Computer Graphics and Programming

Lecture 9
Texture Mapping

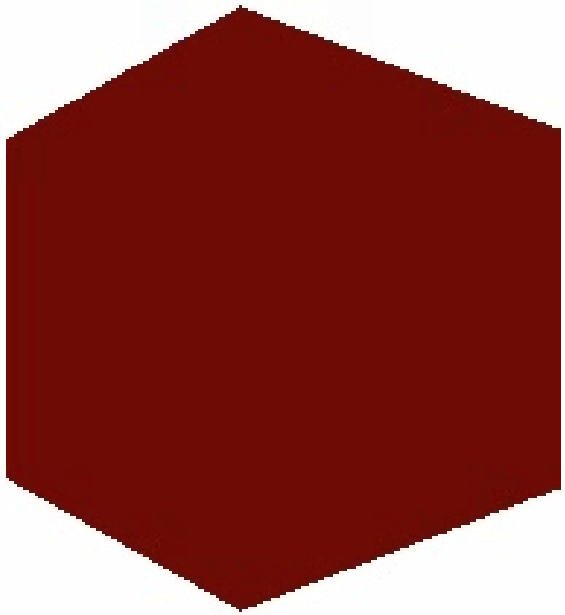
Jeong-Yean Yang

2020/12/8

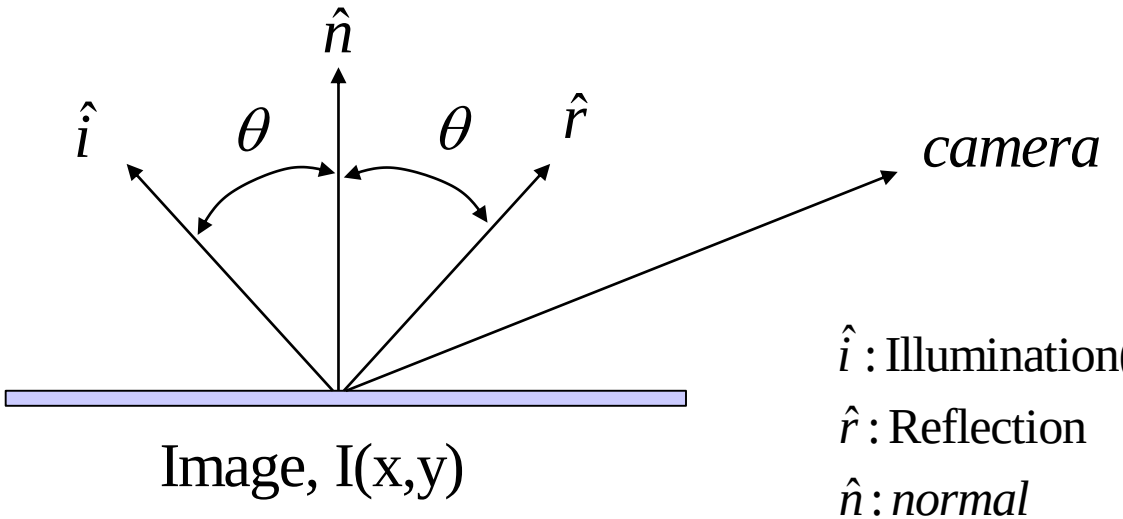
0 What is Texture Mapping?

OpenGL supports for Three Colors + Texture

- Three Colors
  - Ambient, diffuse, specular
- Texture(bitmap)
  - Image



Texture(Image) mapping is added on Diffuse color



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

$\hat{r}$: Reflection

$\hat{n}$: *normal*

- Colors with texture mapping

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

$$\therefore \text{color} = \text{Image} + \text{diffuse} = I(x, y) + \hat{i} \circ \hat{n}$$

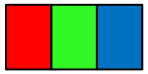
1 How to handle Image buffer

Various Image Types

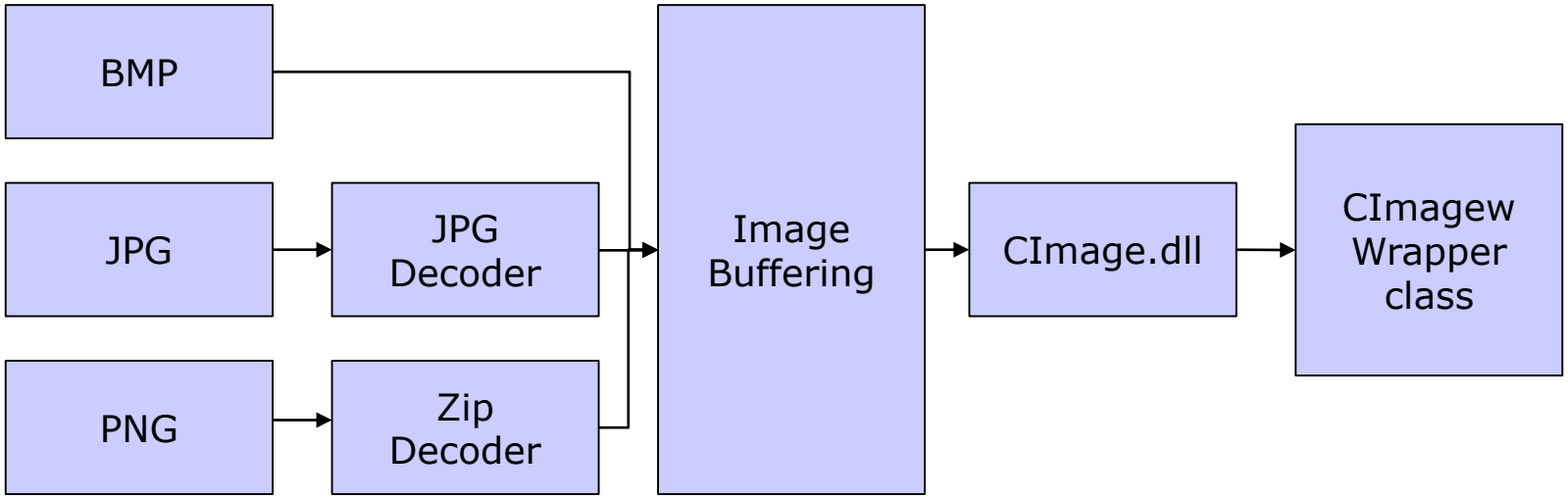
- Various types of Image formats
  - Bmp, jpg, gif, png, pcx, tif, and so on

- BMP, JPG, PNG are famous image formats in Graphics
  - 24bit: BMP, JPG , 32bit: PNG
- One pixel of image is described by 8, 16, 24, 32 bit.
  - RGB = 24bit
  - RGBA = 32bit
  - Gray = 8bit
  - RGB(RGB565) = 16 bit
    - R =0~31
    - G=0~63
    - B=0~31

OLED pixel

1 pixel = 

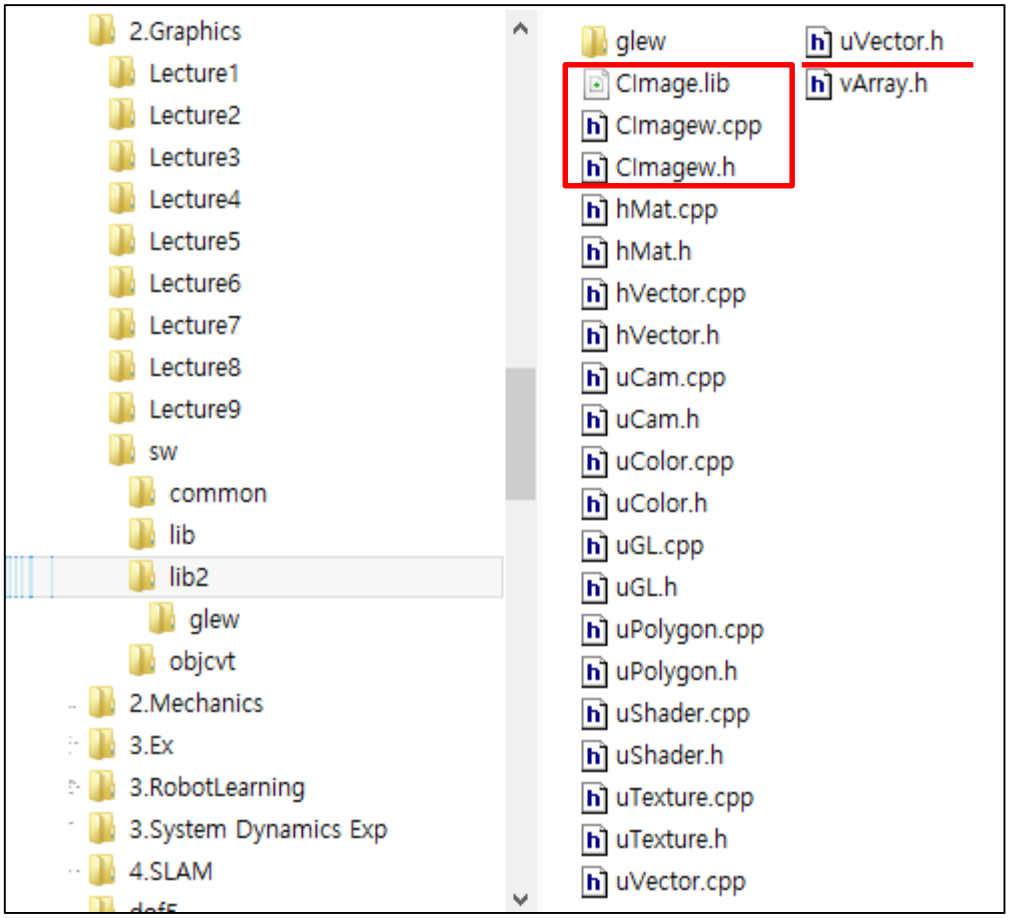
This Class Provides Image Library



- CImagew is a wrapper class for convenient purpose
  - Easy to access RGB or RGBA buffer

- **From Lecture 9, we will use ‘lib2’ instead of ‘lib’**

‘lib2’ for Texture Mapping

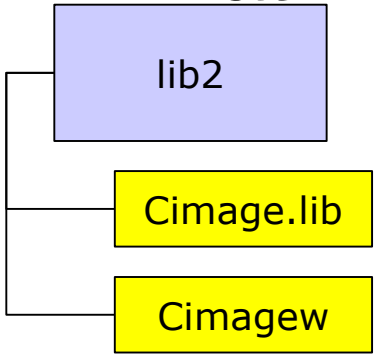


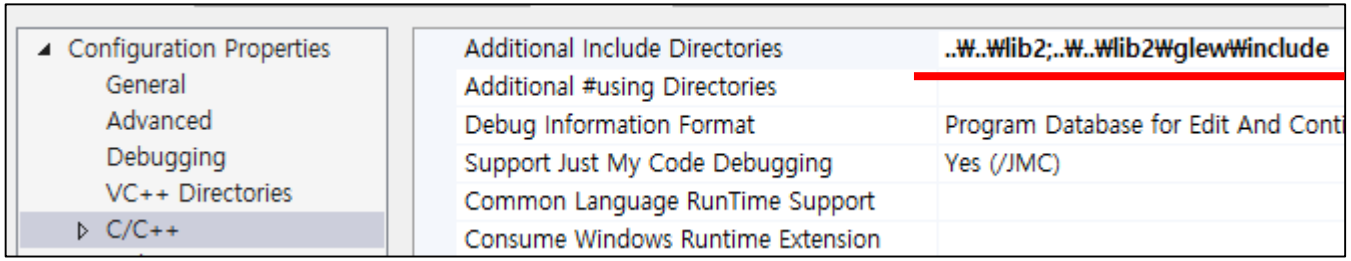
- lib for solid color
- lib2 for image handling

- Changes in lib2

```
// Texture Mapping
class uVertex // Positio
{
public:
        uVertex(){}
public:
        uVector v;
        uVector n;
        struct
        {
            float u,v;
        } tx;
};
```

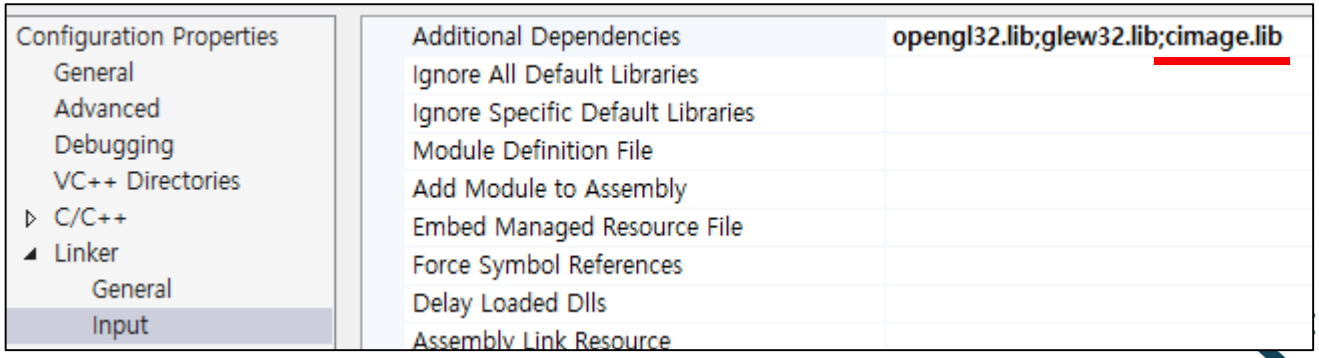
Ex) uRT-01-Basic
Path Setting for Compiling and Linking



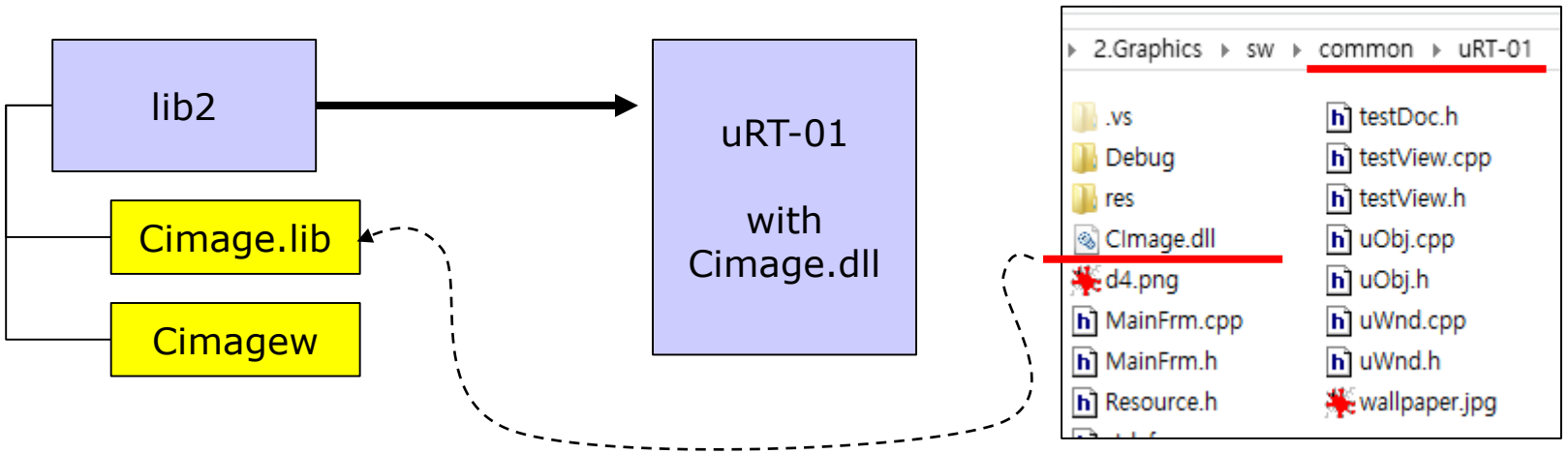


- **"../../lib2" is added for "Additional Library Directories"**

cimage.lib for cimage.dll
Is added



Ex) uRT-01-Basic
DLL : Interface library and header file



- If we use DLL,
  - We need interface library and wrapper class
- glew32.dll ← interface(glew32.lib) + include(glew.h)
- Cimage.dll ← interface(cimage.lib) + Cimagew class

Ex) uRT-01-Basic
Image Loading

```
#include "CImagew.h"
class CtestView : public CView
{
protected: // create from serialization only
    CtestView() noexcept;
    DECLARE_DYNCREATE(CtestView)

// Attributes
public:
    CtestDoc* GetDocument() const;
// Operations
public:
    CImagew     img;
```

1. Variable Declaration

```
void CtestView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    //img.Load("d4.png");
    img.Load("wallpaper.jpg");
}
```

2. Image loading

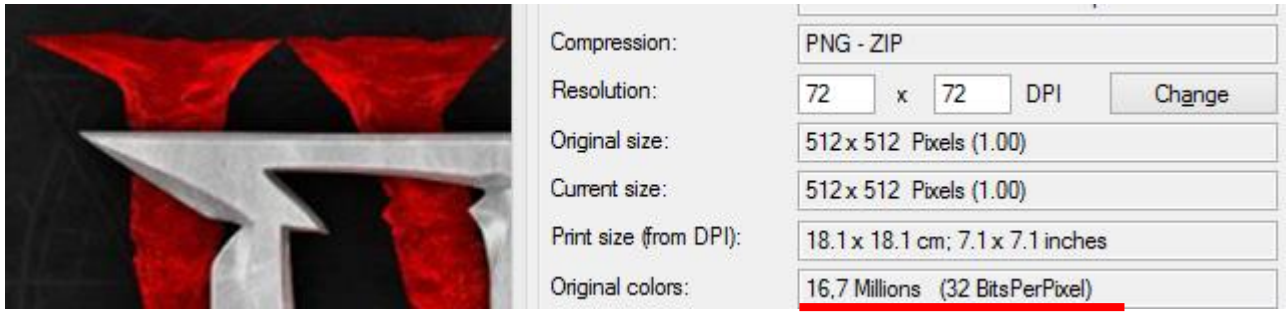
```
void CtestView::OnDraw(CDC* pDC)
{
    CtestDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    img.Draw(pDC, 0,0);
}
```

3. Draw function

- 1. Declare CImagew
- 2. Image loading
- 3. Draw an image at point (0,0)

Image Buffering

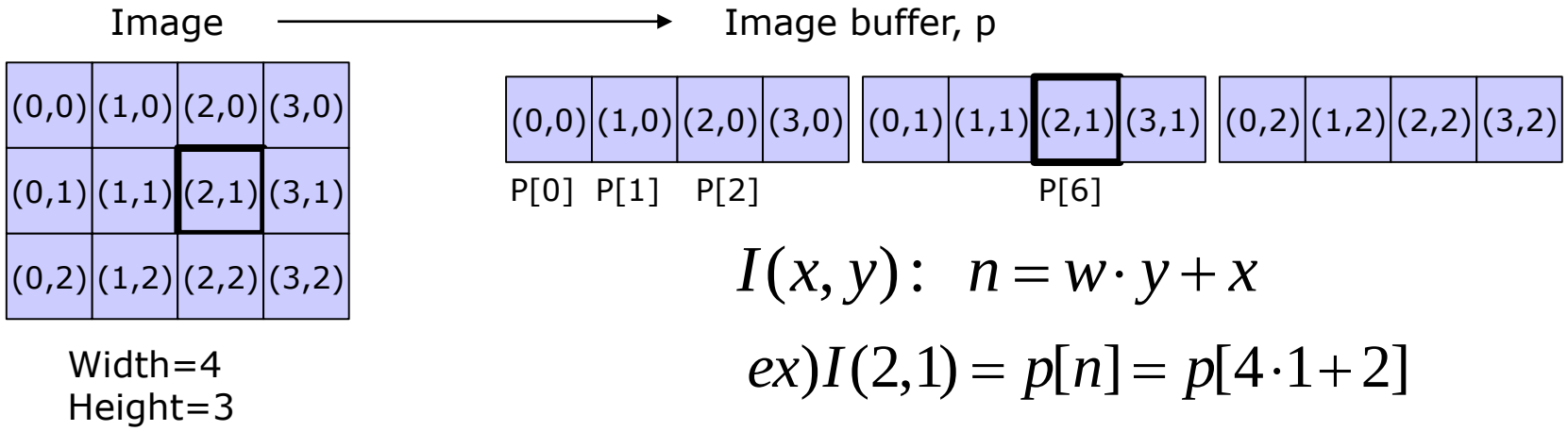
- Confirm Image bit first before image buffering.
- Ex) d4.png → 32bit → RGBA



- wallpaper.jpg → 24bit → RGB

Ex) uRT-02-Buffering-24bit-BGR

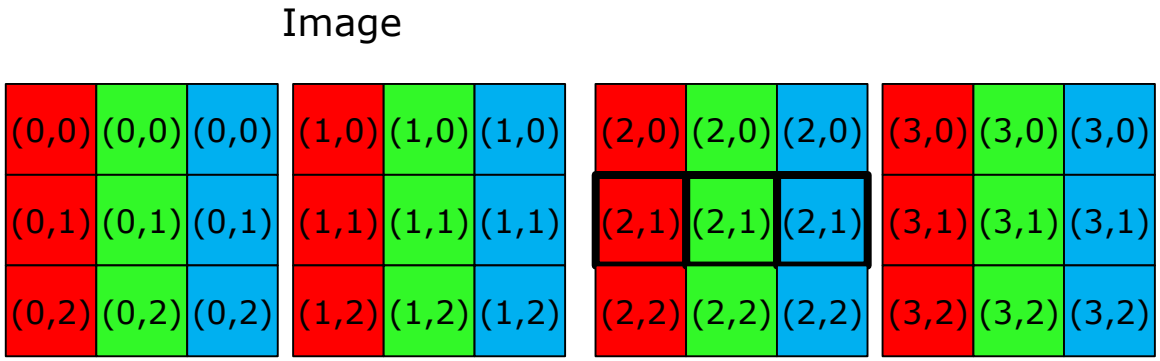
- 8bit Image buffering



$$I(x,y):\quad n = w\cdot y + x$$

$$ex)\, I(2,1) = p[n] = p[4\cdot 1 + 2]$$

- 24 bit image buffering



$$\mathrm{R}(x,y):\quad \mathrm{r} = 3\cdot w\cdot y + 3\cdot x$$

$$\mathrm{G}(x,y):\quad \mathrm{g} = 3\cdot w\cdot y + 3\cdot x + 1$$

$$\mathrm{B}(x,y):\quad \mathrm{b} = 3\cdot w\cdot y + 3\cdot x + 2$$

$$ex)\, \mathrm{R}(2,1) = p[r] = p[3\cdot 4\cdot 1 + 3\cdot 2]$$

$$\mathrm{G}(2,1) = p[\mathrm{g}] = p[3\cdot 4\cdot 1 + 3\cdot 2 + 1]$$

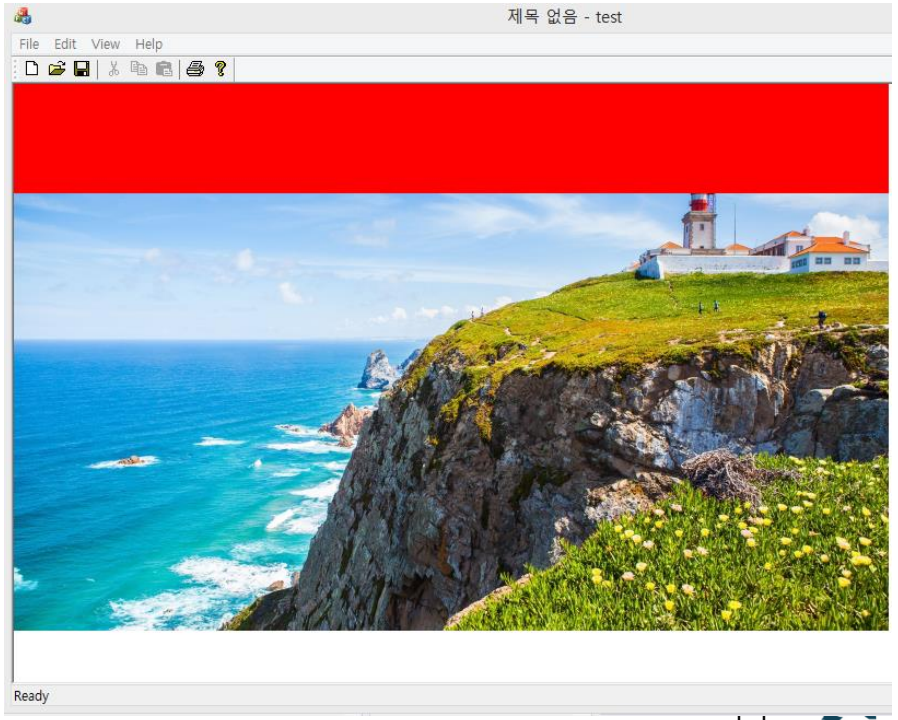
$$\mathrm{B}(2,1) = p[\mathrm{b}] = p[3\cdot 4\cdot 1 + 3\cdot 2 + 2]$$

JPG uses BGR instead of using RGB

- JPG decoding uses BGR in general
  - For proofed Image processing, you must understand everything

```
BYTE *p = img.GetBuffer();

// 24bit jpg with BGR
for (int j=0;j<100;j++)
for (int i=0;i<img.w;i++)
{
    p[ 3*img.w*j + 3*i] = 0;
    p[ 3*img.w*j + 3*i+1] = 0;
    p[ 3*img.w*j + 3*i+2] = 255;
}
```

JPEG is NOT R8G8B8,
but B8G8R8 model



32bit RGBA Vs. BGRA

- Ex) uRT-03-Buffering-32bit-BGRA

R8G8B8A8

$$R(x,y): \quad r = 4 \cdot w \cdot y + 4 \cdot x$$
$$G(x,y): \quad g = 4 \cdot w \cdot y + 4 \cdot x + 1$$
$$B(x,y): \quad b = 4 \cdot w \cdot y + 4 \cdot x + 2$$
$$A(x,y): \quad a = 4 \cdot w \cdot y + 4 \cdot x + 3$$

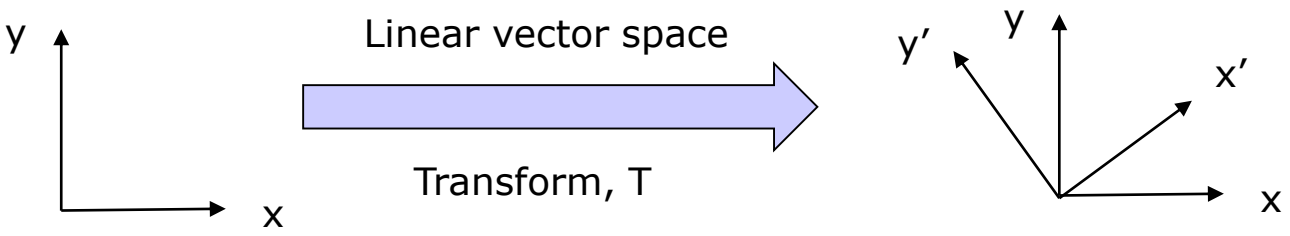
B8G8R8A8

$$B(x,y): \quad r = 4 \cdot w \cdot y + 4 \cdot x$$
$$G(x,y): \quad g = 4 \cdot w \cdot y + 4 \cdot x + 1$$
$$R(x,y): \quad b = 4 \cdot w \cdot y + 4 \cdot x + 2$$
$$A(x,y): \quad a = 4 \cdot w \cdot y + 4 \cdot x + 3$$

- For example, “wallpaper32.png” is 32 bit PNG

RGB Color Space

- RGB is linear? Is it Independent?



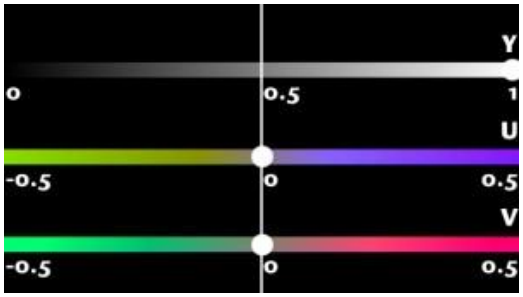
$$\hat{X} = \begin{pmatrix} x & y \end{pmatrix}^T$$

$$\hat{X}' = T\hat{X}$$

- Color that we see is NOT linear
  - Red(10%) + Red(20%) ≠ Red(30%) → Non Linear
- RGB buffer in images are linear
- Thus, Nonlinear Color Scene → Capture → Storing linearly → RGB buffer is linear

Y'UV Space

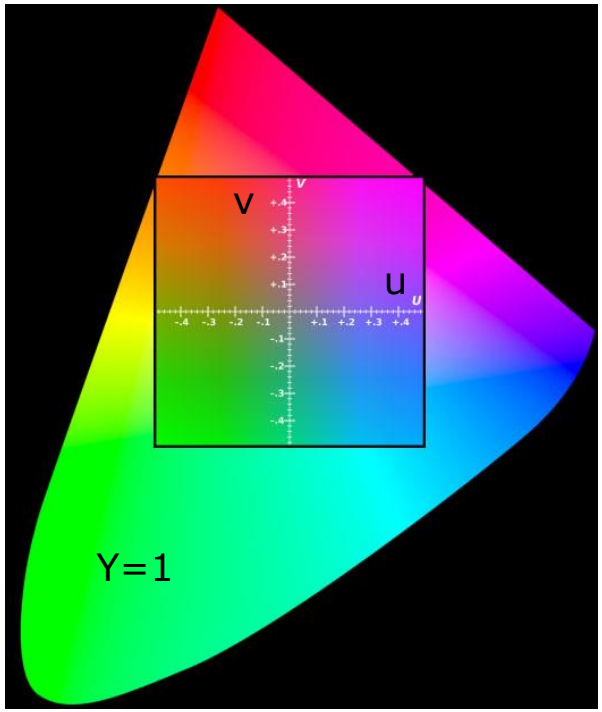
- YUV is transformed by Y'UV space
  - Y' : luminance component
  - UV: Chrominance component
  - U :Blue projection
  - V: Red projection



$$\begin{pmatrix} Y' \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1473 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

```
A =

    0.2990    0.5870    0.1140
   -0.1473   -0.2889    0.4360
    0.6150   -0.5150   -0.1000

?rank(A)

ans =

     3
```

Rank =3
Y'UV is a linear transform of RGB

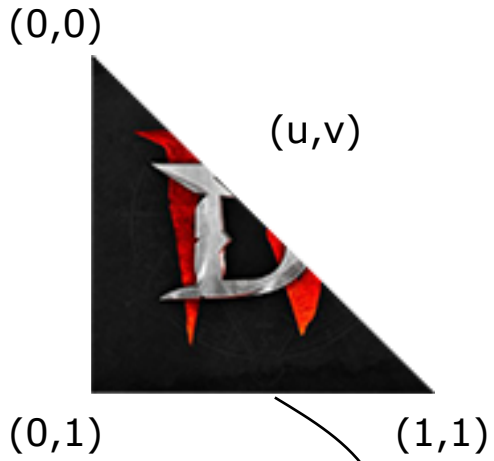


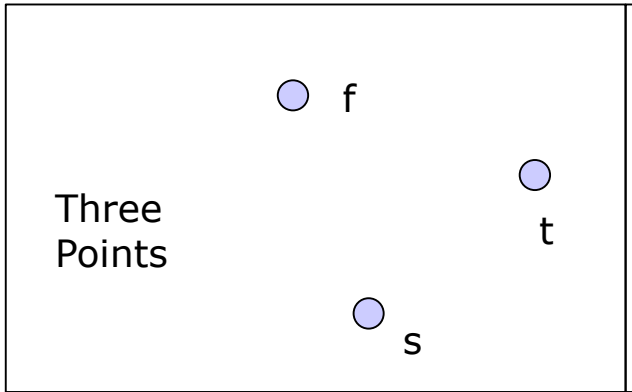
2 Texture Mapping in OpenGL with lib2

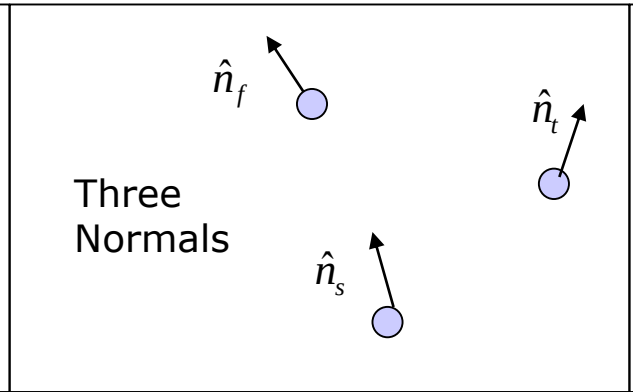
New uVertex class for Texture Mapping (UV mapping)

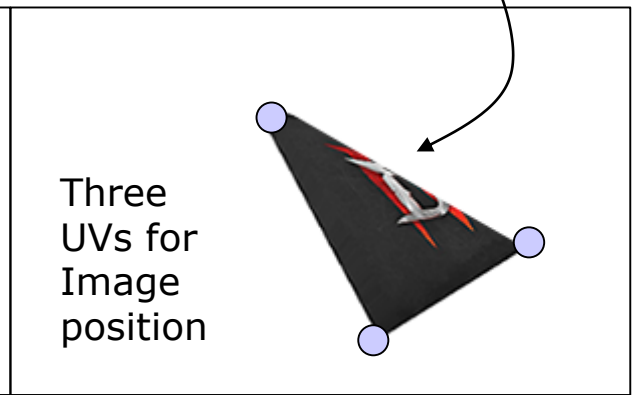
- uVertex in lib2

```
// Texture Mapping
class uVertex // Position(3), normal(3), UV(2)
{
public:
        uVertex(){}
public:
        uVector v;
        uVector n;
        struct
        {
            float u,v;
        } tx;
};
```









uTexture for Image Loading

```
uTexture::uTexture()
{
    tid= 0;
}

BOOL uTexture::Load(char *p)
{
    // load image
    BOOL bRet = img.Load(p);
    int w,h;
    if (bRet==FALSE)    return FALSE;
    img.BGR2RGB();
    BYTE *pa = img.GetBuffer();

    // create texture buffer
    glGenTextures(1, &tid);
    glBindTexture(GL_TEXTURE_2D,tid);

    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_LINEAR);

    // 32 bit image
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,img.w,img.h, 0, GL_RGBA, GL_UNSIGNED_BYTE,img.GetBuffer());

    glBindTexture(GL_TEXTURE_2D,0);

    return bRet;
}
```

Image loading and buffering

```
class uTexture
{
public:
        uTexture();
public:
        BOOL    Load(char *);
public:
        GLuint  tid;
        CImagew img;
};
```

OpenGL Texture handle

uTexture for Image Loading

```
uTexture::uTexture()
{
    tid= 0;
}

BOOL uTexture::Load(char *p)
{
    // load image
    BOOL bRet = img.Load(p);
    int w,h;
    if (bRet==FALSE)    return FALSE;
    img.BGR2RGB();
    BYTE *pa = img.GetBuffer();

    // create texture buffer
    glGenTextures(1, &tid);
    glBindTexture(GL_TEXTURE_2D,tid);

    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_LINEAR);

    // 32 bit image
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,img.w,img.h, 0, GL_RGBA, GL_UNSIGNED_BYTE,img.GetBuffer());

    glBindTexture(GL_TEXTURE_2D,0);

    return bRet;
}
```

```
class uTexture
{
public:
        uTexture();
public:
        BOOL    Load(char *);
public:
        GLuint  tid;
        CImagew img;
};
```

Generate texture handle

uTexture for Image Loading

```
uTexture::uTexture()
{
    tid= 0;
}

BOOL uTexture::Load(char *p)
{
    // load image
    BOOL bRet = img.Load(p);
    int w,h;
    if (bRet==FALSE)    return FALSE;
    img.BGR2RGB();
    BYTE *pa = img.GetBuffer();

    // create texture buffer
    glGenTextures(1, &tid);
    glBindTexture(GL_TEXTURE_2D,tid);

    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_LINEAR);

    // 32 bit image
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,img.w,img.h, 0, GL_RGBA, GL_UNSIGNED_BYTE,img.GetBuffer());

    glBindTexture(GL_TEXTURE_2D,0);

    return bRet;
}
```

```
class uTexture
{
public:
        uTexture();
public:
        BOOL    Load(char *);
public:
        GLuint  tid;
        CImagew img;
};
```

Texture setting

uTexture for Image Loading

```
uTexture::uTexture()
{
    tid= 0;
}

BOOL uTexture::Load(char *p)
{
    // load image
    BOOL bRet = img.Load(p);
    int w,h;
    if (bRet==FALSE)    return FALSE;
    img.BGR2RGB();
    BYTE *pa = img.GetBuffer();

    // create texture buffer
    glGenTextures(1, &tid);
    glBindTexture(GL_TEXTURE_2D,tid);

    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_LINEAR);

    // 32 bit image
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,img.w,img.h, 0, GL_RGBA, GL_UNSIGNED_BYTE,img.GetBuffer());

    glBindTexture(GL_TEXTURE_2D,0);

    return bRet;
}
```

```
class uTexture
{
public:
        uTexture();
public:
        BOOL    Load(char *);
public:
        GLuint  tid;
        CImagew img;
};
```

Copy Image buffer into
GPU by handle, tid.

uTexture for Image Loading

```
uTexture::uTexture()
{
    tid= 0;
}

BOOL uTexture::Load(char *p)
{
    // load image
    BOOL bRet = img.Load(p);
    int w,h;
    if (bRet==FALSE)    return FALSE;
    img.BGR2RGB();
    BYTE *pa = img.GetBuffer();

    // create texture buffer
    glGenTextures(1, &tid);
    glBindTexture(GL_TEXTURE_2D,tid);

    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_LINEAR);

    // 32 bit image
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,img.w,img.h, 0, GL_RGBA, GL_UNSIGNED_BYTE,img.GetBuffer());

    glBindTexture(GL_TEXTURE_2D,0);

    return bRet;
}
```

Close Texture handle

```
class uTexture
{
public:
        uTexture();
public:
        BOOL    Load(char *);
public:
        GLuint  tid;
        CImagew img;
};
```

uWnd and uObj for Texture Mapping

```
void uWnd::Loading()
{
    SetBK(RGB(255,255,255));
    m_cam.T = m_cam.T.Trans(0,0,-10.5);

    m_sh.Load("PhongTex.vsh","PhongTex.fsh");
    m_tx.Load("d4.png");
```

Load Image, d4.png

```
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);

    pVer[0].tx.u    = 0;    pVer[0].tx.v    = 1;
    pVer[1].tx.u    = 1;    pVer[1].tx.v    = 1;
    pVer[2].tx.u    = 1;    pVer[2].tx.v    = 0;
    pVer[3].tx.u    = 0;    pVer[3].tx.v    = 0;
    pVer[4].tx.u    = 0;    pVer[4].tx.v    = 1;
    pVer[5].tx.u    = 1;    pVer[5].tx.v    = 1;
    pVer[6].tx.u    = 1;    pVer[6].tx.v    = 0;
    pVer[7].tx.u    = 0;    pVer[7].tx.v    = 0;
```

- UV value for MakeBox and MakeCyl

VBO for Texture Mapping

```
// Texture Mapping
class uVertex // Position(3), normal(3), UV(2)
{
public:
        uVertex(){}
public:
        uVector v;
        uVector n;
        struct
        {
            float u,v;
        } tx;
};
```

Position: (x,y,z) for 12 bytes

Normal: (x,y,z) for 12 bytes

UV: (u,v) for 8 bytes

0th Three elements
Position (x,y,z)

```
// call vertex buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo);
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
glEnableVertexAttribArray(2);
glVertexAttribPointer(2,2, GL_FLOAT,FALSE,sizeof(uVertex),(void*)24);    // texture uv
```


VBO for Texture Mapping

```
// Texture Mapping
class uVertex // Position(3), normal(3), UV(2)
{
public:
        uVertex(){}
public:
        uVector v;
        uVector n;
        struct
        {
            float u,v;
        } tx;
};
```

Position: (x,y,z) for 12 bytes

Normal: (x,y,z) for 12 bytes

UV: (u,v) for 8 bytes

1st Three elements Normal (x,y,z)

After 12 bytes from Positon x,y,z

```
// 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
glEnableVertexAttribArray(2);
glVertexAttribPointer(2,2, GL_FLOAT,FALSE,sizeof(uVertex),(void*)24);    // texture uv
```


VBO for Texture Mapping

```
// Texture Mapping
class uVertex // Position(3), normal(3), UV(2)
{
public:
        uVertex(){}
public:
        uVector v;
        uVector n;
        struct
        {
            float u,v;
        } tx;
};
```

Position: (x,y,z) for 12 bytes

Normal: (x,y,z) for 12 bytes

UV: (u,v) for 8 bytes

2nd TWO elements
Textures (u,v)

After 24 bytes
Position(x,y,z)
+
Normal(x,y,z)

```
// 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
glEnableVertexAttribArray(2);
glVertexAttribPointer(2,2, GL_FLOAT,FALSE,sizeof(uVertex),(void*)24);    // texture uv
```


uObj::Draw() for Drawing Texture

```
// call texture
if (pCurrentTexture)
{
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D,pCurrentTexture->tid);
}
```

Bind texture handle

```
class uTexture
{
public:
        uTexture();
public:
        BOOL    Load(char *);
public:
        GLuint  tid;
        CImagew img;
};
```

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

```
if (pCurrentTexture)
{
    glDisable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D,0);
}
```

Unbind texture handle

UV value is defined as Normalized Image Position

- Image pixel coordinate

(x,y)=(0,0)

(511,511)

- Normalized Position by U and V

(u,v)=(0,0)

(1,0)

(0,1)

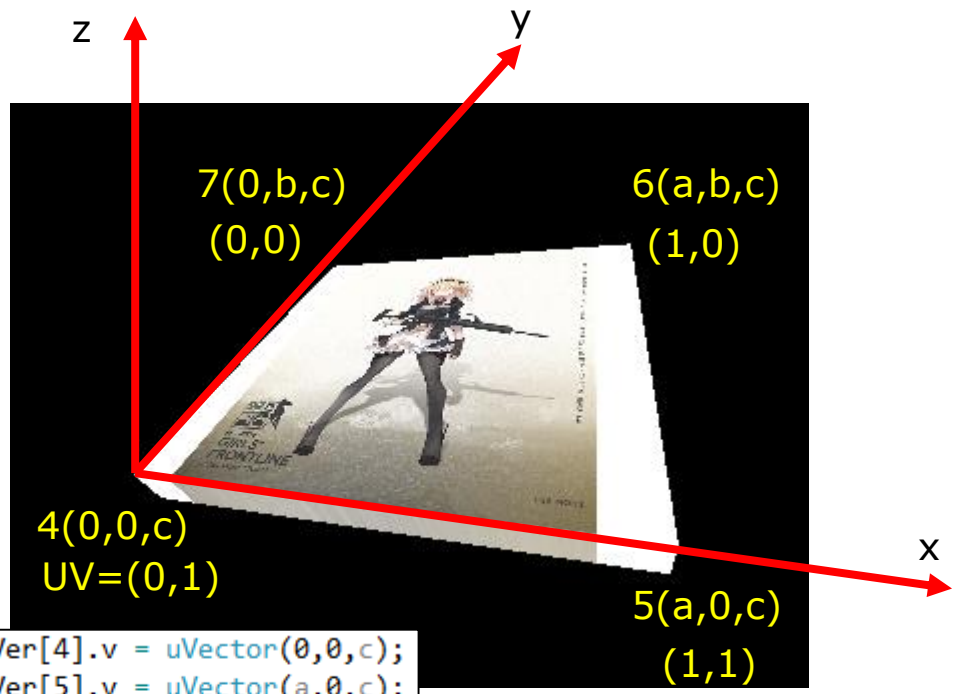
(1,1)

Ex) uGL-20-TextureMapping

- Original Image

Width=1024
Height=1024

MakeBox(10,10,1)

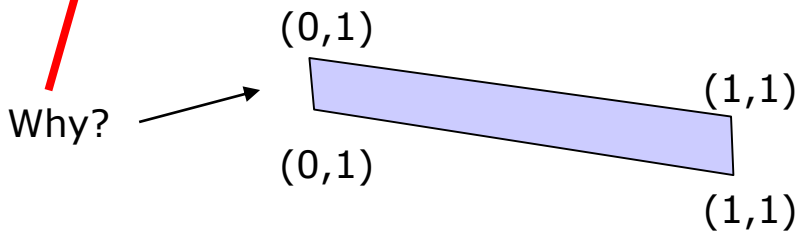



```
pVer[4].v = uVector(0,0,c);
pVer[5].v = uVector(a,0,c);
pVer[6].v = uVector(a,b,c);
pVer[7].v = uVector(0,b,c);
```

```
pVer[4].tx.u    = 0;    pVer[4].tx.v    = 1;
pVer[5].tx.u    = 1;    pVer[5].tx.v    = 1;
pVer[6].tx.u    = 1;    pVer[6].tx.v    = 0;
pVer[7].tx.u    = 0;    pVer[7].tx.v    = 0;
```


Ex) uGL-20-TextureMapping

- It looks somewhat Unusual. Why?



UV Tricks
ex) uGL-20-TextureMapping2

Original

U= 1 at right

```
pVer[4].tx.u    = 0;    pVer[4].tx.v    = 1;
pVer[5].tx.u    = 1;    pVer[5].tx.v    = 1;
pVer[6].tx.u    = 1;    pVer[6].tx.v    = 0;
pVer[7].tx.u    = 0;    pVer[7].tx.v    = 0;
```

U=0.5 at right

```
pVer[4].tx.u    = 0;    pVer[4].tx.v    = 1;
pVer[5].tx.u    = 0.5;  pVer[5].tx.v    = 1;
pVer[6].tx.u    = 0.5;  pVer[6].tx.v    = 0;
pVer[7].tx.u    = 0;    pVer[7].tx.v    = 0;
```


Features of Texture Mapping

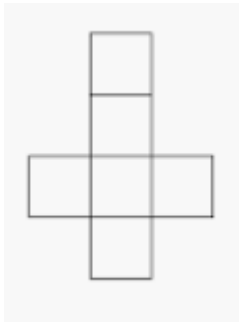
- Images are wrapped and skewed by UV map
  - Use uGL-20-TextureMapping, testG36-big.exe

Original

Skewed examples

Camera walking is based on Perspective Projection, Which is **Non Euclidean Transform**.

Ex) UV setting (24 vertices)
uGL-21-TX-box



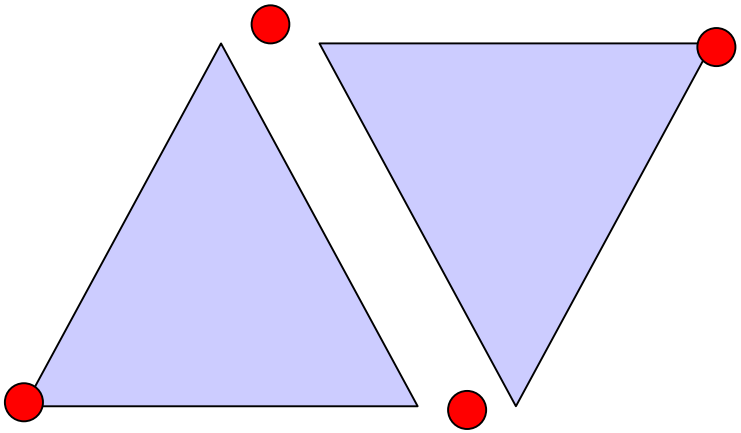

```
// back
pVer[12].v = uVector(0,b,0);    pVer
pVer[13].v = uVector(a,b,0);    pVer
pVer[14].v = uVector(a,b,c);    pVer
pVer[15].v = uVector(0,b,c);    pVer
pVer[12].tx.u = 0;  pVer[12].tx.v =
pVer[13].tx.u = 1;  pVer[13].tx.v =
pVer[14].tx.u = 1;  pVer[14].tx.v =
pVer[15].tx.u = 0;  pVer[15].tx.v =
pPoly[6].Set(12,14,13);
pPoly[7].Set(12,15,14);

// left
pVer[16].v = uVector(0,b,0);    pVer
pVer[17].v = uVector(0,0,0);    pVer
pVer[18].v = uVector(0,0,c);    pVer
```

Special Point of OpenGL Texture

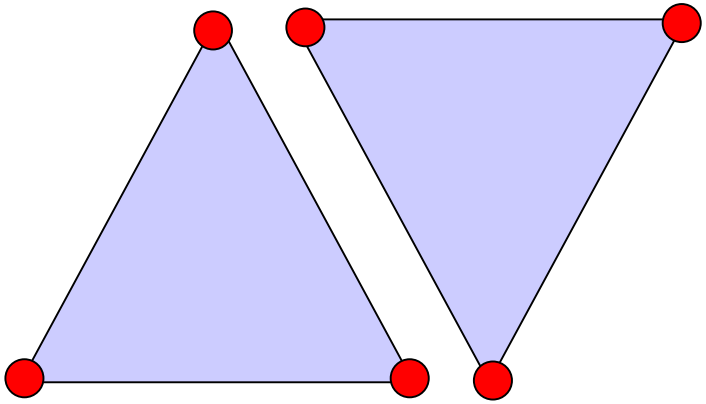
- Keep in your mind.
- Every Dimension of OpenGL Texture MUST be power of 2.
  - D4.png, width and height =512=2^9
  - G36.png, width and height = 1024 = 2^10
- Texture with width = 512 and height=1024 is also Good.

Meaning of that Vertices in a Polygon are NOT Duplicated

- Each Polygon has its own vertices.
  - UV mappings and Normal vectors are not used for other polygon
- Important features for Shading and UV Mapping
  - Flat and Gouraud shadings require Independent Normal Vectors
  - UV mapping also requires their own vertices for polyogns



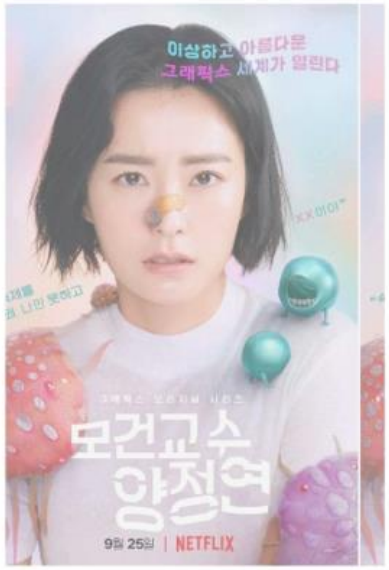
Duplicated vertex



NON-Duplicated vertex

Meaning of that
Vertices in a Polygon are NOT Duplicated
ex) uGL-47-Morgan-Prof 1 and 2

U = U +0.01



3 Dynamic Drawing (Dynamic Vertex in OpenGL)

OpenGL has Two Types for Object Building in GPU

- Static Mode
  - Data with vertex and element(face) are INVARIANT

```
glBindBuffer(GL_ARRAY_BUFFER, vs);
glBufferData(GL_ARRAY_BUFFER, sizeof(uVertex)* nVer, pVer, GL_STATIC_DRAW);
```

- Dynamic Mode
  - Data with vertex and element are Varying.

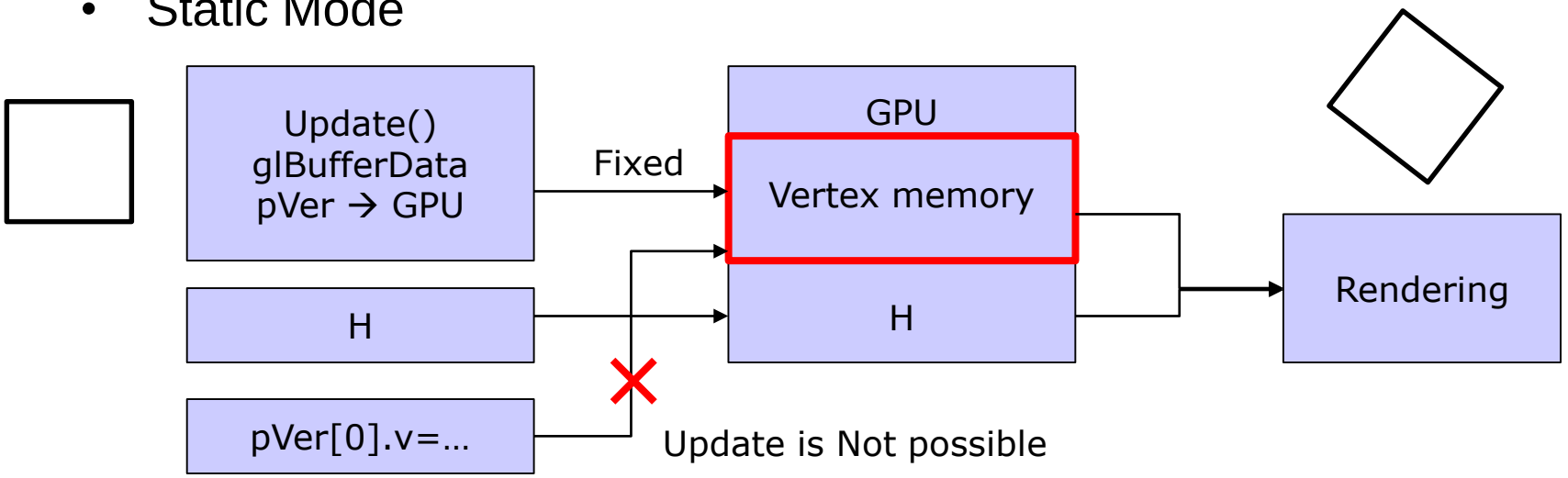
```
glBindBuffer(GL_ARRAY_BUFFER, vs);
if (bStatic)    glBufferData(GL_ARRAY_BUFFER, sizeof(uVertex)* nVer, pVer, GL_STATIC_DRAW);
else            glBufferData(GL_ARRAY_BUFFER, sizeof(uVertex)* nVer, pVer, GL_DYNAMIC_DRAW);
```

  - Update again with modified vertex (ReUpdate())

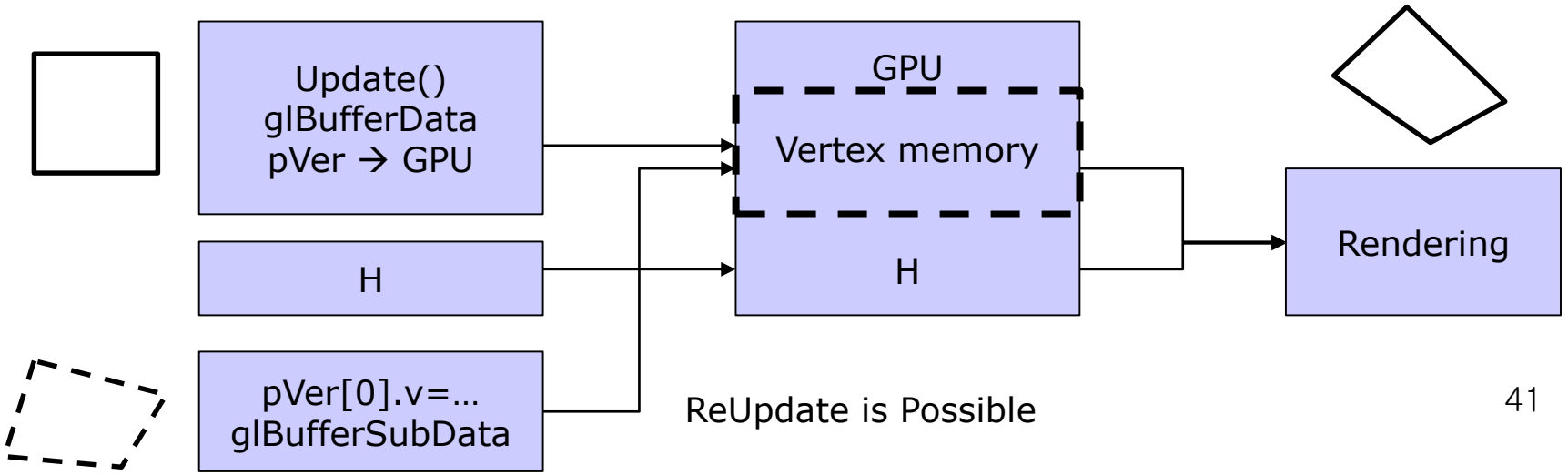
```
void uObj::ReUpdate()
{
    glBindBuffer(GL_ARRAY_BUFFER, vs);
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(uVertex)* nVer, pVer);
}
```


Concept of Dynamic Vertex

- Static Mode



- Dynamic Mode



Ex) uGL-22-TX-Dynamic

- Use wallpaper.png(32bit)

- What's wrong? **Normal vector must be changed**

Change of Normal Vector in Dynamic Mode