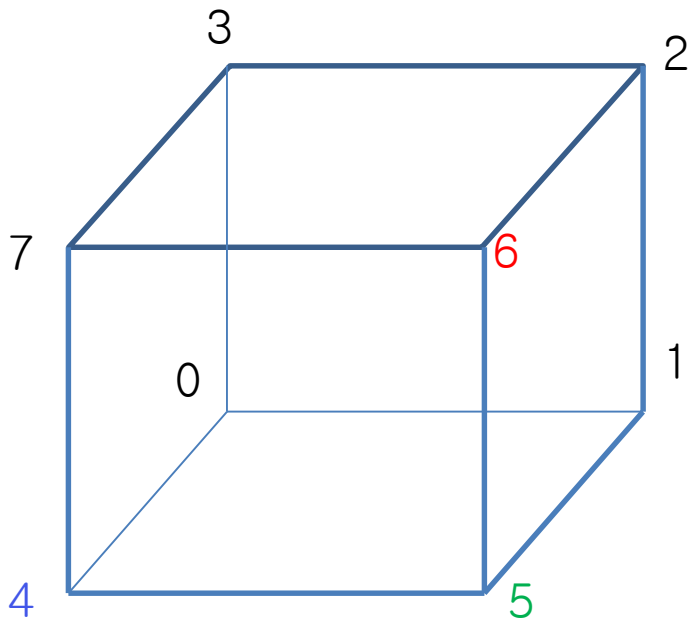
Data Transfer

Data Transfer Problem

- **Every frame may have to process millions of graphics data:**
  - Vertex coordinates, normals, colors, texture coordinates
  - Textures
  - OpenGL commands
- **Are there more efficient ways?**
  - Yes. OpenGL Extensions have been developed continuously for this purpose.
  - In this class, we study
    - **Display list**
    - **Per vertex data array**
    - **Abstract Buffer Objects**
      - **VBO**
      - **PBO**
      - **FBO**

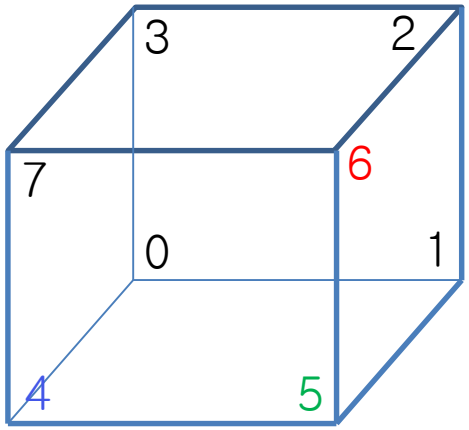
Our Example Cube



Drawing the Cube

- 24 vertex calls may be required.

```
float v[8][3] = {
            {-1.f,-1.f,-1.f}, { 1.f,-1.f,-1.f}, { 1.f, 1.f,-1.f}, {-1.f, 1.f,-1.f},   // back
            {-1.f,-1.f, 1.f}, { 1.f,-1.f, 1.f}, { 1.f, 1.f, 1.f}, {-1.f, 1.f, 1.f}    // front
};

float vv[24][3] = {
            {-1.f,-1.f,-1.f}, {-1.f, 1.f,-1.f}, { 1.f, 1.f,-1.f}, { 1.f,-1.f,-1.f},    // back
            {-1.f,-1.f, 1.f}, { 1.f,-1.f, 1.f}, { 1.f, 1.f, 1.f}, {-1.f, 1.f, 1.f},    // front
            { 1.f, 1.f,-1.f}, {-1.f, 1.f,-1.f}, {-1.f, 1.f, 1.f}, { 1.f, 1.f, 1.f},    // up
            {-1.f,-1.f,-1.f}, { 1.f,-1.f,-1.f}, { 1.f,-1.f, 1.f}, {-1.f,-1.f, 1.f},    // down
            { 1.f,-1.f,-1.f}, { 1.f, 1.f,-1.f}, { 1.f, 1.f, 1.f}, { 1.f,-1.f, 1.f},    // right
            {-1.f, 1.f,-1.f}, {-1.f,-1.f,-1.f}, {-1.f,-1.f, 1.f}, {-1.f, 1.f, 1.f}     // left
};
```




```
glBegin(GL_QUADS);
  for(i=0; i<24; i++) {
    glVertex3fv(vv[i]);
  }
glEnd();
```

Extra calls needed

- **24 vertex calls may be required.**
  - There can be extra calls for normal, color, and texture.
  - **Too many GL command calls in every frame!**

```
glBegin(GL_LINES);
    for(int i = 0; i<24; i++)
    {
        glVertex3f(vertices[3*i+ 0], vertices[3*i + 1], vertices[3*i + 2]);
        glColor3f(colors[3*i+ 0], colors[3*i + 1], colors[3*i + 2]);
        glVertex3f(normals[3*i+ 0], normals[3*i + 1], normals[3*i + 2];
    }
glEnd();
```

Display List

- **Keeping a group of OpenGL commands in the device memory**
  - Once created & compiled, all the command calls and associated data are copied into the device memory (i.e., GPU's memory)
  - Reduces CPU cycles taken for the transfer
- **Usage**

```
// creating a display list
GLuint dlist = glGenLists(1);
// compiling the display list
glNewList(dlist, GL_COMPLIE);
glBegin(GL_QUADS);
...
glEnd();
glEndList();
```

```
// drawing the display list
glCallList(dlist);



// delete when not used any more
glDeleteLists(dlist, 1);
```

Per vertex data array

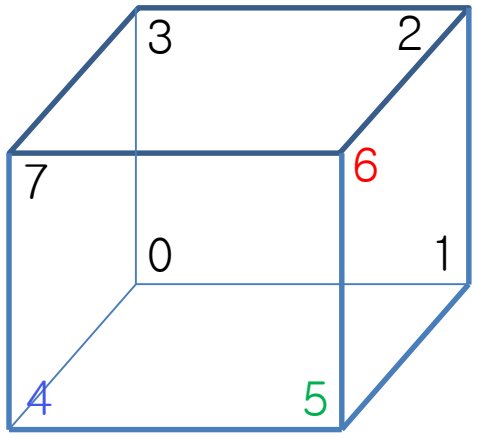
- **Instead of sending individual GL commands to GPU, it sends only the data and lets the drawing be done at the GPU side.**
- **One possible such data are per-vertex data arrays**
  - Per-vertex data arrays can consist of
    - Vertex array
    - Normal array
    - Color array
    - Texture coordinate array

Usage of Per-Vertex Data Array

With 24 (vertex + color)

```
// Activating..
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY); // there can be more enabling..
// Specifying the data..
glVertexPointer(3, GL_FLOAT, 0, vv);
glColorPointer(3, GL_FLOAT, 0, cc);  // glNormalPointer()..
// Now, the data are in the GPU side.
// With this context, drawing can be done by a single GL command..
glDrawArrays(GL_QUADS, 0, 24);
// Deactivating
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
```

```
glVertexPointer(size, type, stride(offset), data);
glDrawArrays(mode, first index, count);
```



Texture Mapping - Review

```
void init() {
   unsigned char bitmap[DIMX*DIMY*3];
   ...
   GLuint texID;
   glEnable(GL_TEXTURE_2D);
   glGenTextures(1, &texID);
   glBindTexture(GL_TEXTURE_2D, texID);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, DIMX, DIMY, 0,
                GL_RGB, GL_UNSIGNED_BYTE, bitmap);
}
```

Image data residing in the host

Generating a texture id

Start binding the texture

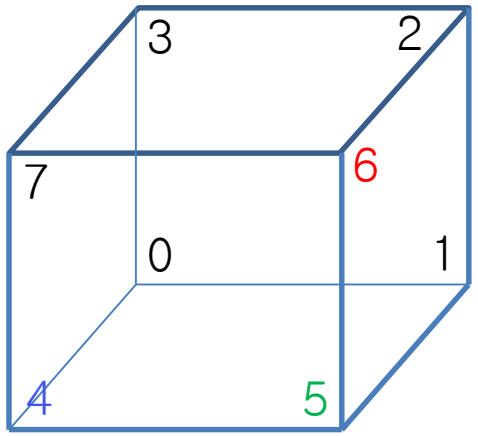
↓ The details

Specify the image data for this texture object.
Now the copy of this texture is existing in the device memory.

Usage of Usage of Per-Vertex Data Array

With 8 (vertex + color + normal) and 24 index

```
// Activating..
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
// Specifying the data..
glVertexPointer(3, GL_FLOAT, 0, v);
glColorPointer(3, GL_FLOAT, 0, c);
glNormalPointer(GL_FLOAT, 0, n);
// This time drawing is done with indices..
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, ind);
// Deactivating as before..
glVertexPointer(size, type, stride(offset), data);
glDrawElements(mode, number of indices, type, data);
```




```
GLubyte ind[24] = {
          0, 3, 2, 1, // back
          4, 5, 6, 7, // front
          2, 3, 7, 6, // up
          0, 1, 5, 4, // down
          1, 2, 6, 5, // right
          3, 0, 4, 7  // left
};
```

Usage of Usage of Per-Vertex Data Array

Syntax of the Commands

```
glVertexPointer(size, type, stride(offset), data);
glDrawArrays(mode, first index, count);
glDrawElements(mode, number of indices, type, data);
```

glVertex tends to be deprecated

- OpenGL ES, another OpenGL family for embedded system, does not have glVertex command.
- Microsoft's DirectX series also do not provide such immediate vertex command.

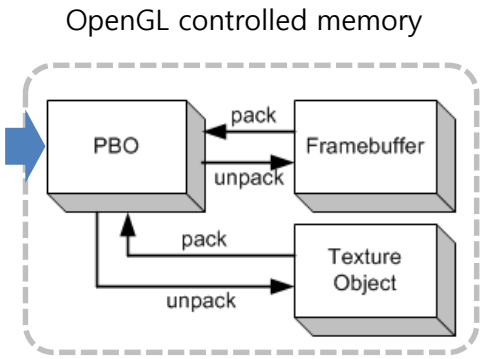
Comparison

- **Display List**
  - id-based, thus once the display list is defined, no further transfer of that list is needed.
  - Once a list is defined, however, it cannot be modified.
  - Summary: flexibility ↓, reusability ↑
- **Per-Vertex Data Array**
  - Reduces command calls & redundant transfer of shared data.
  - The context cannot be saved for further referencing.
    - There isn't any such thing as "vertex array id".
    - When program switches among multiple contexts, the arrays must be resent.
  - Summary: flexibility ↑, reusability ↓
- **Abstract Buffer Objects**
  - They create buffer objects for vertex or pixel data on the device memory
  - They provide functions to reference the data
  - They can be read and updated by mapping the buffer
  - Summary: flexibility ↑, reusability ↑

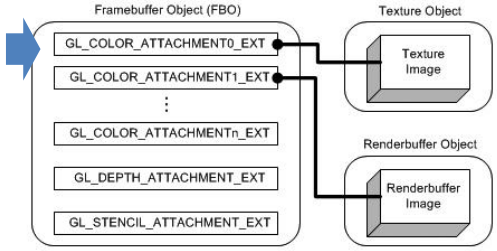
Abstract Buffer Objects

- **Vertex Buffer Object (VBO)**
  - allows vertex array data to be stored in the device memory.
  - *GL_ARB_vertex_buffer_object*

- **Pixel Buffer Object (PBO)**
  - allows pixel data to be stored in the device memory for further intra-GPU transfer
  - *GL_ARB_pixel_buffer_object*



- **Frame Buffer Object (FBO)**
  - allows rendered contents (color, depth, stencil) to be stored in non-displayable framebuffers (e.g., texture object, renderbuffer object)
  - *GL_EXT_framebuffer_object*



Usage of Vertex Buffer Object

With 24 (vertex + color)

```
// Similar to creating texture
GLuint vboId;
float data[] = {...};
glGenBuffers(1, &vboId);
glBindBuffer(GL_ARRAY_BUFFER, vboId);
glBufferData(GL_ARRAY_BUFFER, sizeof(data), data, GL_STATIC_DRAW);
// Drawing is similar to vertex array..
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY); // can enable more arrays..
glVertexPointer(3, GL_FLOAT, 0, 0);  // Starting offset of the buffer
glColorPointer(3, GL_FLOAT, 0, (void *)(sizeof(data)/2));
glDrawArrays(GL_QUADS, 0, 24); // glDrawElements() can be used instead
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
// Unbinding vboId..
glBindBuffer(GL_ARRAY_BUFFER, 0);
// deleting vboId..
glDeleteBuffers(1, &vboId);
```

Usage of Vertex Buffer Object

With 24x2 (vertex + color)

```
float data[48][3] = {
        {-1.f,-1.f,-1.f}, {-1.f, 1.f,-1.f}, { 1.f, 1.f,-1.f}, { 1.f,-1.f,-1.f},   // back
        {-1.f,-1.f, 1.f}, { 1.f,-1.f, 1.f}, { 1.f, 1.f, 1.f}, {-1.f, 1.f, 1.f},   // front
        { 1.f, 1.f,-1.f}, {-1.f, 1.f,-1.f}, {-1.f, 1.f, 1.f}, { 1.f, 1.f, 1.f},   // up
        {-1.f,-1.f,-1.f}, { 1.f,-1.f,-1.f}, { 1.f,-1.f, 1.f}, {-1.f,-1.f, 1.f},   // down
        { 1.f,-1.f,-1.f}, { 1.f, 1.f,-1.f}, { 1.f, 1.f, 1.f}, { 1.f,-1.f, 1.f},   // right
        {-1.f, 1.f,-1.f}, {-1.f,-1.f,-1.f}, {-1.f,-1.f, 1.f}, {-1.f, 1.f, 1.f},   // left

        { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, // back  0, 3, 2, 1
        { 0.f, 0.f, 1.f}, { 0.f, 1.f, 0.f}, { 1.f, 0.f, 0.f}, { 1.f, 1.f, 1.f}, // front 4, 5, 6, 7
        { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, { 1.f, 1.f, 1.f}, { 1.f, 0.f, 0.f}, // up    2, 3, 7, 6
        { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, { 0.f, 1.f, 0.f}, { 0.f, 0.f, 1.f}, // down  0, 1, 5, 4
        { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, { 1.f, 0.f, 0.f}, { 0.f, 1.f, 0.f}, // right 1, 2, 6, 5
        { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, { 0.f, 0.f, 1.f}, { 1.f, 1.f, 1.f}  // left  3, 0, 4, 7
};
```

Usage of Vertex Buffer Object

With 8 (vertex + color) and 24 index

```
float v[8][3] = {
            {-1.f,-1.f,-1.f}, { 1.f,-1.f,-1.f}, { 1.f, 1.f,-1.f}, {-1.f, 1.f,-1.f},
            {-1.f,-1.f, 1.f}, { 1.f,-1.f, 1.f}, { 1.f, 1.f, 1.f}, {-1.f, 1.f, 1.f}
};
float c[8][3] = {
            { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f}, { 0.f, 0.f, 0.f},
            { 0.f, 0.f, 1.f}, { 0.f, 1.f, 0.f}, { 1.f, 0.f, 0.f}, { 1.f, 1.f, 1.f}
};
GLubyte ind[24] = {
            0, 3, 2, 1, // back
            4, 5, 6, 7, // front
            2, 3, 7, 6, // up
            0, 1, 5, 4, // down
            1, 2, 6, 5, // right
            3, 0, 4, 7  // left
};
// Suppose that VBO for vertex, color, normal is already created…
GLuint vboElementId;
glGenBuffers(1, &vboElementId);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboElementId);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(ind), ind, GL_STATIC_DRAW);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
...
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboElementId);
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, 0); // Starting offset
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

Another Example Code

With 24 (vertex + normal + color) subdata sets

```
glGenBuffers(1, &vboID);
glBindBuffer(GL_ARRAY_BUFFER, vboID);

glBufferData(GL_ARRAY_BUFFER,
             sizeof(vv)+sizeof(nn)+sizeof(cc), 0, GL_STATIC_DRAW);
glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(vv), vv);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(vv), sizeof(vv)+sizeof(nn), nn);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(vv)+sizeof(nn),
                                        sizeof(vv)+sizeof(nn)+sizeof(cc), cc);
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);

glVertexPointer(3, GL_FLOAT, 0, 0);
glNormalPointer(GL_FLOAT, 0, (void*) sizeof(vv));
glColorPointer(3, GL_FLOAT, 0, (void*) (sizeof(vv)+sizeof(nn)));

glDrawArrays(GL_QUADS, 0, 24);

glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Data Transfer: VBO, VAO

Data in OpenGL

- **Everything you will ever do with OpenGL will involve buffers full of data.**
  - We call buffers in OpenGL "buffer object"

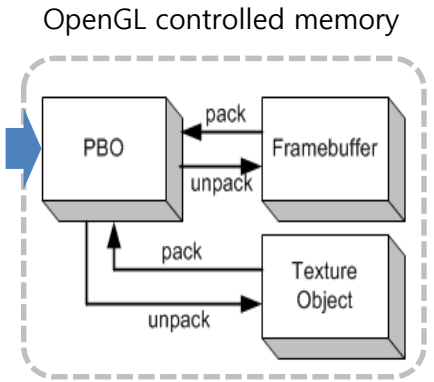
OpenGL Object

- **Buffer Object**
  - **Vertex Buffer Object :** uploading vertex data to the VRAM
  - **Element Buffer Object:** uploading index data for vertices to the VRAM.
  - **Pixel Buffer Object:** used for asynchronous pixel transfer operation.
  - **Frame Buffer Object:** used for rendered contents to be stored in non-displayable framebuffers
  - **Storage Buffer Object:** used to store and retrieve data from with i n the GLSL (Especially for compute shader)
  - **Uniform Buffer Object:** used to store uniform data for a shader p rogram
- Vertex Array Object, Texture …

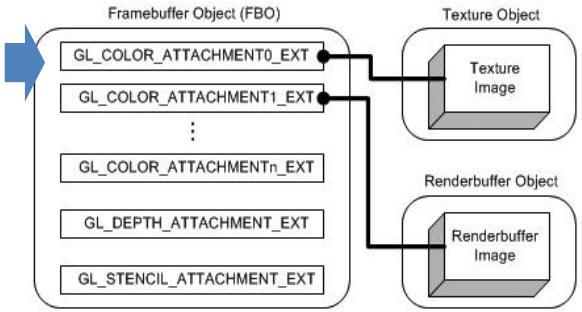
Abstract Buffer Objects

- **Vertex Buffer Object (VBO)**
  - allows vertex array data to be stored in the device memory.
  - *GL_ARB_vertex_buffer_object*

- **Pixel Buffer Object (PBO)**
  - allows pixel data to be stored in the device memory for further intra-GPU transfer
  - *GL_ARB_pixel_buffer_object*



- **Frame Buffer Object (FBO)**
  - allows rendered contents (color, depth, stencil) to be stored in non-displayable framebuffers (e.g., texture object, renderbuffer object)
  - *GL_EXT_framebuffer_object*



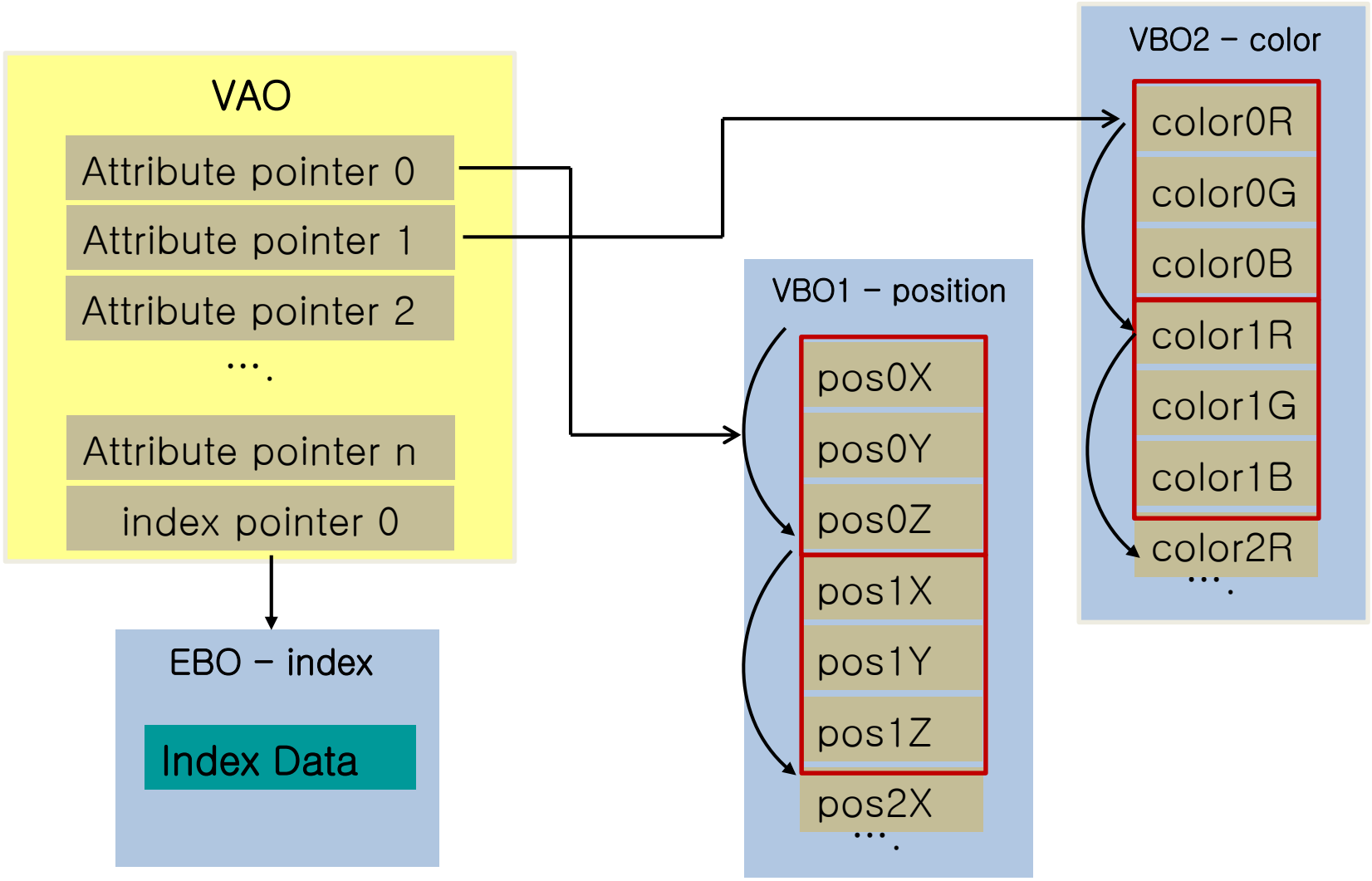
VBO, VAO and EBO

- **Vertex Buffer Object**
  - Buffer for vertex data
    - Vertex data ≡ Per vertex data
  - Position, Normal vector, Color, etc.

- **Vertex Array Object**
  - Special type of object that encapsulates all the vertex data
  - Instead of containing the actual data, it holds references to the vertex buffers, the index buffer

- **Element Buffer Object?**
  - Special type of object that encapsulates index data of vertices

Concept of VBO, VAO and EBO



Create VBO

- **Create & Initialize VBO**

```
//ID
GLuint buffer;

//Generate buffers
glGenBuffers(1, &buffer);

//Bind
glBindBuffer(GL_ARRAY_BUFFER, buffer);

//Initialize & Transfer
1) glBufferData(GL_ARRAY_BUFFER, Number, data, GL_STATIC_DRAW);

2) glBufferData(GL_ARRAY_BUFFER, Number, NULL, GL_STATIC_DRAW);
glBufferSubdata(GL_ARRAY_BUFFER, 0, sizeof(data), data);

//Unbind
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Update VBO

- **There are two ways to update the VBO.**
  - glBufferSubData() and glMapBuffer()

```
static const float data[] = { 1.0, 1.0, 1.0, 1.0 };

//First method - glbuffersubdata
glBufferSubdata(GL_ARRAY_BUFFER, 0, sizeof(data), data);


//Second method - mapping
void *ptr = glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);

memcpy(ptr, data, sizeof(data));

glUnmapBuffer(GL_ARRAY_BUFFER);
```

Create & Update VAO

- **glBindBuffer(GL_ARRAY_BUFFER, buffer) tells that the VBO buffer is to be attached to VAO.**
  - glVertexAttribPointer(…) tells the details of the above attachment.

```
//Create VAO
Gluint VAO;
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);

//Refer buffer(VBO)
glBindBuffer(GL_ARRAY_BUFFER, buffer);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, NULL);
glEnableVertexAttribArray(0);

//unbind
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

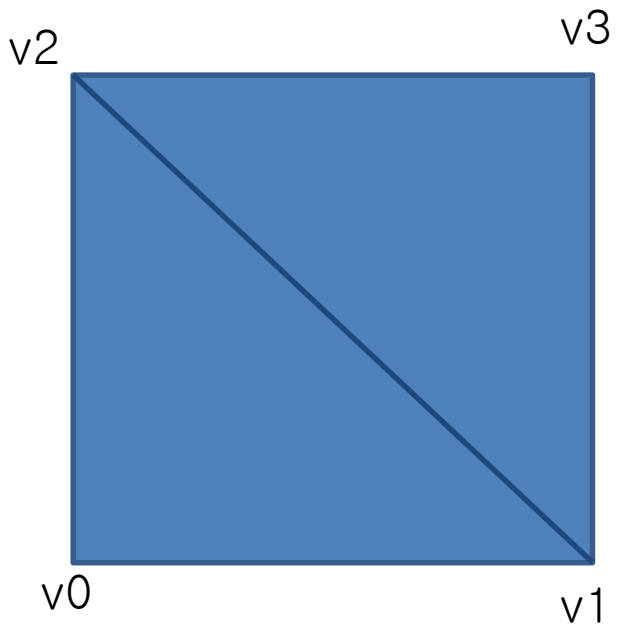
Draw VAO

- **Draw without index**
  - `glDrawArrays(mode, first, count);`
  - Constructs a sequence of geometric primitives using array elements starting at first index and ending at first + count – 1 of each enabled array.

- **Draw with index**
  - `glDrawElements(mode, number of indices, type, data);`

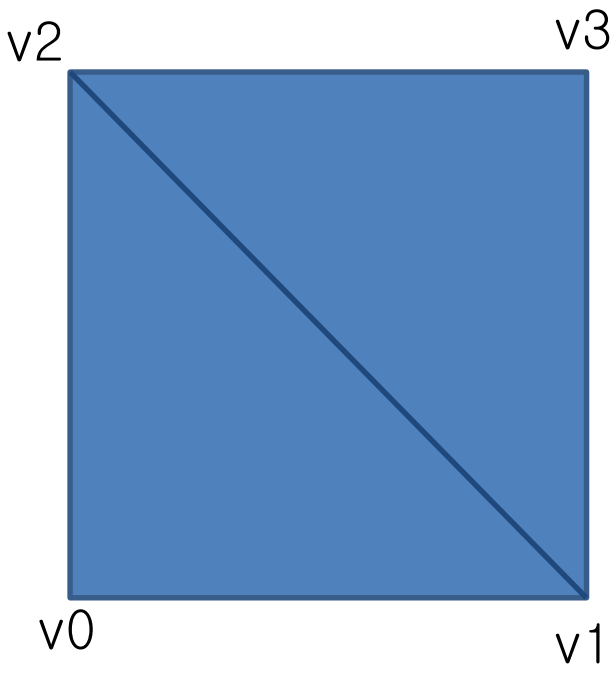
Draw VAO Example

- Draw without index
  - `glDrawArrays(GL_TRIANGLES, 0, 6);`
- Draw with index
  - `glDrawElements(GL_TRIANGLES, 6, GL_UNSIGED_SHORT, indx_data);`



Draw VAO without index

- Draw without index

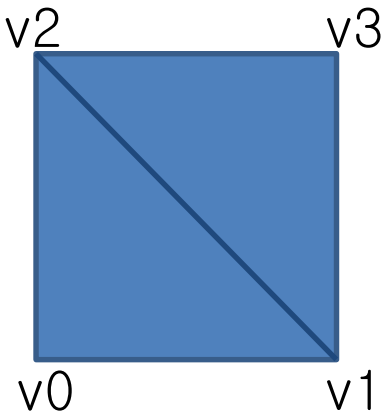


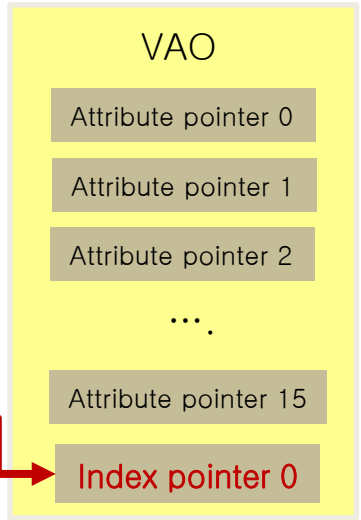

```
static const float position[] = { -1.0f, -1.0f, 0.0f,    //v0
                                  -1.0f, -1.0f, 0.0f,    //v1
                                  -1.0f, -1.0f, 0.0f,    //v2
                                  -1.0f, -1.0f, 0.0f,    //v1
                                   -1.0f, -1.0f, 0.0f,   //v3
                                  -1.0f, -1.0f, 0.0f,    //v2
} ;

GLuint VAO;
GLuint buffer;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &buffer);

glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(positions), positions, GL_STATIC_DRAW);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, NULL);
glEnableVertexAttribArray(0);

//display function
glDrawArrays(GL_TRIANGLES, 0, 6);
```

Draw VAO with Index

- Draw with index

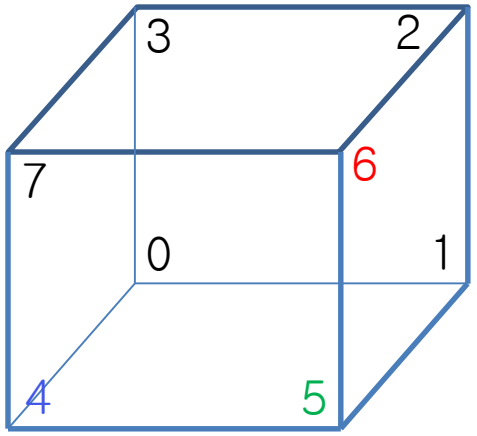
```
static const float position[] = { -1.0f, -1.0f, 0.0f,    //v0
                                  -1.0f, -1.0f, 0.0f,    //v1
                                  -1.0f, -1.0f, 0.0f,    //v2
                                  -1.0f, -1.0f, 0.0f,    //v3
} ;
static const unsigned short index[] = { 0, 1, 2, 1, 3, 2};

GLuint VAO;
GLuint buffer
GLuint indice
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &buffers);
glGenBuffers(1, &indice);

glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(positions), positions, GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, NULL);
glEnableVertexAttribArray(0);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indice);
glBufferData(GL_ARRAY_BUFFER, sizeof(indice), indice, GL_STATIC_DRAW);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

//display function
glDrawElement(GL_TRIANGLES, 6, GL_UNSIGNED_SHORT, 0);
```





Practice: Draw WireCube with VAO

- **Copy Sample Skeleton Code**
  - vglconnect ID@163.152.20.246
  - cp –r /home/share/DataTransfer ./
  - cd DataTransfer
- **Notepad: DataTransfer function 작성**
- **Compile & run program**
  - make
  - vglrun ./EXE

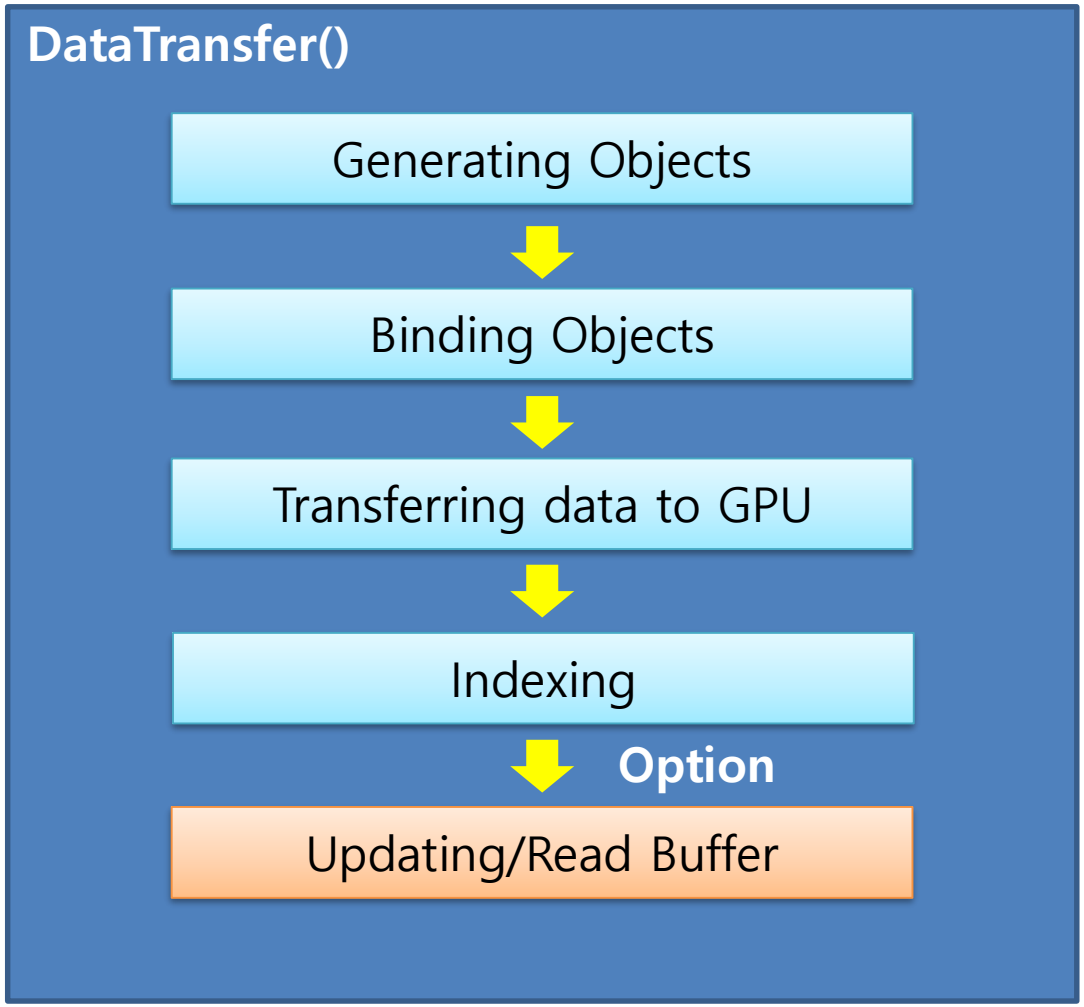


Program structure

```
#include "XWindow.h"
#include <stdio.h>
#include <stdlib.h>
#include <string>
using namespace std;
//function declaration//
//Global variables//

int main(int argc, char *argv[]) {
    //Window Initialization//

    InitGL();
    DataTransfer();
    while(1) {
        Display();
        KeyboardCallback();
    }
}
```

```
float vertices[] = { 0.5f, 0.5f, 0.5f, //Node 0
                     0.5f, 0.5f,-0.5f, //Node 1
                    -0.5f, 0.5f,-0.5f, //Node 2
                    -0.5f, 0.5f, 0.5f, //Node 3
                     0.5f,-0.5f, 0.5f, //Node 4
                     0.5f,-0.5f,-0.5f, //Node 5
                    -0.5f,-0.5f,-0.5f, //Node 6
                    -0.5f,-0.5f, 0.5f  //Node 7
}

unsigned short indices = { 0, 1,  1, 2,  2, 3,  3, 0,
                           0, 4,  1, 5,  2, 6,  3, 7,
                           4, 5,  5, 6,  6, 7,  7, 4
}
```

Data Transfer function structure



DataTransfer Code@Main

```
void DataTransfer(){

    //Generating Buffer Objects
    glGenVertexArrays(1, VAO);
    glGenBuffers(1, VBO);
    glGenBuffers(1, EBO);
    //Transferring Vertex data to Device
    glBindVertexArray(VAO[0]);
    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
    glBufferData(GL_ARRAY_BUFFER, 4 * sizeof(vertices), vertices, GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER,0);
    //Transferring Index data to Device
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO[0]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, 4 * sizeof(indices), indices, GL_STATIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

Vertex Attribute pointer

VAO

VBO1

EBO

Shader code

```
//Vertex Shader code
#version 130
layout(location = 0)in vec3 aPos;//Alternative to Attribute type  variable
unitform  mat4 modelview
void main()
{
        gl_Position = modelview * vec4(aPos.x, aPos.y, aPos.z, 1.0);
}

//Fragment Shader code
#version 130
void main()
{
        vec4 color = vec4(0.0,0.0,0.0,1.0);
        gl_FragColor = color;
}
```

Vertex Attribute pointer

Display function@Main

```
void display(){
        glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glUseProgram(Program);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glRotatef(15.0f, -1.0f, 1.0f, 0.0f);
        glBindVertexArray(VAO[0]);
        glDrawElements(GL_LINES, 24, GL_UNSIGNED_SHORT, 0);
        glXSwapBuffers(dpy, win);
        glUseProgram(0);
}
```

Result

- Result of the program is same with glBegin program.
- However, Data Transfer is occurred only once!
- As the amount of the data increases, Buffer objects get more useful.