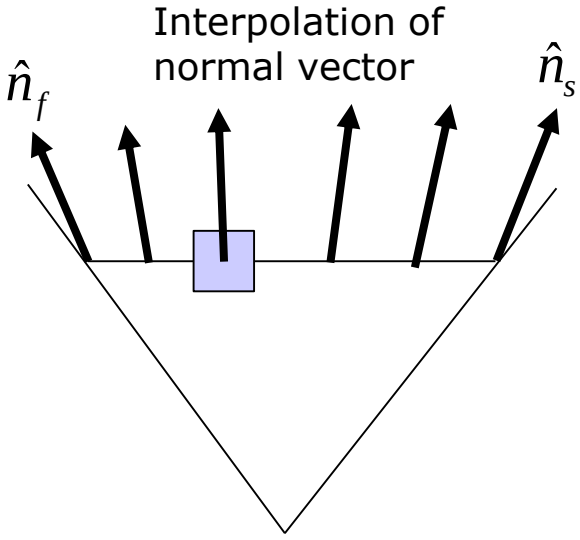
$$\cos\theta = \hat{i} \circ \hat{n}$$

- Shading approximates colors between vertices.



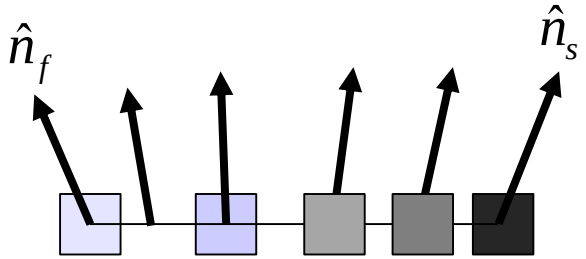
Fill Polygon

$$\hat{v} = \lambda \hat{f} + (1-\lambda)\hat{s}$$



$$\hat{n} = (1-\lambda)\hat{n}_f + \lambda\hat{n}_s$$

Why Shading uses Interpolation of Vertex Normal Vectors

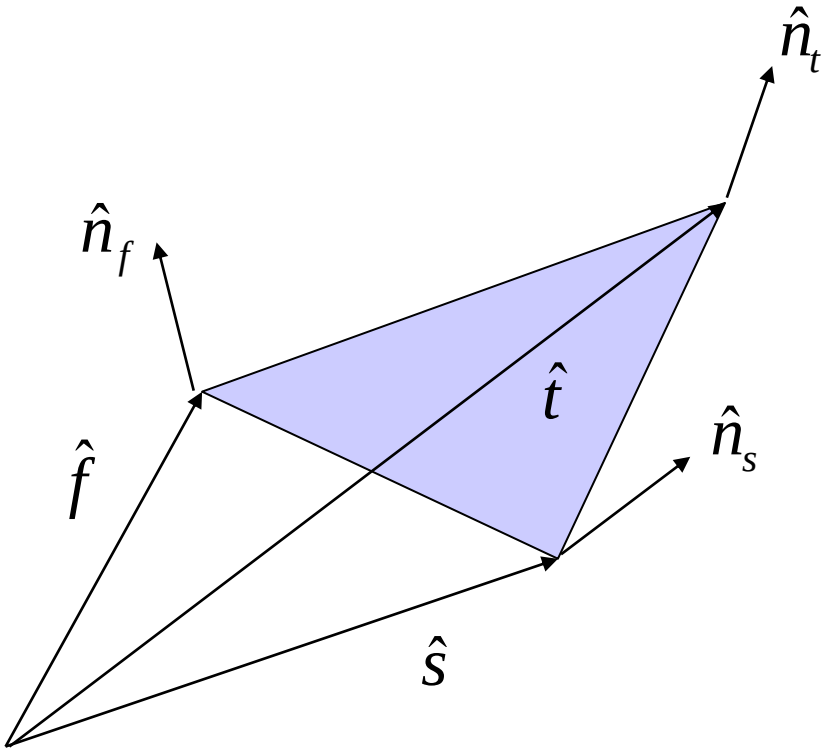


$$\hat{n} = (1-\lambda)\hat{n}_f + \lambda\hat{n}_s \rightarrow \cos\theta = \hat{i} \circ \hat{n}$$

- GPU memory has been limited
  - 1. Only vertex has graphical information
  - 2. If vertex has normal vector,
  - 3. Then, interpolation of normal vectors varies smooth color transition
  - (4. But today we have GLSL for Pixel-based color management)

Shading with Normal

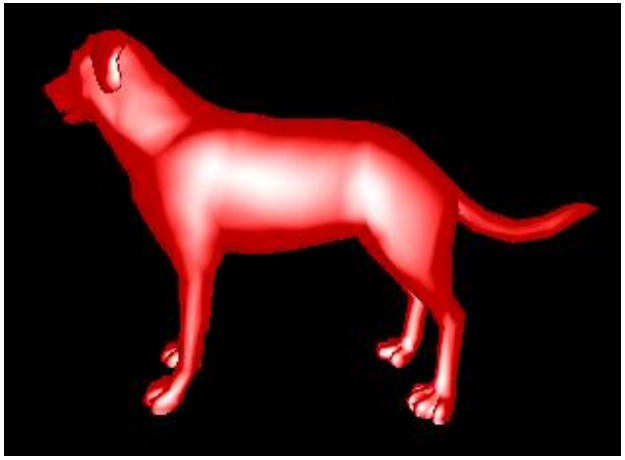
- Definition of Vertex Normal
  - Each vertex has its own normal
  - Any Normal will be good(Thus, it is NOT a surface normal)
- **Normal vector in shading means the color variance**



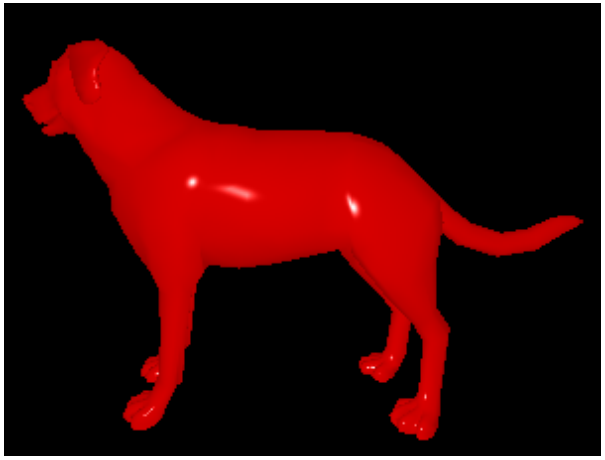
Famous Three Shading

- 1. Flat shading
- 2. Gouraud shading
- 3. Phong shading (Phong Reflection Model)

Flat Shading



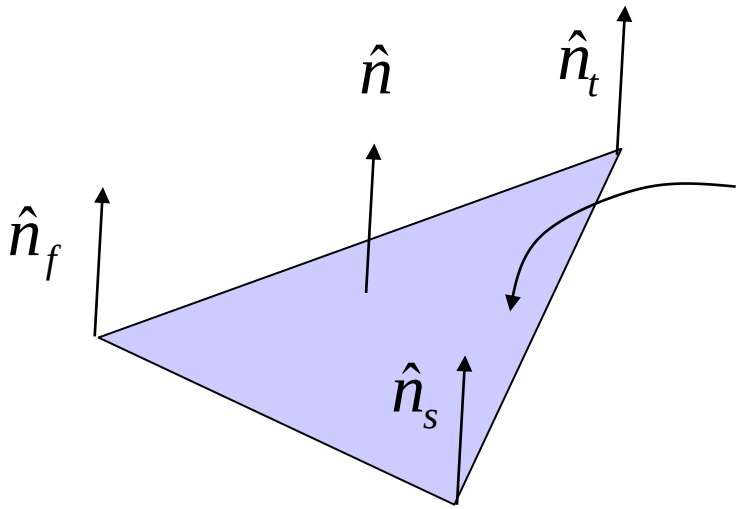
Gouraud Shading



Phong Shading

Flat Shading: How to express surfaces with **Flatness**

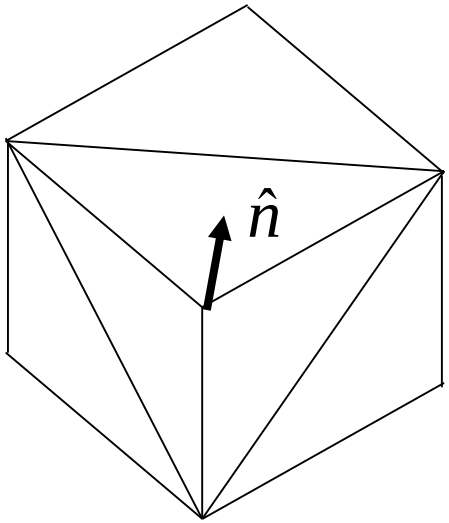
- All vertex normal in a polygon are same

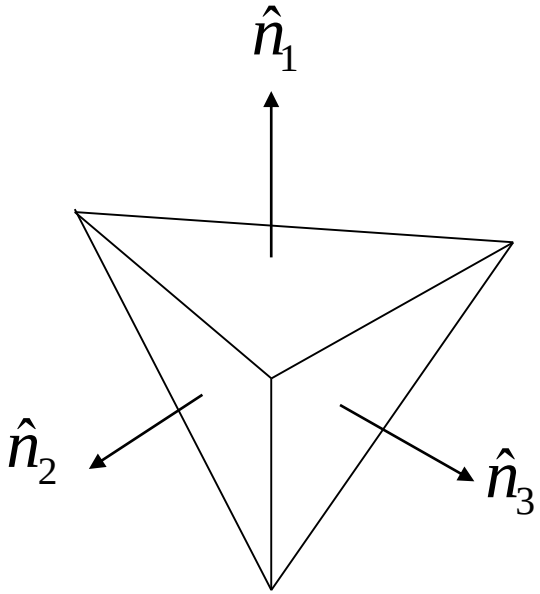


$$\hat{n} = (\hat{t} - \hat{s}) \times (\hat{f} - \hat{s})$$
$$\hat{n}_f = \hat{n}_s = \hat{n}_t = \hat{n}$$

$\Rightarrow$ $\cos\theta = \hat{i} \circ \hat{n}$ $\Rightarrow$ All pixel colors in the polygons are same!

Gouraud Shading
How to express surfaces with **Smoothness**





$$\hat{n} = \frac{\hat{n}_1 + \hat{n}_2 + \hat{n}_3}{3}$$

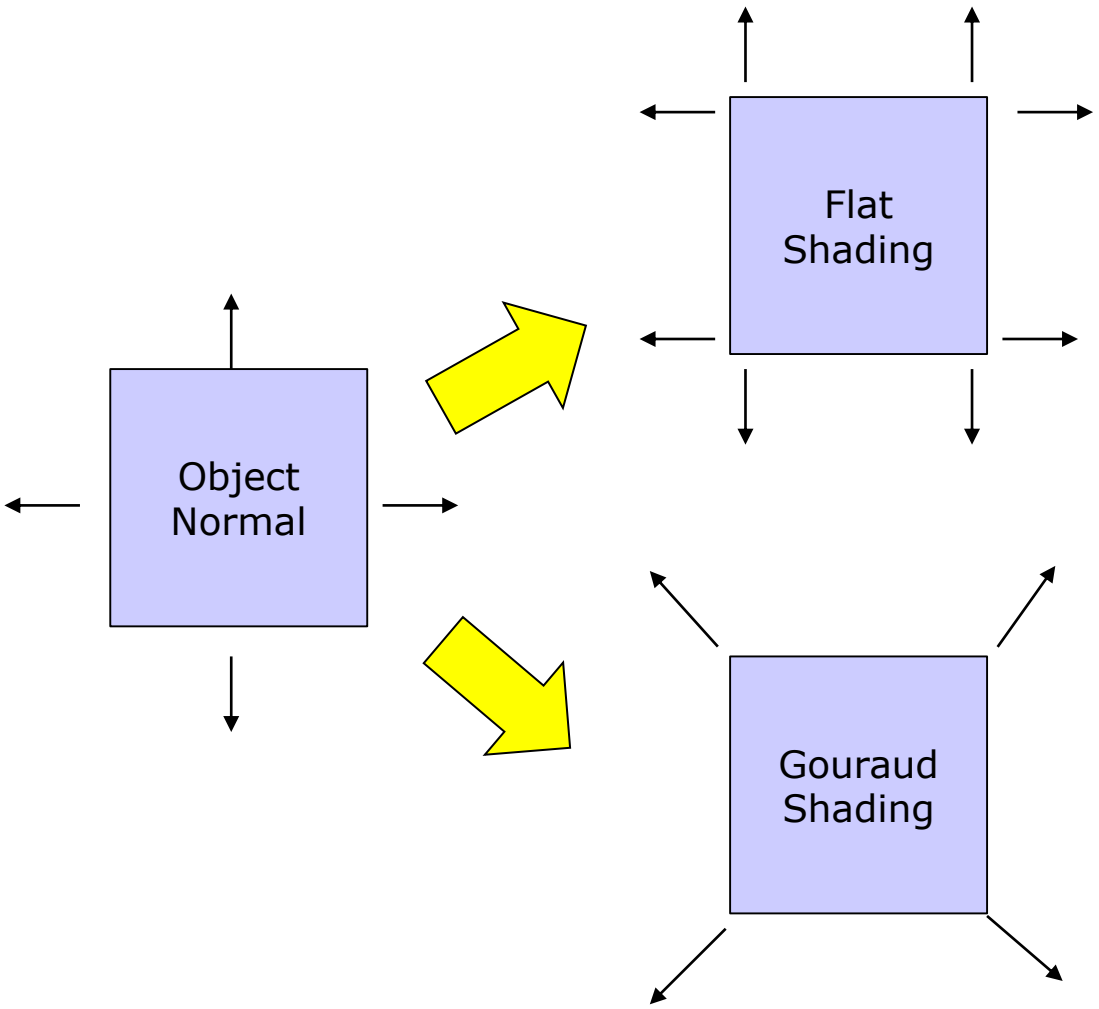
$$= \frac{1}{N} \sum_{i \in Neighbor}^{N} \hat{n}_i$$

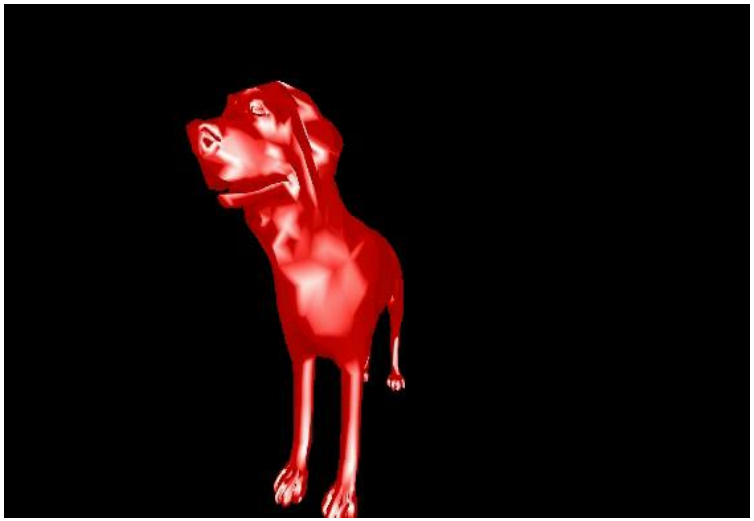
Gouraud shading averages

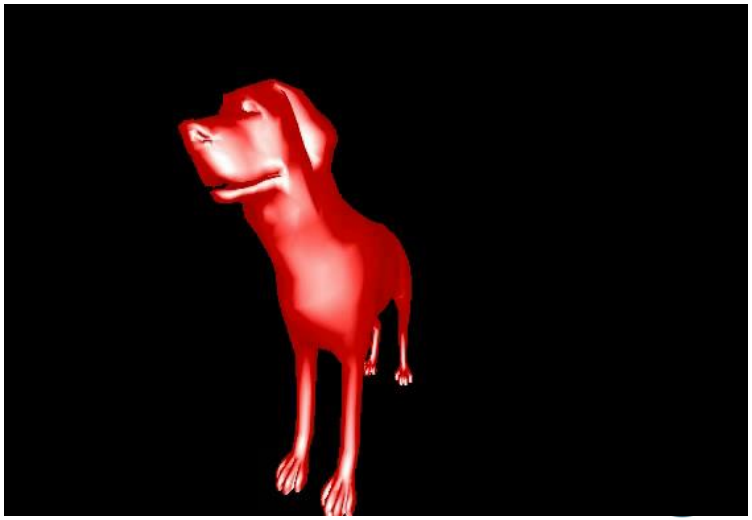
all neighboring polygons' normal vectors

→ Normal vectors are smooth

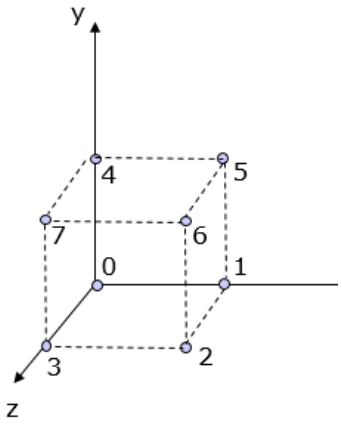
Concept of Flat Vs. Gouraud Shading







Flat Shading: Pseudo code



- Average of vertex's Normal

1. Create new vertex with nPoly * 3
 uVertex *pNew = new uVertex[nPoly*3]

2. k = 0
   For i = 0 to nPoly
      uPolygon p = pPoly[i]
      n = GetNormal(p)

      pNew[k].v = pVer[p.f];
      pNew[k].n = n;
      k++;
      pNew[k].v = pVer[p.s];
      pNew[k].n = n;
      k++;
      pNew[k].v = pVer[p.t];
      pNew[k].n = n;
      k++;

3. delete pVer
   nVer = nPoly*3
   pVer = pNew;

Example
1. pNew=new uVertex[36]
2. For i = 0 to nPoly
3. ex)
   polygon 0 has { 3, 2, 6}
   n = GerNormal(Polygon 0)
   pNew[0].v = pVer[3]
   pNew[0].n = n
   pNew[1].v = pVer[2]
   pNew[1].n = n
   pNew[2].v = pVer[6]
   pNew[2].n = n

   polygon 1 has { 3, 6, 7}
   n = GerNormal(Polygon 0)
   pNew[i].v = pVer[3];
   pNew[i].n = n
   ….

Gouraud Shading: Pseudo code

- Average of vertex's Normal

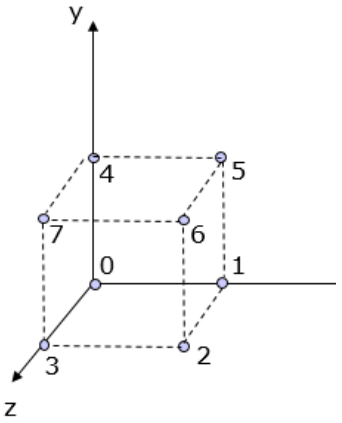
1. Create new vertex buffer with vertex number
   uVector *psum = new uVector[nVer]

2. Create new int buffer for counting overlapped vertex.
   int *nsum = new int [nVer]

3. For i = 0 to nPoly
   uPolygon p = pPoly[i]
   n = GetNormal(p)

   psum[p.f] = psum[p.f] + n; nsum[p.f]++;
   psum[p.s] = psum[p.s] + n; nsum[p.s]++;
   psum[p.t] = psum[p.t] + n; nsum[p.t]++;

4. For i= to nVer
   psum[i] = psum[i]/nsum[i]
   pVer[i].n = psum[i].Unit()



Example
1. pSum=new uVector[8]
2. nPoly = 2*8

polygon 0 = { 3,2,6}
polygon 1= { 3,6,7}
….

pSum[3] += polygon 0's normal
…
pSum[3]+= polygon 1's normal
…