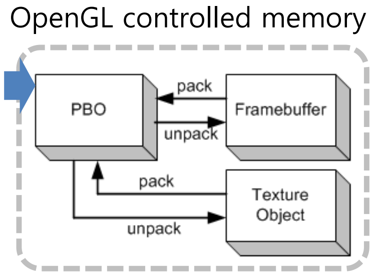
Data Transfer: PBO & FBO

KOREA UNIVERSITY

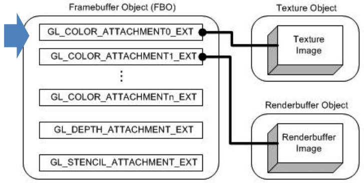
KUCG

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*

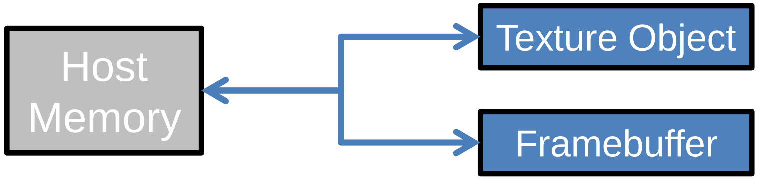


Pixel Buffer Object

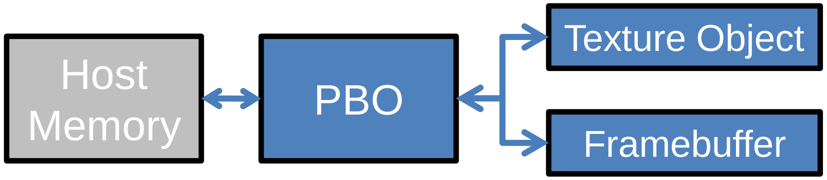
- **Can be considered as an extension of VBO**
  - But instead of storing vertex data, it stores **pixel data**
  - Pixel data can be managed more efficiently via PBO

Speeding up Pixel Data Transfer with PBO

- **Via PBO, you can make pixel data transfer done within the device memory.**
  - Conventional Pixel Data Transfer

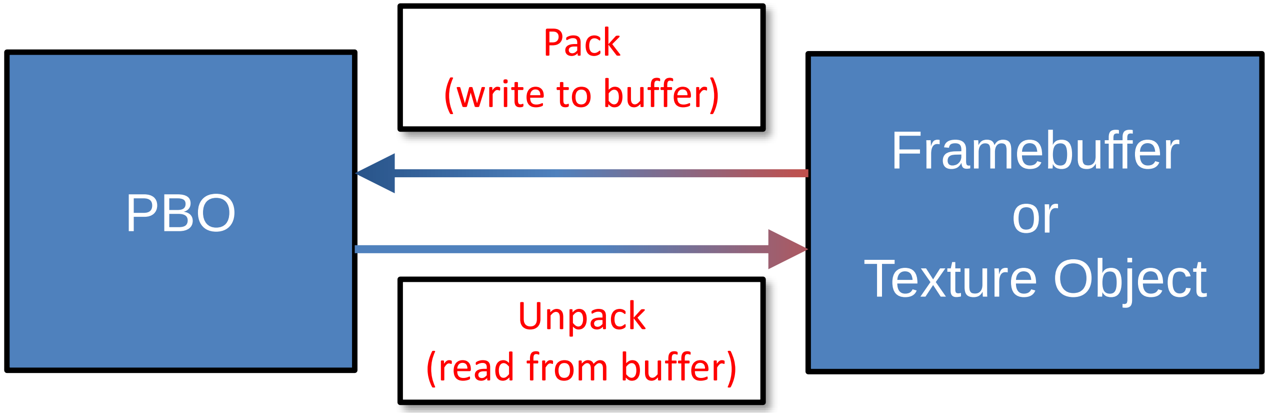


  - Using PBO



Usage of PBO

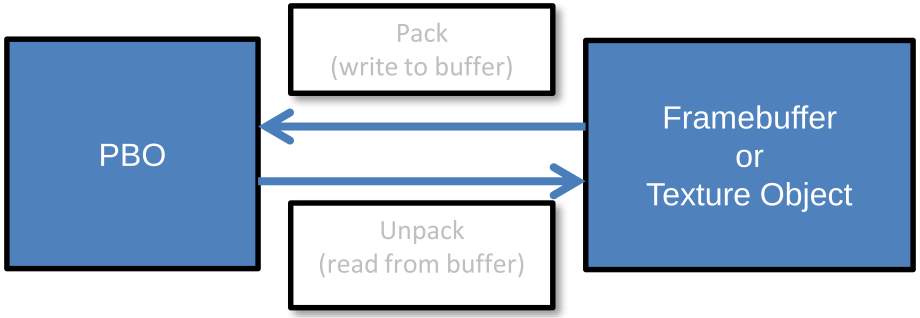
- **PBO has two targets (storages):**
  - GL_PIXEL_PACK_BUFFER
  - GL_PIXEL_UNPACK_BUFFER



Usage of PBO: Creating & Deleting

- Usage: Create & Delete

```
// Similar to creating VBO
GLuint pboId;
glGenBuffers(1, &pboId);
...
glDeleteBuffers(1, &pboId);
```

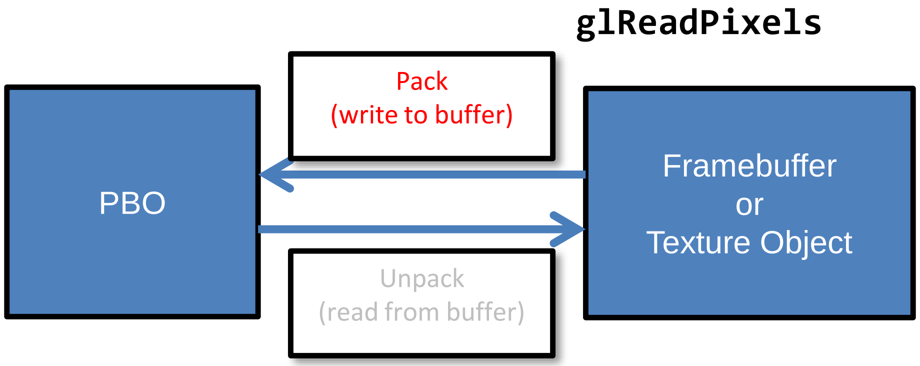


Usage of PBO: Reading framebuffer

- Usage: PBO for reading

```
// For example, read pixels from the front framebuffer to PBO
glReadBuffer(GL_FRONT);
glBindBuffer(GL_PIXEL_PACK_BUFFER, pboId);
glReadPixels(0,0, Width,Height, GL_RGBA, GL_UNSIGNED_BYTE, 0);
```

With the current pbo context, 0 means the data is transferred to pbo

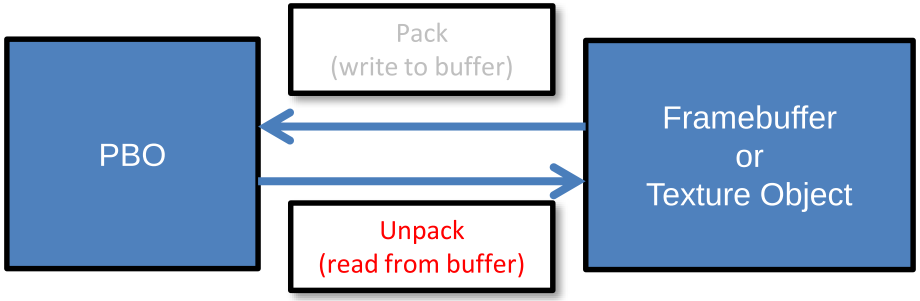


Usage of PBO: Writing to Texture

- Usage: PBO for writing

```
// For example, copy pixels from PBO to texture object
glBindTexture(GL_TEXTURE_2D, texId);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboId);
glTexSubImage2D(GL_TEXTURE_2D,0, 0,0,w,h, GL_RGBA, GL_UNSIGNED_BYTE, 0);
```

With the current pbo context, 0 means the data source is pbo

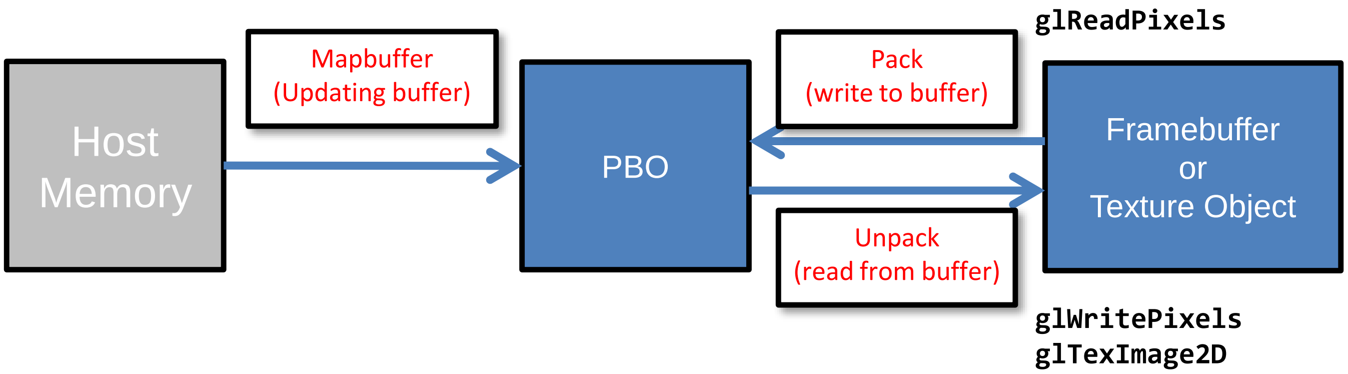


```
glWritePixels
glTexImage2D
```

Usage of PBO: Updating data

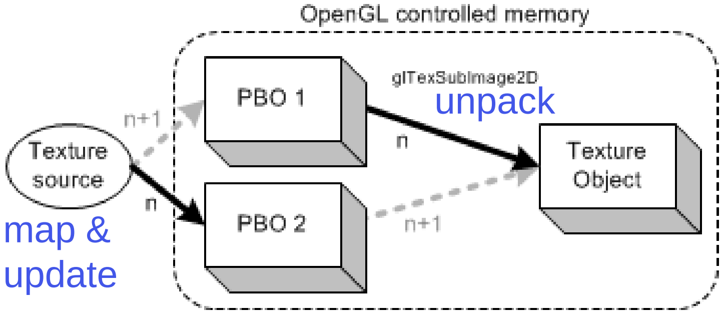
- Usage: Update PBO

```
// Similar to updating VBO
glBindBuffer(GL_PIXEL_(UN)PACK_BUFFER, pboId);
Glubyte *ptr = glMapBuffer(GL_PIXEL_(UN)PACK_BUFFER, GL_(WRITE)READ_ONLY);
if(ptr) {
    // Update data directly on the mapped buffer
    ...
    glUnmapBuffer(GL_PIXEL_(UN)PACK_BUFFER);
}
```

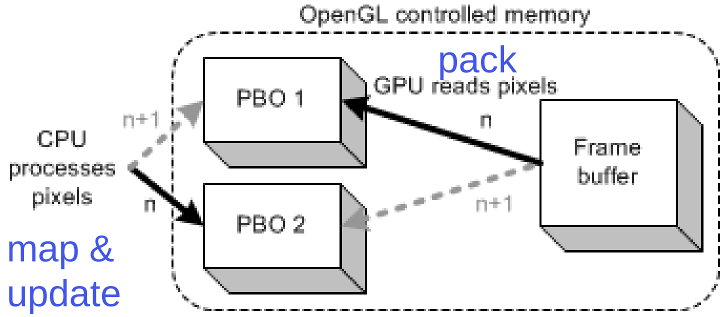

glReadPixels
Host Memory
Mapbuffer (Updating buffer)
PBO
Pack (write to buffer)
Framebuffer or Texture Object
Unpack (read from buffer)
glWritePixels
glTexImage2D

Use of Multiple PBOs

- **To maximize the streaming performance, multiple PBOs can be used.**
  - ex. Asynchronous uploading textures from CPU



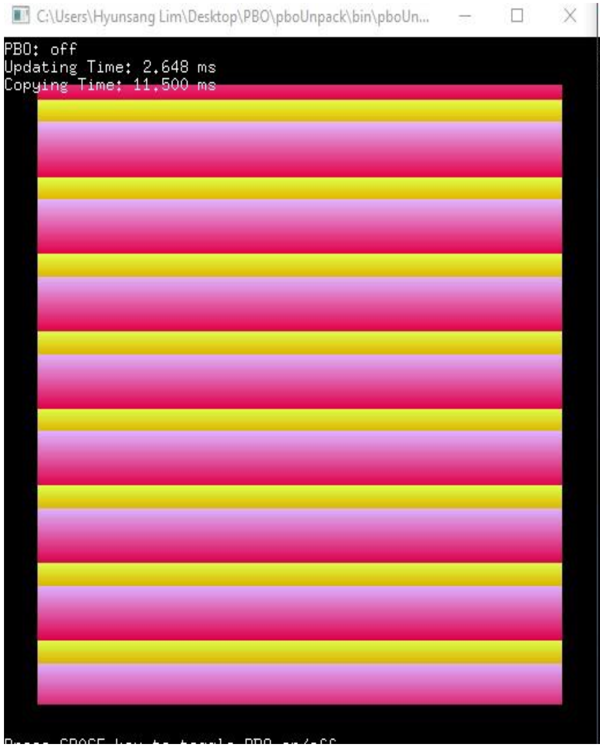
  - ex. Asynchronous read-back



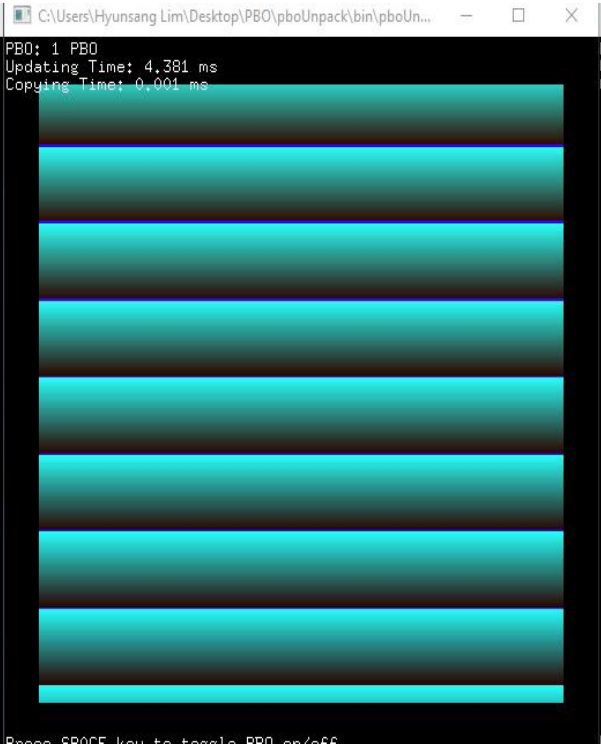
PBO Coding Exercise

- **Copy Sample Skeleton Code**
  - vglconnect ID@163.152.20.246
  - cp –r /home/share/14_PBO ./[Folder name]
  - cd [Folder name]
- **Notepad: Shader code 수정**
- **Compile program**
  - make
  - vglrun ./EXE

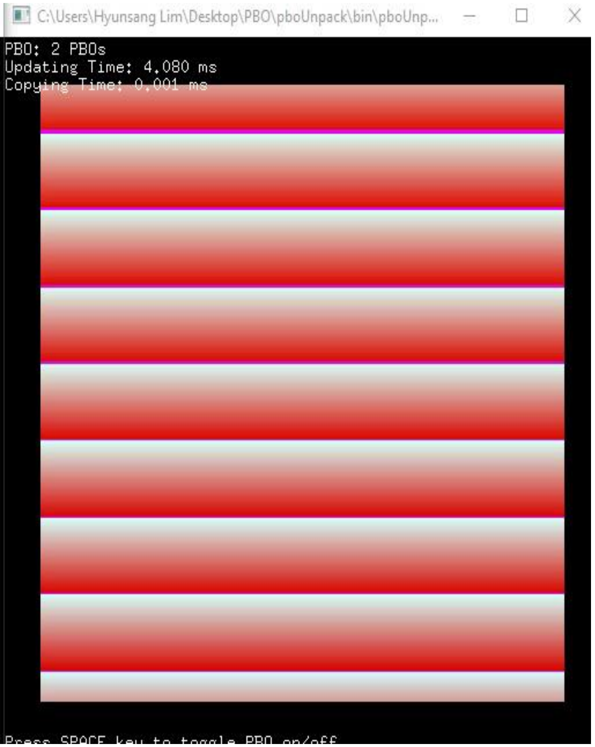
Expected Result



```
PBO: off
Updating Time: 2.648 ms
Copying Time: 11.500 ms

Press SPACE key to toggle PBO on/off
```

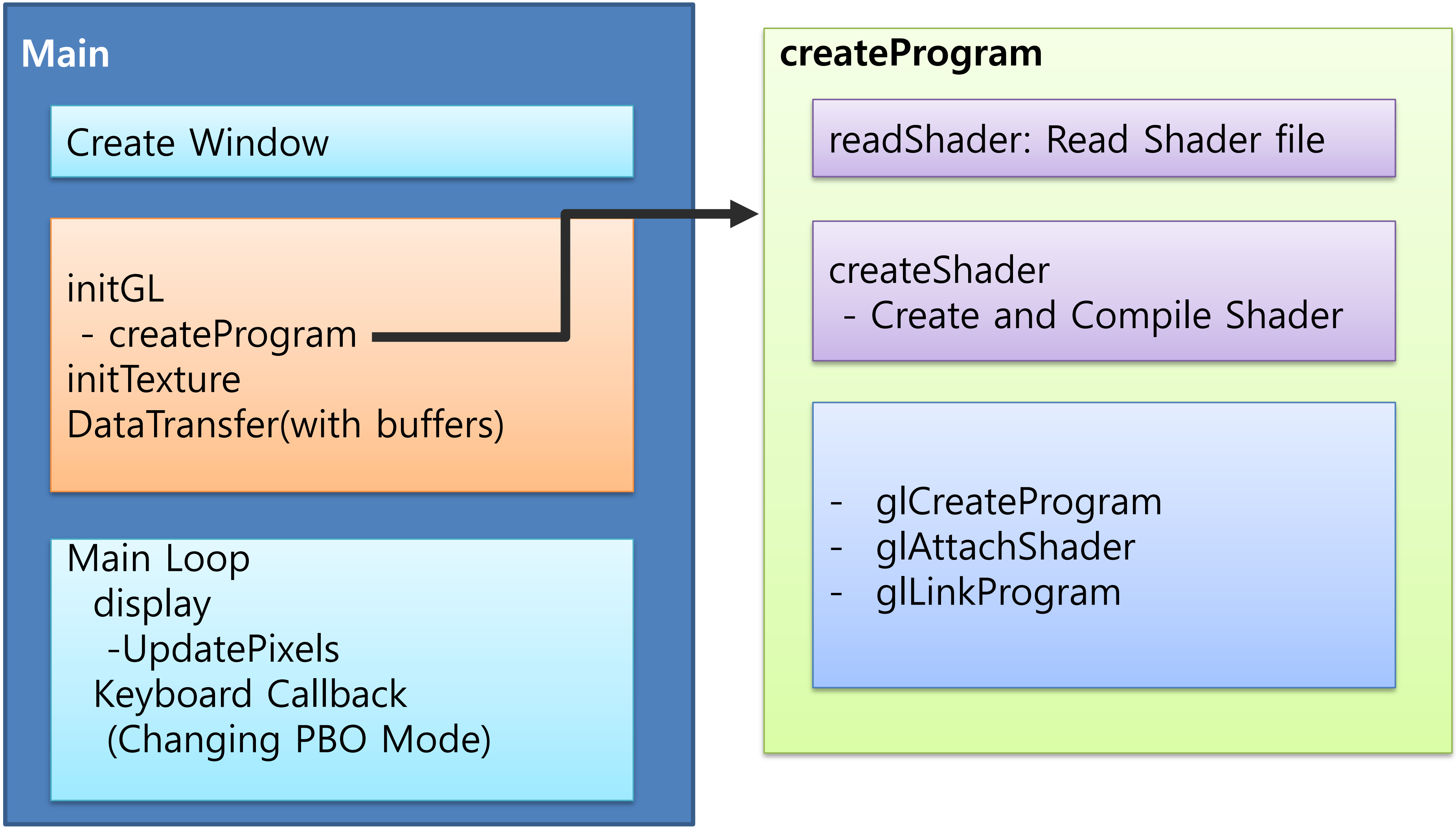


```
PBO: 1 PBO
Updating Time: 4.381 ms
Copying Time: 0.001 ms

Press SPACE key to toggle PBO on/off
```



```
PBO: 2 PBOs
Updating Time: 4.080 ms
Copying Time: 0.001 ms

Press SPACE key to toggle PBO on/off
```

Program Flow



Main

Create Window

initGL
- createProgram

initTexture

DataTransfer(with buffers)

Main Loop
  display
    -UpdatePixels
  Keyboard Callback
    (Changing PBO Mode)

createProgram

readShader: Read Shader file

createShader
- Create and Compile Shader

- glCreateProgram
- glAttachShader
- glLinkProgram

Program structure

```
#include headers
using namespace std;
//function declaration//
//Global variables//
float *vertices;
float *textCoord;
Float *normals;
int main(int argc, char *argv[]) {

    //Window Initialization//
    initGL();
    initTexture();
    DataTransfer();
    while(1) {
        Display();
        KeyboardCallback();
         }
     }
}
```

initGL()

```
void initGL(){
        imageData = new GLubyte[DATA_SIZE];
        memset(imageData, 0, DATA_SIZE); //initialization of pixel data with 0
        glewInit();
        createProgram();
        glEnable(GL_DEPTH_TEST);
}

void createProgram(){
        char *vertexShaderSource = ReadFile("Vertex.glsl");
        char *fragmentShaderSource = ReadFile("Fragment.glsl");
        unsigned int VertShader = createShader(vertexShaderSource, GL_VERTEX_SHADER);
        unsigned int FragShader = createShader(fragmentShaderSource, GL_FRAGMENT_SHADER);

        Program = glCreateProgram();
        glAttachShader(Program, VertShader);
        glAttachShader(Program, FragShader);
        glLinkProgram(Program);
}
```

initTexture()

```
void initTexture(){
		glGenTextures(1, &textureID);
		glBindTexture(GL_TEXTURE_2D, textureID);
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
		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_RGBA8, IMAGE_WIDTH, IMAGE_HEIGHT, 0, 
PIXEL_FORMAT, GL_UNSIGNED_BYTE, (GLvoid*)imageData); //initializing pixel data with ImageData
		glBindTexture(GL_TEXTURE_2D, 0);
}
```

DataTransfer Code@Main – PBO part

```
void DataTransfer(){
        //Generating PBO
        glGenBuffers(2, pboIds);
        //Binding
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboIds[0]);
        //Allocating GPU Memory for pixel data
        glBufferData(GL_PIXEL_UNPACK_BUFFER, DATA_SIZE, 0, GL_STREAM_DRAW);
         //Binding
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboIds[1]);
         //Allocating GPU Memory for pixel data
        glBufferData(GL_PIXEL_UNPACK_BUFFER, DATA_SIZE, 0, GL_STREAM_DRAW);
         //Unbinding
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
```

DataTransfer Code@Main – VAO part

```
glGenVertexArrays(1, VAO);
glGenBuffers(2, VBO);
glGenBuffers(1, EBO);
glBindVertexArray(VAO[0]);
```

```
float vertices[] = { 1.0f, 1.0f, 0.0f,
                    -1.0f, 1.0f, 0.0f,
                    -1.0f,-1.0f, 0.0f,
                     1.0f,-1.0f, 0.0f}
```

```
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);
```

```
float texcoord[] = { 1.0f, 0.0f,
                     0.0f, 0.0f
                     0.0f, 1.0f,
                     1.0f, 1.0f}
```

```
glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
glBufferData(GL_ARRAY_BUFFER, 4 * sizeof(texcoord), texcoord, GL_STATIC_DRAW);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
glEnableVertexAttribArray(1);
```

```
Int indices[] = { 0, 1, 2
                  2, 3, 0}
```

```
    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);
}
```

Display function@Main

```
void display() {
    static int index = 0;                 pboMode is changed by Keyboad input.
    int nextIndex = 0;
    if (pboMode > 0){
        if (pboMode == 1){
            index = nextIndex = 0;          Single PBO Mode
        }
        else if (pboMode == 2){
            index = (index + 1) % 2;          Double PBO Mode
            nextIndex = (index + 1) % 2;
        }
        glBindTexture(GL_TEXTURE_2D, textureID);
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboIds[index]);
        glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, IMAGE_WIDTH, IMAGE_HEIGHT, PIXEL_FORMAT, GL_UNSIGNED_BYTE, 0); //Data Transfer
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboIds[nextIndex]);
        glBufferData(GL_PIXEL_UNPACK_BUFFER, DATA_SIZE, 0, GL_STREAM_DRAW);
        GLubyte* ptr = (GLubyte*)glMapBuffer(GL_PIXEL_UNPACK_BUFFER, GL_WRITE_ONLY);
        if (ptr)
        {
            updatePixels(ptr, DATA_SIZE); //Updating Pixel data function
            glUnmapBuffer(GL_PIXEL_UNPACK_BUFFER);  // release pointer to mapping buffer
        }
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
    }
    else{                                       Not using PBO
        glBindTexture(GL_TEXTURE_2D, textureID);
        glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, IMAGE_WIDTH, IMAGE_HEIGHT, PIXEL_FORMAT, GL_UNSIGNED_BYTE, (GLvoid*)imageData);
        updatePixels(imageData, DATA_SIZE); //Updating Pixel data function
    }
```

Display function@Main

```
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

        glBindTexture(GL_TEXTURE_2D, textureID);
        glUseProgram(Program);
        glBindVertexArray(VAO[0]);
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);

        glUseProgram(0);
        glBindVertexArray(0);
        glBindTexture(GL_TEXTURE_2D, 0);
        glXSwapBuffers(dpy, win);
}
```

updatePixels@Display

```
void updatePixels(GLubyte* dst, int size){
    static int color = 0;
    if (!dst)
        return;
    int* ptr = (int*)dst;
    // copy 4 bytes at once
    for (int i = 0; i < IMAGE_HEIGHT; ++i){
        for (int j = 0; j < IMAGE_WIDTH; ++j){
            *ptr = color;
            ++ptr;
        }
        color += 257;
    }
    ++color;
}
```

Shader code

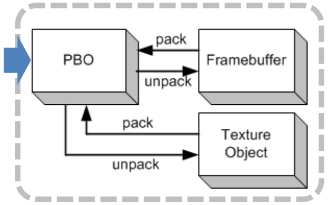
```
//Vertex Shader code
#version 130
layout(location = 0) in vec3 aPos;
layout(location = 1) in vec2 aTexcoord;
out vec2 Texcoord;
void main()
{
        gl_Position = vec4(aPos, 1.0);
        Texcoord = aTexcoord;
}
//Fragment Shader code
#version 130
in vec2 Texcoord;
out vec4 FragColor;
uniform sampler2D texture1;
void main()
{
        FragColor = texture(texture1, Texcoord);
}
```

Abstract Buffer Objects

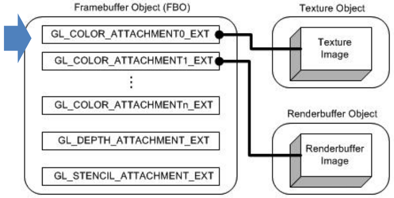
- Vertex Buffer Object (VBO)
  - allows vertex array data to be stored in the device memory.
  - *GL_ARB_vertex_buffer_object*

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

OpenGL controlled memory

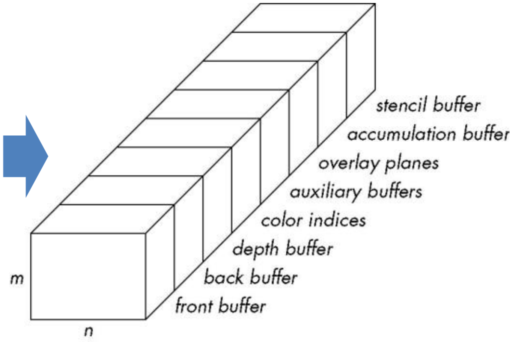


- **Frame Buffer Object (FBO)**
  - allows rendered contents (color, depth, stencil) to be stored in non-displayable framebuffers (e.g., texture object, renderbuffer object)
  - *GL_EXT_framebuffer_object*

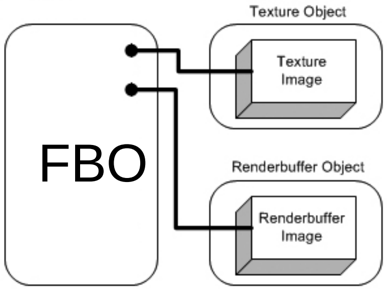


Frame Buffer Object

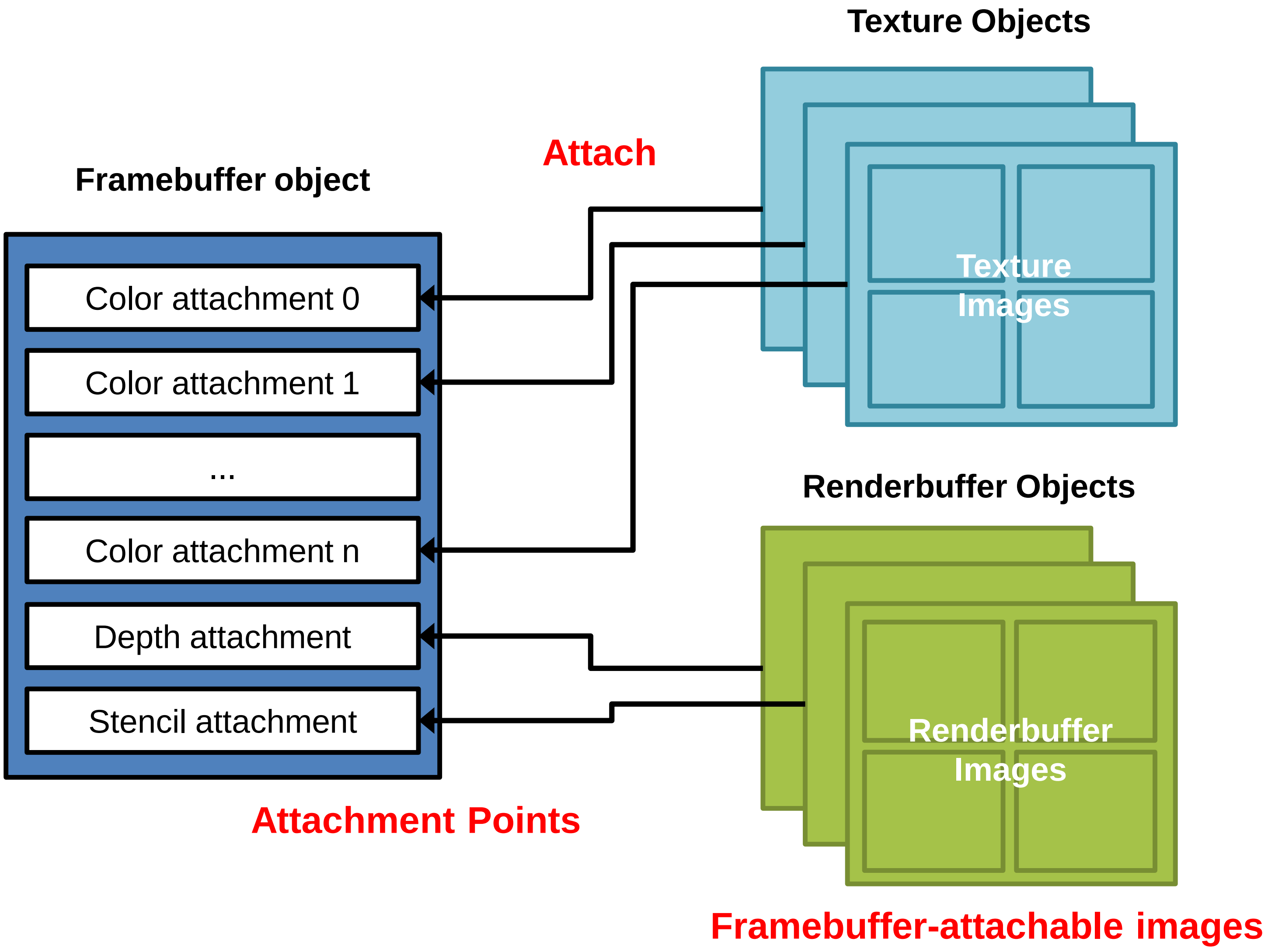
- **Framebuffer:**
  - A collection of logical buffers
    - color, depth, stencil, accumulation
  - The final rendering destination
    - *window-system-provided* framebuffer



- **Framebuffer Object**
  - A struct that holds pointers to the memory (see the next page).
  - The content stored at the memory pointed by the pointers can be framebuffer attachable images (which is also called *application-created* framebuffer).
  - GL Extension allows rendered content to be directed to the framebuffer attachable images instead of the framebuffer.
  - Framebuffer attachable images can be:
    - Textures
    - Renderbuffers (off-screen buffers)



FBO: a struct that holds pointers to the memory

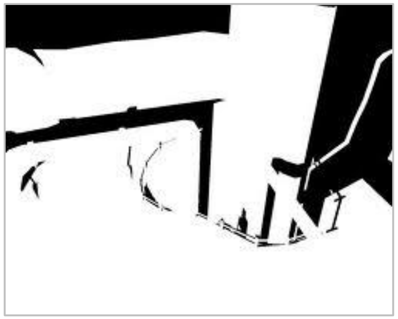


Renderbuffer Object

- **Renderbuffer**
  - It is off-screen buffer; the content is not shown.
  - Optimized only for being used as render targets.
    - No sampler, no glTexImage2d, …
  - Usually, used to store OpenGL logical buffers such as **stencil or depth buffers.**
  - The only way to use renderbuffer is to attach it to a FBO.

Attachment Points

- **To render the scene correctly, we need a collection of logical buffers.**
  - color, depth, stencil, accumulation, …



- **FBO supports color, depth, stencil attachment points.**

Why Render to Texture?

- **Allows results of rendering to framebuffer to be directly read as texture.**

- **Better performance**
  - avoids copy from framebuffer to texture (using such as glCopyTexSubImage2D)

- **More applications**
  - Dynamic textures: procedurals, reflections
  - Multi-pass techniques: anti-aliasing, motion blur, depth of field
  - Image processing effects
  - GPGPU

Usage of FBO

```
// Generate FBO ID
GLuint fboID;
glGenFramebuffer(1, &fboID);
// Bind FBO
glBindFramebuffer(GL_FRAMEBUFFER, fboID);

// ...do something with this FBO, such as
// attaching texture or renderbuffer

// unbind FBO
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Usage of FBO: Attaching Texture

```
// Generate texture
GLuint texId;
glGenTextures(1, &texID);
// Attach texture for color drawing
glFramebufferTexture2D(GL_FRAMEBUFFER_EXT,
                             GL_COLOR_ATTACHMENTn_EXT,
                             GL_TEXTURE_2D, texID, 0);
// or for depth drawing
glFramebufferTexture2D(GL_FRAMEBUFFER_EXT,
                             GL_DEPTH_ATTACHMENT_EXT,
                             GL_TEXTURE_2D, texID, 0);
```

Usage of FBO: Attaching RenderBuffer

```
// Generate renderbuffer
GLuint rbID;
glGenRenderBuffer(1, &rbID);

// Attach renderbuffer to framebuffer
glFramebufferRenderbuffer(GL_FRAMEBUFFER,
                             GL_DEPTH_ATTACHMENT,
                             GL_RENDERBUFFER,
                             rbID);
```

Usage of FBO: Check Completeness

```
if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
{
     printf("ERROR::FRAMEBUFFER:: Framebuffer is not complete!\n");

}
```

Usage of FBO: Rendering

```
glBindFramebuffer(GL_FRAMEBUFFER, fboID);

    // Rendering commands to FBO

glBindFramebuffer(GL_FRAMEBUFFER, 0); // unbind

glBindTexture(GL_TEXTURE_2D, textureID);
glGenerateMipmap(GL_TEXTURE_2D); //Generating Mipmap
glBindTexture(GL_TEXTURE_2D, 0);


// Rendering commands to Screen
```

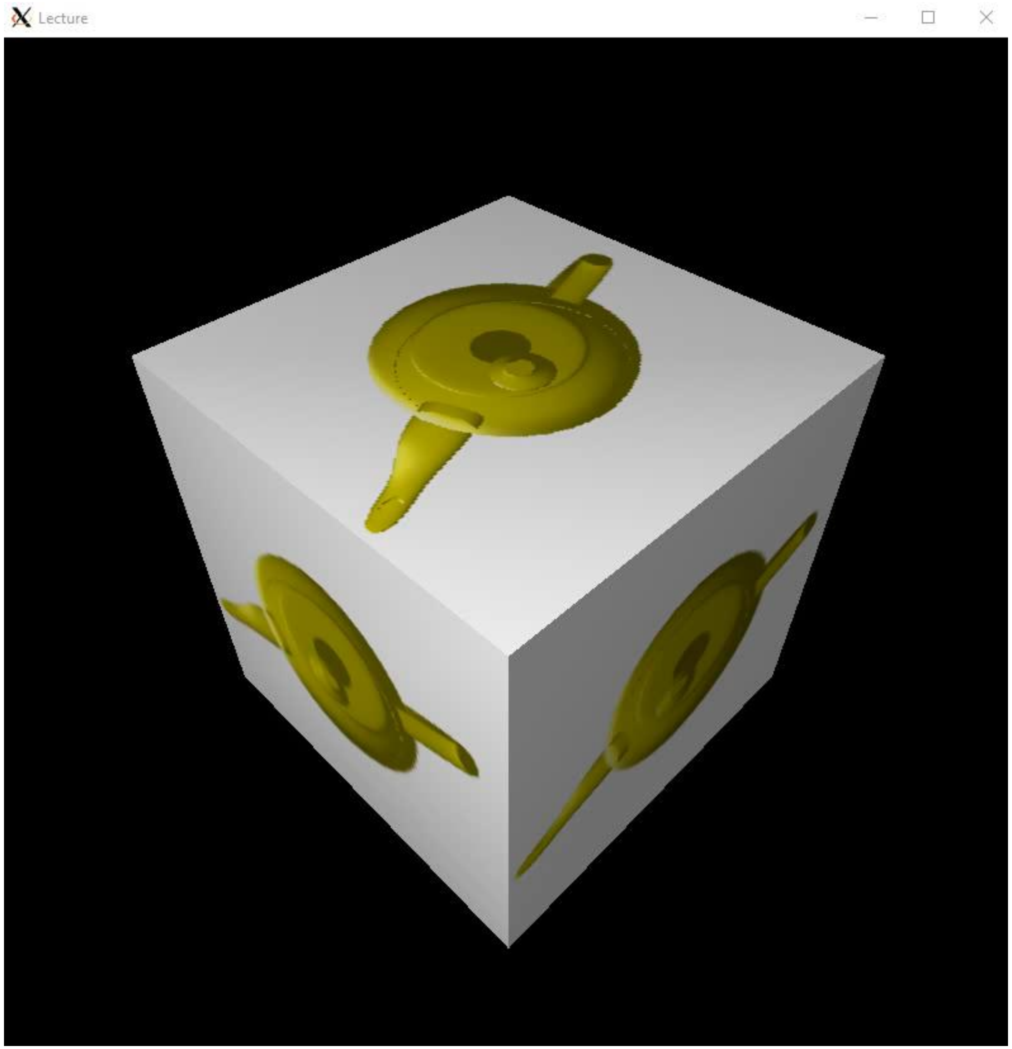
PBO Coding Exercise

- **Copy Sample Skeleton Code**
  - vglconnect ID@163.152.20.246
  - cp –r /home/share/15_FBO ./[Folder name]
  - cd [Folder name]

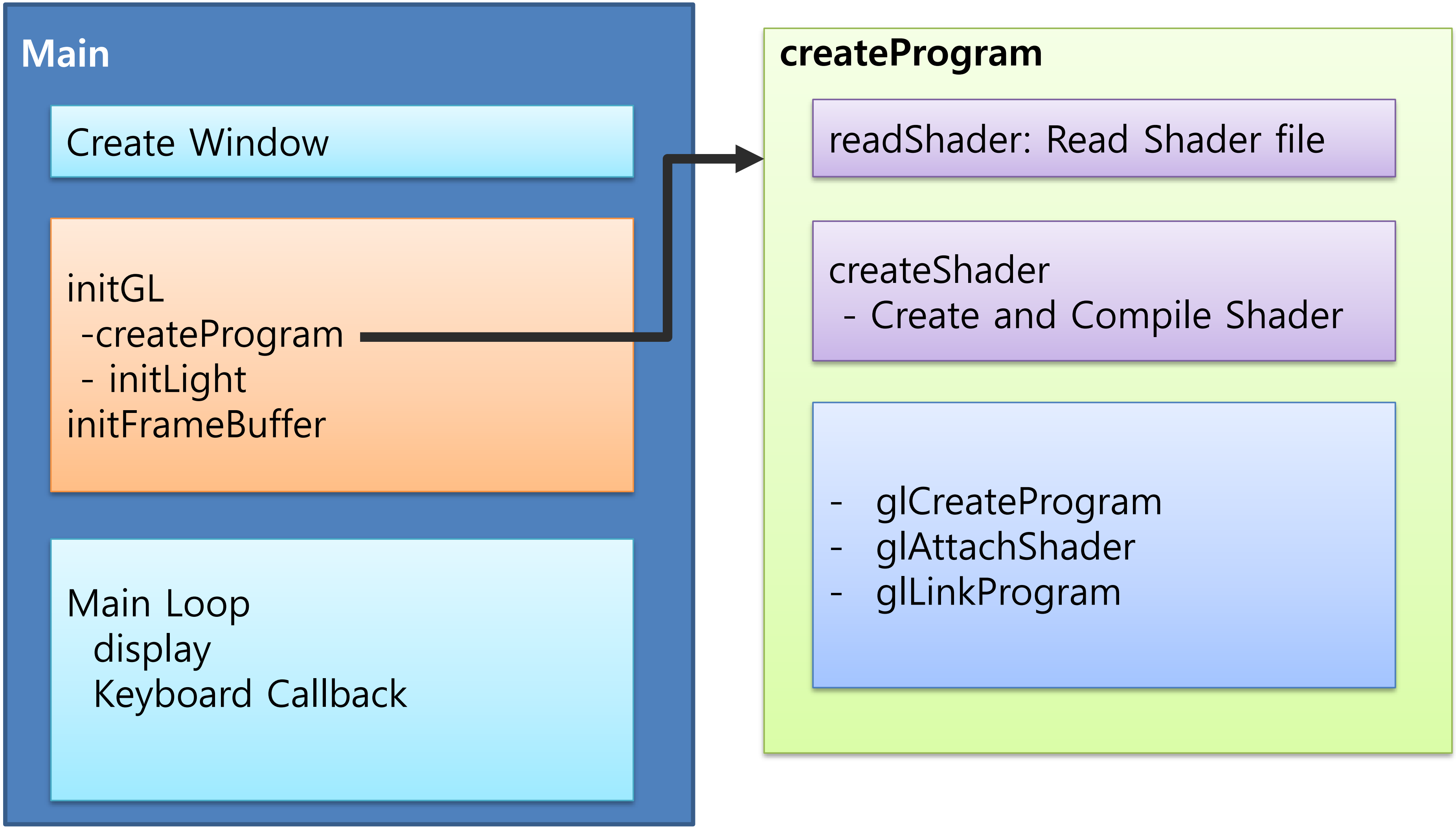
- **Notepad: Shader code 수정**

- **Compile program**
  - make
  - vglrun ./EXE

Expected Result



Program Flow



Main

Create Window

initGL
 -createProgram
 - initLight
initFrameBuffer

Main Loop
 display
 Keyboard Callback

createProgram

readShader: Read Shader file

createShader
 - Create and Compile Shader

- glCreateProgram
- glAttachShader
- glLinkProgram

Program structure

```
#include headers
using namespace std;
//function declaration//
//Global variables//
float *vertices;
float *textCoord;
Float *normals;
int main(int argc, char *argv[]) {

    //Window Initialization//
    initGL();
    initFrameBuffer();

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

initGL()

```
void initGL(){
        glewInit();
        createProgram();
        initLights();

        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
        glEnable(GL_TEXTURE_2D);
        glEnable(GL_CULL_FACE);
        glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
        glEnable(GL_COLOR_MATERIAL);
}
```

initLight @ initGL()

```
void initLight(){
    //set up light colors (ambient, diffuse, specular)
    GLfloat lightKa[] = {0.2f, 0.2f, 0.2f, 1.0f};  // ambient light
    GLfloat lightKd[] = {0.7f, 0.7f, 0.7f, 1.0f};  // diffuse light
    GLfloat lightKs[] = {1.0f, 1.0f, 1.0f, 1.0f};  // specular light
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightKa);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightKd);
    glLightfv(GL_LIGHT0, GL_SPECULAR, lightKs);

    //position the light
    float lightPos0[4] = {-0.7, 0, -0.7, 1}; // positional light
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);
    glEnable(GL_LIGHT0);// MUST enable each light source after configuration

}
```

createProgram@ initGL()

```
void createProgram(){
        char *vertexShaderSource = ReadFile("Vertex.glsl");
        char *fragmentShaderSource = ReadFile("Fragment.glsl");
        unsigned int VertShader = createShader(vertexShaderSource, GL_VERTEX_SHADER);
        unsigned int FragShader = createShader(fragmentShaderSource,
GL_FRAGMENT_SHADER);

        Program = glCreateProgram();
        glAttachShader(Program, VertShader);
        glAttachShader(Program, FragShader);
        glLinkProgram(Program);
}
```

initFrameBuffers()@main

```
void initFrameBuffers(){
    glGenFramebuffers(1, &fboID); //Generating FBO
    glBindFramebuffer(GL_FRAMEBUFFER, fboID); //Binding FBO

    glGenTextures(1, &textureID); //Generating Texture
    glBindTexture(GL_TEXTURE_2D, textureID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
    GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP, GL_TRUE);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, TEXTURE_WIDTH, TEXTURE_HEIGHT,
    0, GL_RGBA, GL_UNSIGNED_BYTE, 0);
    glBindTexture(GL_TEXTURE_2D, 0); //Unbinding FBO
```

initFrameBuffers()Main –Cont.

```
    glGenRenderbuffers(1, &rboDepthID); //Generating RBO
    glBindRenderbuffer(GL_RENDERBUFFER, rboDepthID); //Binding RBO
    glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT,
    TEXTURE_WIDTH, TEXTURE_HEIGHT);
    glBindRenderbuffer(GL_RENDERBUFFER, 0); //Unbinding RBO

    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
    GL_TEXTURE_2D, textureID, 0); //Attaching Texture to FBO
    glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
    GL_RENDERBUFFER, rboDepthID); //Attaching RBO to FBO
    if (glCheckFramebufferStatus(GL_FRAMEBUFFER) !=
    GL_FRAMEBUFFER_COMPLETE){
        cout << "ERROR::FRAMEBUFFER:: Framebuffer is not complete!" << endl;
    }

    glBindFramebuffer(GL_FRAMEBUFFER, 0); //Unbinding FBO
}
```

Display function@Main –Cont.

```
void display() {
        int playTime = glutGet(GLUT_ELAPSED_TIME); //Get time
        const float ANGLE_SPEED = 5; //Angular velocity
        float angle = ANGLE_SPEED * playTime;
```

Drawing to FBO

```
        glBindFramebuffer(GL_FRAMEBUFFER, fboId);// set the rendering destination to FBO
        glViewport(0, 0, TEXTURE_WIDTH, TEXTURE_HEIGHT);
        glUseProgram(program);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(60.0f, (float)(TEXTURE_WIDTH)/TEXTURE_HEIGHT, 1.0f, 100.0f);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glTranslatef(0, 0, -3.2); glClearColor(1, 1, 1, 1);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotatef(angle*0.5f, 1, 0, 0); glRotatef(angle, 0, 1, 0); glRotatef(angle*0.7f, 0, 0, 1);
        glutSolidTeapot(1.0f); //Drawing Teapot
        glPopMatrix();
        glUseProgram(0);
        glBindFramebuffer(GL_FRAMEBUFFER, 0); // unbind
```

Display function@Main

```
    glBindTexture(GL_TEXTURE_2D, textureID);
    glGenerateMipmap(GL_TEXTURE_2D); //Generating Mipmap
    glBindTexture(GL_TEXTURE_2D, 0);

    // back to normal viewport and projection matrix
```

Drawing to screen

```
    glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0f, (float)(SCREEN_WIDTH)/SCREEN_HEIGHT, 1.0f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0, 0, -4);
    glRotatef(45.0f, 1, 0, 0);   // pitch
    glRotatef(45.0f, 0, 1, 0);   // heading
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
    glPushMatrix();
    drawCube(); //Drawing Cube function
    glPopMatrix();
    glXSwapBuffers(dpy, win);
}
```

drawingCube()@Display

```
void draw(){
        glBindTexture(GL_TEXTURE_2D, textureID); //Binding texture which is attached to FBO
        glColor4f(1, 1, 1, 1);
        glBegin(GL_TRIANGLES);
        // Front faces
        glNormal3f(0, 0, 1);
        // face v0-v1-v2
        glTexCoord2f(1, 1);  glVertex3f(1, 1, 1);
        glTexCoord2f(0, 1);  glVertex3f(-1, 1, 1);
        glTexCoord2f(0, 0);  glVertex3f(-1, -1, 1);
        // face v2-v3-v0
        glTexCoord2f(0, 0);  glVertex3f(-1, -1, 1);
        glTexCoord2f(1, 0);  glVertex3f(1, -1, 1);
        glTexCoord2f(1, 1);  glVertex3f(1, 1, 1);

        //Drawing Other faces
        ......
        glEnd();
        glBindTexture(GL_TEXTURE_2D, 0); //Detach the texture
}
```

Shader code : Vertex shader

```
//Vertex Shader code
#version 130

out vec3 normal, lightDir, halfVector;
void main() {
        normal = normalize(gl_NormalMatrix*gl_Normal);
        lightDir = normalize(gl_LightSource[0].position.xyz);
        halfVector = normalize(gl_LightSource[0].halfVector.xyz);
        gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

Shader code : Fragment shader

```
//Fragment Shader code
#version 130

in vec3 normal, lightDir, halfVector;
vec4 matKa = vec4(1.0f, 1.0f, 0.0f, 1.0f);
vec4 matKd = vec4(0.6f, 0.6f, 0.0f, 1.0f);
vec4 matKs = vec4(1.0f, 1.0f, 1.0f, 1.0f);
void main() {
        vec3 n, h;
        float NdotL, NdotH;
        vec4 color = matKa * gl_LightSource[0].ambient + matKa * gl_LightModel.ambient;
        n = normalize(normal);
        NdotL = max(dot(n,lightDir),0.0);
        if (NdotL > 0.0) {
                color += matKd * gl_LightSource[0].diffuse * NdotL;
                h = normalize(halfVector);
                NdotH = max(dot(n,h),0.0);
                color += matKs * gl_LightSource[0].specular * pow(NdotH, 5);
        }
        gl_FragColor = color;
}
```