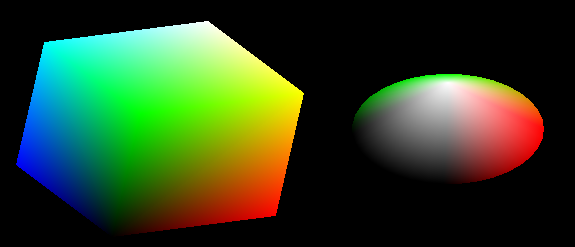
1. OpenGL VBO

OpenGL 1.0에서 삼각형을 그리는데는 아래와 같은 코드가 사용 되었습니다.  
  
glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
glLoadIdentity();  
glBegin(GL\_TRIANGLES);  
glVertex2f(-1.0f, -1.0f);  
glVertex2f(1.0f, -1.0f);  
glVertex2f(0.0f, 1.0f);  
glEnd();  
  
OpenGL 3.0이후에서도 여전히 위의 코드로 삼각형을 그릴수 있지만 아래와 같이 VBO를 사용하도록 권고하고 있습니다. glBegin()의 문제점은 병렬 처리가 거의 불가능 하다고합니다.  
  
static const GLfloat g\_vertex\_buffer\_data[] = {  
-1.0f, -1.0f, 0.0f,  
1.0f, -1.0f, 0.0f,  
0.0f, 1.0f, 0.0f,  
};  
glEnableClientState( GL\_VERTEX\_ARRAY );  
glVertexPointer( 3, GL\_FLOAT, 0, &g\_vertex\_buffer\_data[0] );  
glDrawArrays(GL\_TRIANGLES, 0, 3); // 버텍스 0에서 시작해서; 총 3개의 버텍스로 -> 하나의 삼각형  
update();  
이코드는 glBegin()을 사용하지 않지만 매번 GPU Memory로 Vertex info를 전달합니다.  
  
glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
glLoadIdentity();  
glBegin(GL\_TRIANGLES);  
glVertex2f(-1.0f, -1.0f);  
glVertex2f(1.0f, -1.0f);  
glVertex2f(0.0f, 1.0f);  
glEnd();  
OpenGL 3.0이후에서도 여전히 위의 코드로 삼각형을 그릴수 있지만 아래와 같이 VBO를 사용하도록 권고하고 있습니다. glBegin()의 문제점은 병렬 처리가 거의 불가능 하다고합니다.   
static const GLfloat g\_vertex\_buffer\_data[] = {  
-1.0f, -1.0f, 0.0f,  
1.0f, -1.0f, 0.0f,  
0.0f, 1.0f, 0.0f,  
};  
  
glEnableClientState( GL\_VERTEX\_ARRAY );  
glVertexPointer( 3, GL\_FLOAT, 0, &g\_vertex\_buffer\_data[0] );  
glDrawArrays(GL\_TRIANGLES, 0, 3); // 버텍스 0에서 시작해서; 총 3개의 버텍스로 -> 하나의 삼각형  
update();  
이코드는 glBegin()을 사용하지 않지만 매번 GPU Memory로 Vertex info를 전달합니다.  
  
static const GLfloat g\_vertex\_buffer\_data[] = {   
-1.0f, -1.0f, 0.0f,  
1.0f, -1.0f, 0.0f,  
0.0f, 1.0f, 0.0f,  
};  
  
glewInit();  
// 초기화  
GLuint vertexbuffer;  
glGenBuffers(1, &vertexbuffer);  
glBindBuffer(GL\_ARRAY\_BUFFER, vertexbuffer);  
glBufferData(GL\_ARRAY\_BUFFER, sizeof(g\_vertex\_buffer\_data), g\_vertex\_buffer\_data, GL\_STATIC\_DRAW);  
  
// 렌더링  
// 버퍼의 첫번째 속성값(attribute) : 버텍스들  
glEnableVertexAttribArray(0);  
glBindBuffer(GL\_ARRAY\_BUFFER, vertexbuffer);  
glVertexAttribPointer(  
0,        // 0번째 속성(attribute). 0 이 될 특별한 이유는 없지만, 쉐이더의 레이아웃(layout)와 반드시 맞추어야 합니다.  
3,        // 크기(size)  
GL\_FLOAT,       // 타입(type)  
GL\_FALSE,        // 정규화(normalized)?  
0,                  // 다음 요소 까지 간격(stride)  
(void\*)0          // 배열 버퍼의 오프셋(offset; 옮기는 값)  
);  
// 삼각형 그리기!  
glDrawArrays(GL\_TRIANGLES, 0, 3); // 버텍스 0에서 시작해서; 총 3개의 버텍스로 -> 하나의 삼각형  
glDisableVertexAttribArray(0);  
[원문 : <http://www.opengl-tutorial.org/kr/beginners-tutorials/tutorial-2-the-first-triangle/>]  
  
위의 코드는 VBO를 사용하고 있고 정점 배열이 미리 준비되어 있는 상태에서 glDrawArrays를 호출하여 여러 원시 기하 도형을 렌더링합니다.  
VBO 배열에는 점점, 법선,색상,Texture등이 있습니다.  
위는 VBO만 사용했지만 VAO, SHADER를 사용하여 반복적으로 Drawing하는부분에서 성능의 향상을 가져올 수 있으며 CPU가 할일을 GPU쪽에서 처리함으로 인해 반복적인 Drawing 작업은 GPU에서처리함으로 CPU는 다른 작업에 시간을 할애할수 있어 많은 장점을 얻을수 있습니다.  
  
  
다음은 Indices를 사용하는 glDrawElements()를 사용하여 렌더링을 해보겠습니다.  
  
static const GLfloat g\_vertex\_buffer\_data[] = {   
0.0f, 0.0f, 0.0f, // Vertex 0  
-0.5f, 0.5f, 0.0f,// Vertex 1   
0.5f, 0.5f, 0.0f, // Vertex 2  
1.0f, 0.0f, 0.0f, // Vertex 3  
0.5f, -0.5f, 0.0f,// Vertex 4  
-0.5f, -0.5f, 0.0f,// Vertex 5  
-1.0f, 0.0f, 0.0f,// Vertex 6  
};  
static const GLubyte indices[] = {  
0,2,1, // 삼각형 1  
0,3,2,// 삼각형 2  
0,4,3,// 삼각형 3  
0,5,4,// 삼각형 4  
0,6,5,// 삼각형 5  
0,1,6,// 삼각형 6  
};  
  
  
glewInit();  
// 초기화  
GLuint vertexbuffer;  
glGenBuffers(1, &vertexbuffer);  
glBindBuffer(GL\_ARRAY\_BUFFER, vertexbuffer);  
glBufferData(GL\_ARRAY\_BUFFER, sizeof(g\_vertex\_buffer\_data), g\_vertex\_buffer\_data, GL\_STATIC\_DRAW);  
  
// 렌더링  
// 버퍼의 첫번째 속성값(attribute) : 버텍스들  
glEnableVertexAttribArray(0);  
glBindBuffer(GL\_ARRAY\_BUFFER, vertexbuffer);  
glVertexAttribPointer(0,  3, GL\_FLOAT, GL\_FALSE, 0, (void\*)0);  
// 삼각형 그리기!  
glDrawElements(GL\_TRIANGLE\_STRIP, sizeof(indices)/sizeof(GLubyte), GL\_UNSIGNED\_BYTE, indices);  
glDisableVertexAttribArray(0);  
  
참조 : <http://gogorchg.tistory.com/entry/Android-Opengl-es-20-glDrawArrays->와-glDrawElements-사용법  
  
Indices[] 테이블이 추가되어 반복된 vertex가 많아질때는 매우 사용하기가 편리합니다.  
<http://soen.kr/lecture/library/opengl/opengl-7.htm> 사이트에 ‘버텍스 배열’에 관한 더 많은 정보를 얻을수 있으라 생각합니다.  
  
위에 언급된 Source Code들은 아래 사이트에 가면 볼수 있습니다.  
<https://github.com/JonglockYoon/OpenGL/tree/master/openglvbovao>

### 2. OpenGL VAO

Vertex Array Object(VAO)는 정점 속성이 Vertex Buffer Object(VBO)에 저장되는 방법을 설명하는 객체입니다. 즉, VAO는 정점 데이터를 저장하는 실제 객체가 아니라 정점 데이터의 디스크립터입니다.  
앞의 VBO Example에 VAO를 추가하는 작업을 하도록 하겠습니다.  
  
  
static const GLfloat g\_vertex\_buffer\_data[] = {   
0.0f, 0.0f, 0.0f, // Vertex 0  
-0.5f, 0.5f, 0.0f,// Vertex 1   
0.5f, 0.5f, 0.0f, // Vertex 2  
1.0f, 0.0f, 0.0f, // Vertex 3  
0.5f, -0.5f, 0.0f,// Vertex 4  
-0.5f, -0.5f, 0.0f,// Vertex 5  
-1.0f, 0.0f, 0.0f,// Vertex 6  
};  
static const GLushort indices[] = {  
0,2,1, // 삼각형 1  
0,3,2,// 삼각형 2  
0,4,3,// 삼각형 3  
0,5,4,// 삼각형 4  
0,6,5,// 삼각형 5  
0,1,6,// 삼각형 6  
};  
  
// VAO 초기화  
GLuint VertexArrayID;  
glGenVertexArrays(1, &VertexArrayID);  
glBindVertexArray(VertexArrayID);  
  
// VBO 초기화  
GLuint vertexbuffer;  
glGenBuffers(1, &vertexbuffer);  
glBindBuffer(GL\_ARRAY\_BUFFER, vertexbuffer);  
glBufferData(GL\_ARRAY\_BUFFER, sizeof(g\_vertex\_buffer\_data), g\_vertex\_buffer\_data, GL\_STATIC\_DRAW);  
  
// 전달Data 명시  
glEnableVertexAttribArray(0);  
//glBindBuffer(GL\_ARRAY\_BUFFER, vertexbuffer);  
glVertexAttribPointer(0,  3, GL\_FLOAT, GL\_FALSE, 0, (void\*)0);  
glBindVertexArray(0);  
  
// 렌더링  
glBindVertexArray(VertexArrayID);  
glDrawElements(GL\_TRIANGLE\_STRIP, sizeof(indices)/sizeof(GLushort), GL\_UNSIGNED\_SHORT, indices);  
glBindVertexArray(0);  
  
육면체 Cube에 Color를 입히는 VAO Example은 아래 사이트에를 참조하면 됩니다.  
<http://www.informit.com/articles/article.aspx?p=1377833&seqNum=8> 

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221336605394&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

위에 언급된 Source Code들은 아래 사이트에 가면 볼수 있습니다.  
Windows에서 실행할 때에는 openglvbovao.exe 파일위치에   
freeglut.dll ,glew32.dll ,glfw3.dll 파일들이 같이 위치해 있어야합니다.  
<https://github.com/JonglockYoon/OpenGL/tree/master/openglvbovao>

### 3. OpenGL Draw Cube with Texture

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221361840907&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

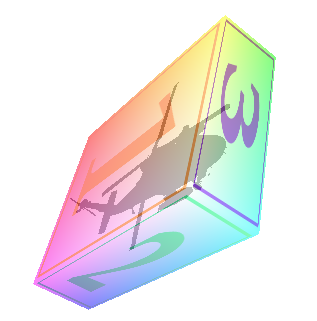
앞장에서 사용한 glDrawElements()와 이장에서는 glBindTexture()를 이용하여 Cube에 Texture를 입히는 과정을 알아보겠습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221361840907&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

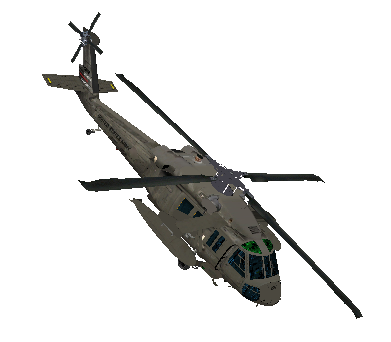
     // cube ///////////////////////////////////////////////////////////////////////  
    // vertex array for glDrawElements() and glDrawRangeElement() =================  
    GLfloat vertices[] = { 1, 1, 1,  -1, 1, 1,  -1,-1, 1,   1,-1, 1,   // v0,v1,v2,v3 (front)  
                           1, 1, 1,   1,-1, 1,   1,-1,-1,   1, 1,-1,   // v0,v3,v4,v5 (right)  
                           1, 1, 1,   1, 1,-1,  -1, 1,-1,  -1, 1, 1,   // v0,v5,v6,v1 (top)  
                          -1, 1, 1,  -1, 1,-1,  -1,-1,-1,  -1,-1, 1,   // v1,v6,v7,v2 (left)  
                          -1,-1,-1,   1,-1,-1,   1,-1, 1,  -1,-1, 1,   // v7,v4,v3,v2 (bottom)  
                           1,-1,-1,  -1,-1,-1,  -1, 1,-1,   1, 1,-1 }; // v4,v7,v6,v5 (back)  
    // normal array  
    GLfloat normals[]  = { 0, 0, 1,   0, 0, 1,   0, 0, 1,   0, 0, 1,   // v0,v1,v2,v3 (front)  
                           1, 0, 0,   1, 0, 0,   1, 0, 0,   1, 0, 0,   // v0,v3,v4,v5 (right)  
                           0, 1, 0,   0, 1, 0,   0, 1, 0,   0, 1, 0,   // v0,v5,v6,v1 (top)  
                          -1, 0, 0,  -1, 0, 0,  -1, 0, 0,  -1, 0, 0,   // v1,v6,v7,v2 (left)  
                           0,-1, 0,   0,-1, 0,   0,-1, 0,   0,-1, 0,   // v7,v4,v3,v2 (bottom)  
                           0, 0,-1,   0, 0,-1,   0, 0,-1,   0, 0,-1 }; // v4,v7,v6,v5 (back)  
   
[#define](https://blog.naver.com/PostListByTagName.nhn?blogId=jerry1455&encodedTagName=define) B 0.9 // Alpha Blend  
    // color array  
    GLfloat colors[]   = { 1, 1, 1,B,   1, 1, 0,B,   1, 0, 0,B,   1, 0, 1,B,   // v0,v1,v2,v3 (front)  
                           1, 1, 1,B,   1, 0, 1,B,   0, 0, 1,B,   0, 1, 1,B,   // v0,v3,v4,v5 (right)  
                           1, 1, 1,B,   0, 1, 1,B,   0, 1, 0,B,   1, 1, 0,B,   // v0,v5,v6,v1 (top)  
                           1, 1, 0,B,   0, 1, 0,B,   0, 0, 0,B,   1, 0, 0,B,   // v1,v6,v7,v2 (left)  
                           0, 0, 0,B,   0, 0, 1,B,   1, 0, 1,B,   1, 0, 0,B,   // v7,v4,v3,v2 (bottom)  
                           0, 0, 1,B,   0, 0, 0,B,   0, 1, 0,B,   0, 1, 1,B }; // v4,v7,v6,v5 (back)  
   
    // texCoord array  
    GLfloat texCoords[]  = { 1, 0,   0, 0,   0, 1,   1, 1,             // v0,v1,v2,v3 (front)  
                             0, 0,   0, 1,   1, 1,   1, 0,             // v0,v3,v4,v5 (right)  
                             1, 1,   1, 0,   0, 0,   0, 1,             // v0,v5,v6,v1 (top)  
                             1, 0,   0, 0,   0, 1,   1, 1,             // v1,v6,v7,v2 (left)  
                             0, 1,   1, 1,   1, 0,   0, 0,             // v7,v4,v3,v2 (bottom)  
                             0, 1,   1, 1,   1, 0,   0, 0, };          // v4,v7,v6,v5 (back)  
   
    // index array of vertex array for glDrawElements() & glDrawRangeElement()  
    GLubyte indices[]  = { 0, 1, 2,   2, 3, 0,      // front  
                           4, 5, 6,   6, 7, 4,      // right  
                           8, 9,10,  10,11, 8,      // top  
                          12,13,14,  14,15,12,      // left  
                          16,17,18,  18,19,16,      // bottom  
                          20,21,22,  22,23,20 };    // back  
   
**vertices**배열은 Vertex 정점 정보이며 **normals**배열은 법선 정보 배열입니다.  
이어서 **colors**, **texCoords**, **indices**배열을 선언합니다.  
OpenGL의 아키텍처는 클라이언트-서버 모델을 기반으로 합니다. OpenGL API를 사용하도록 작성된 응용 프로그램은 "클라이언트"이며 CPU에서 실행됩니다. OpenGL 그래픽 엔진 (GLSL 쉐이더 프로그램 포함)의 구현은 "서버"이며 GPU에서 실행됩니다. VBO 버퍼는 GPU에 할당되고 CPU 프로그램에 의해 채워집니다. VBO는 그래픽카드의 메모리를 이용하는 방식입니다.  
glDrawElements()를 이용하여 매번마다 Vertex 정점정보를 GPU로 전달하지않고 indices 정보만으로그려지게 됩니다.  
glEnableClientState(), glDisableClientState()  
위의 두 함수는 클라이언트 측 기능을 사용가능 또는 사용하지 않음으로 설정하는 함수입니다.  
  
glPushMatrix();  
glEnable(GL\_TEXTURE\_2D);  
   
// enable and specify pointers to vertex arrays  
**glEnableClientState**(GL\_NORMAL\_ARRAY);  
**glEnableClientState**(GL\_COLOR\_ARRAY);  
**glEnableClientState**(GL\_TEXTURE\_COORD\_ARRAY);  
**glEnableClientState**(GL\_VERTEX\_ARRAY);  
   
glEnable( GL\_BLEND );  
glBlendFunc( GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA );  
   
glNormalPointer(GL\_FLOAT, 0, normals);  
glColorPointer(4, GL\_FLOAT, 0, colors);  
glTexCoordPointer(2, GL\_FLOAT, 0, texCoords);  
glVertexPointer(3, GL\_FLOAT, 0, vertices);  
   
glBindTexture(GL\_TEXTURE\_2D, id\_ogl\_texture[0]);  
glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_BYTE, &indices[0]);  
glBindTexture(GL\_TEXTURE\_2D, id\_ogl\_texture[1]);  
glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_BYTE, &indices[6\*1]);  
glBindTexture(GL\_TEXTURE\_2D, id\_ogl\_texture[2]);  
glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_BYTE, &indices[6\*2]);  
glBindTexture(GL\_TEXTURE\_2D, id\_ogl\_texture[3]);  
glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_BYTE, &indices[6\*3]);  
glBindTexture(GL\_TEXTURE\_2D, id\_ogl\_texture[4]);  
glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_BYTE, &indices[6\*4]);  
glBindTexture(GL\_TEXTURE\_2D, id\_ogl\_texture[5]);  
glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_BYTE, &indices[6\*5]);  
   
**glDisableClientState**(GL\_VERTEX\_ARRAY);  // disable vertex arrays  
**glDisableClientState**(GL\_TEXTURE\_COORD\_ARRAY);  
**glDisableClientState**(GL\_COLOR\_ARRAY);  
**glDisableClientState**(GL\_NORMAL\_ARRAY);  
   
glPopMatrix();  
update();  
   
Windows에서 실행 할 때에는 openglcube.exe 파일과 같은 폴더에 assimp-vc140-mt.dll, DevIL.dll, ILU.dll, ILUT.dll,jpeg62.dll, libpng16.dll, zlib1.dll 파일들이 같이 있어야 실행이 됩니다.  
Mouse Wheel : 확대 및 축소, Mouse PressMove : 회전 조작을 할수 있습니다.  
  
images 폴더 도 실행파일 또는 Makefile파일 위치에 있어야 Texture 이미지가 정상적으로 로드됩니다.  
   
전체 Source Code는 다음에서 볼 수 있습니다.  
<https://github.com/JonglockYoon/QT-OpenGL/tree/master/openglcube>

### 4. OpenGL Draw Obj with BBox Texure

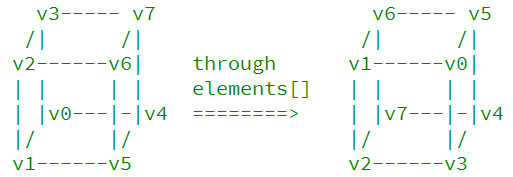
File -> readFile 에서uh60.obj를 로드합니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221363688067&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

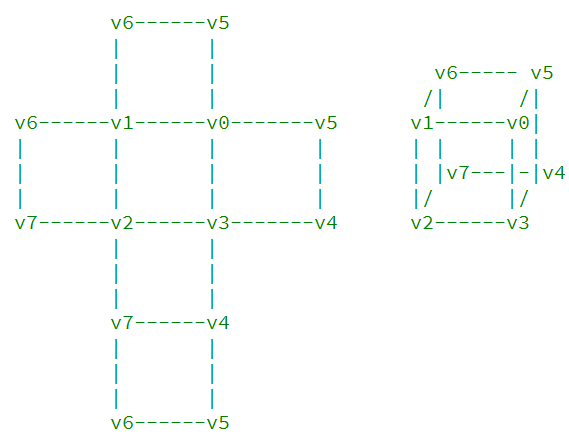
Box안에Blackhawk 헬리곱터가 있고,  Box에는 Texture 이미지가 그려져 있으며 각 점점마다 color가 설정되어 있네요.   
uh60.obj파일은uh60.mtl및 자체texture jpg파일들이 있습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221363688067&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

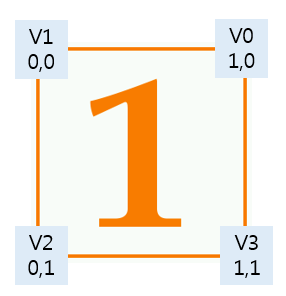
Obj관련 모든 정보를 로드하면 이와 같이 됩니다만 여기서는mertial 정보는 가져오지 않고 오직vertex정보만rendering하여 외곽Box를 그리고Box에Texture를 입혀봅니다.   
여기서 다루는model파일정보는 모두<https://github.com/assimp/assimp>사이트의 코드 및library를  이용합니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221363688067&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

ai에서 load 되는 model file의 외곽Box정보의face들을 현재 정의된 table로 변환하기위한 elements[]를이용해서 정점table 및 normal table을변환합니다.   
  
const GLubyte elements[]=   
{   
6,2,1,5,//front // ai의 6215의 vertex위치가 0123위치로 변하게됩니다.   
6,5,4,7,//right // 0345가 됩니다.   
6,7,3,2,//top // 0561   
2,3,0,1,//left   
0,4,5,1,//bottom   
4,0,3,7,//back   
};   
aiVector3D vertex[8];   
BBox\_GetVertices(pBBox,vertex);   
for(int i=0;i<24;i++){   
aiVector3D a=vertex[elements[i]];   
vertices[i\*3]=a.x;   
vertices[i\*3+1]=a.y;   
vertices[i\*3+2]=a.z;   
}   
int no[6]={2,0,1,0,1,2};   
for(int i=0;i<6;i++){   
for(int j=0;j<4;j++){   
normals[i\*12+j\*3+no[i]]=vertices[i\*12+j\*3+no[i]];   
}   
}   
  
위의 변환 과정을 거쳐obj modelvertex정보가 변하게되므로 아래의colors,texCoords,indices는 이전과 동일하게 사용하게됩니다.   
//이전 post의 내용과 동일합니다   
[#define](https://blog.naver.com/PostListByTagName.nhn?blogId=jerry1455&encodedTagName=define)B0.5   
GLfloat colors[]={1,1,1,B,1,1,0,B,1,0,0,B,1,0,1,B,//v0,v1,v2,v3(front)   
1,1,1,B,1,0,1,B,0,0,1,B,0,1,1,B,//v0,v3,v4,v5(right)   
1,1,1,B,0,1,1,B,0,1,0,B,1,1,0,B,//v0,v5,v6,v1(top)   
1,1,0,B,0,1,0,B,0,0,0,B,1,0,0,B,//v1,v6,v7,v2(left)   
0,0,0,B,0,0,1,B,1,0,1,B,1,0,0,B,//v7,v4,v3,v2(bottom)   
0,0,1,B,0,0,0,B,0,1,0,B,0,1,1,B};//v4,v7,v6,v5(back)   
  
//texCoordarray   
GLfloat texCoords[]={1,0,0,0,0,1,1,1,//v0,v1,v2,v3(front)   
0,0,0,1,1,1,1,0,//v0,v3,v4,v5(right)   
1,1,1,0,0,0,0,1,//v0,v5,v6,v1(top)   
1,0,0,0,0,1,1,1,//v1,v6,v7,v2(left)   
0,1,1,1,1,0,0,0,//v7,v4,v3,v2(bottom)   
0,1,1,1,1,0,0,0,};//v4,v7,v6,v5(back)   
  
//index array of vertex array for glDrawElements() & glDrawRangeElement()   
GLubyte indices[]={0,1,2,2,3,0,//front   
4,5,6,6,7,4,//right   
8,9,10,10,11,8,//top   
12,13,14,14,15,12,//left   
16,17,18,18,19,16,//bottom   
20,21,22,22,23,20};//back   
  
여기까지 obj 파일의 외곽 Box정보를 Load하고 Texture를 입히는 과정을 보았습니다.   
처음 그림에서 보여진 Box안에 헬리곱터 그림이 보이죠? 이것은 Alpha Bland를 0.5로 지정했기 때문에 Box내부의 object가 보이는것입니다.   
  
사전에 ImportTextures()함수를 이용하여 id\_ogl\_texture에 Texture이미지를 Load해둡니다.   
  
texCoords[]는 그려질 이미지의 정점 정보입니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221363688067&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

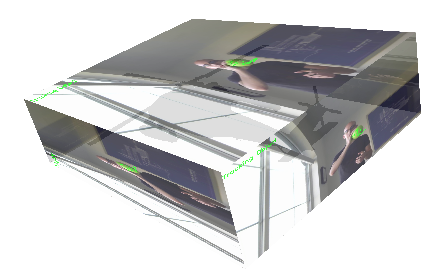
Front는“1,0,0,0,0,1,1,1” 으로 정의 되어 있는데 다음 그림으로 설명이 될 수 있습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221363688067&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

Box내부에 그려진 object가 display되는 과정은 다음과 같습니다.   
Assimp::ImporterClass 의 ReadFile 함수를 호출하여 aiScene\*에 Scene정보를 로드하고 CGLView::SetScene()으로 정보를 전달합니다.   
  
Draw\_Node()함수를 이용해서 최종 화면에Rendering된 obj파일이display되게 됩니다.   
  
전체 model파일 load과정은 assimp-4.1.0/tools/assimp\_qt\_viewer를 참조하면 좋을 것 같습니다.   
  
  
다음으로ImportTextures()함수는 무었을 하는지 살펴보겠습니다.   
DevIL 라이브러리를 사용합니다. DevIL은 OpenGL 스타일로 사용이 됩니다.   
  
ilInit();//InitializationofDevIL.   
glGenTextures(6,id\_ogl\_texture);//TextureIDgeneration.   
  
// 아래 코드들로 6개의 이미지를 Load합니다.   
ilGenImages(1,&id\_image);//Generate DevIL image ID.   
ilBindImage(id\_image);    
ilLoadImage(fileloc.toLocal8Bit());    
ilConvertImage(IL\_RGBA,IL\_UNSIGNED\_BYTE);   
glBindTexture(GL\_TEXTURE\_2D,id\_ogl\_texture[n]);   
glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);   
glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);   
glTexImage2D(GL\_TEXTURE\_2D,0,ilGetInteger(IL\_IMAGE\_BPP),ilGetInteger(IL\_IMAGE\_WIDTH),  
   ilGetInteger(IL\_IMAGE\_HEIGHT),0, ilGetInteger(IL\_IMAGE\_FORMAT),GL\_UNSIGNED\_BYTE,ilGetData());   
ilDeleteImages(1,&id\_image);   
  
Windows에서 실행할때에는 openglobj.exe 파일과 같은 폴더에 assimp-vc140-mt.dll, DevIL.dll, ILU.dll, ILUT.dll,jpeg62.dll, libpng16.dll, zlib1.dll 파일들이 같이 있어야 실행이 됩니다.   
Wheel : 확대 및 축소, Mouse PressMove : 회전 조작을 할수 있습니다.   
  
전체 Source Code는 다음에서 볼 수 있습니다.   
[https://github.com/JonglockYoon/QT-OpenGL/tree/master/openglobj](https://github.com/JonglockYoon/QT-OpenGL/tree/master/openglcube)

### 5. Video Draw on BBox Texure with Obj and OpenCV

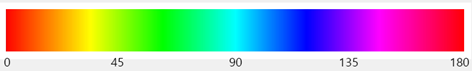
File -> readFile 에서 uh60.obj를 로드합니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221367511195&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

Obj 파일용도는4장과 동일하게BBox를 구하는데 사용됩니다.물론Vertex정보만으로Rendering해서model정보를 표현합니다.   
  
동영상은[https://github.com/jonathanfoster/ball-tracking사이트에 있는ball\_tracking\_example.mp4](https://github.com/jonathanfoster/ball-tracking%EC%82%AC%EC%9D%B4%ED%8A%B8%EC%97%90%20%EC%9E%88%EB%8A%94%20ball_tracking_example.mp4)을 이용합니다   
이Program에서 하고자 하는 목적은OpenCV로 동영상을Load하여Object를 추적하고 추적결과를 동영상에 표시하며,  OpenGL을 이용하여BBox에Texture로 올리는 것을 목표로 합니다.   
OpenCV와OpenGL의 기능을 연동해서 모델링한 결과의 각각 여섯면에OpenCV의 결과를OpenGL Texture기능으로 보여줄 수 있습니다.   
  
이것을 응용하여 어떤 물체를 카메라로Grab하여 분석하고 물체의Model File을 랜더링 하면서, 분석한 결과를 랜더링한 물체에Overlap 시켜 분석결과의 직관적인 표현이 가능합니다.   
  
물체를 모델링할 때 위 목적으로Texture를 같이 디자인 한다면BBox를 이용하지 않고Model file의Material정보와 결합한다면Model표면에 분석결과를 표현할 수도 있을 것 입니다.   
  
**1.mp4 file load**  
cv::VideoCapturecapture;   
capture.open("ball\_tracking\_example.mp4");   
  
**2. 분석**  
capture.read(cameraFeed);   
cv::cvtColor(cameraFeed,HSV,cv::COLOR\_BGR2HSV);   
cv::inRange(HSV,cv::Scalar(29,86,64),cv::Scalar(64,256,256),threshold); //Hue 26~64사이의 값을가진 물체를 찾습니다   
base.FilterLargeArea(&iplImage); //가장 큰blob만 남김.   
trackFilteredObject(x,y,threshold,cameraFeed); 

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221367511195&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

찾고자하 는object

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221367511195&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

hue color value

찾고자하는 object는 Hue value 29~64사이의 값을 가집니다.  
Saturation(채도)는 86 ~ 256, Value(명도)는 64~256 값의 object를 찾아서threshold 에넣어줍니다.  채도 와 명도는 아래 그림으로 설명될 수 있습니다.

[https://postfiles.pstatic.net/MjAxODA5MjlfMjk1/MDAxNTM4MTg0ODgwNDAz.trjBomyrkngtNxhoXK_mFPYigGh632YIk0rVyk990-cg.hQMKXkTBRUxJVR-ufCbIWJcdlTqrDc7IfQWtNq5er_kg.PNG.jerry1455/sv.png?type=w773](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221367511195&parentCategoryNo=&categoryNo=11&viewDate=&isShowPopularPosts=false&from=postList)

S & V

**3.Texture로 변환**  
cv::imencode(".bmp",cameraFeed,buf);// Mat format을BMP format으로 변환.   
ilLoadL(IL\_BMP,(constvoid\*)&buf[0],size);// DevIL Library로BMP load   
ilConvertImage(IL\_RGBA,IL\_UNSIGNED\_BYTE);// RGBA format으로 변경   
glTexImage2D(GL\_TEXTURE\_2D, …., ilGetData());// Image buffer를OpenGL로 전달.   
update(); // paintGL()실행.   
  
**4. Rendering**  
Draw\_Node(mScene->mRootNode); // model file rendering   
DoCubeDisplay(mScene\_BBox); // BBox에Texture표현.   
  
앞에서의Post들은 고정Texture이미지를 사용한 반면 여기서는 이미지 매Frame마다Texture를 갱신해 줌으로써Cube의 표면에 동영상을 표시할 수 있게 됩니다.   
  
**OpenCV에서 지원하는 이미지Format들은 다음과 같습니다.**  
  Windows bitmaps - \*.bmp, \*.dib (always supported)   
  JPEG files - \*.jpeg, \*.jpg, \*.jpe (see the Notes section)   
  JPEG 2000 files - \*.jp2 (see the Notes section)   
  Portable Network Graphics - \*.png (see the Notes section)   
  Portable image format - \*.pbm, \*.pgm, \*.ppm (always supported)   
  Sun rasters - \*.sr, \*.ras (always supported)   
  TIFF files - \*.tiff, \*.tif (see the Notes section)   
  
cv::imencode()함수는 다음과 같이 사용할수 있습니다.   
std::vector<uchar>buf;   
std::vector<int>params(2);   
params[0]=CV\_IMWRITE\_JPEG\_QUALITY;   
params[1]=80;   
cv::imencode(".jpg",cameraFeed,buf,params);   
  
 사용되는 parameter 값은 각 format에 따라 다음과 같이 사용됩니다.   
CV\_IMWRITE\_JPEG\_QUALITY: 0 to 100 (better quality) // default: 80% quality   
CV\_IMWRITE\_PNG\_COMPRESSION: 0 to 9 (smaller size) // default: maximum compression (9)   
CV\_IMWRITE\_PXM\_BINARY: 0 or 1 // PPM, PGM, or PBM format, default 1   
  
Full source는 다음 사이트에서 확인할 수 있습니다.  
[https://github.com/JonglockYoon/QT-OpenGL/tree/master/videocubeobj](https://github.com/JonglockYoon/QT-OpenGL/tree/master/videocubeobj" \t "_blank)