Chapter 12:
PBO & FBO

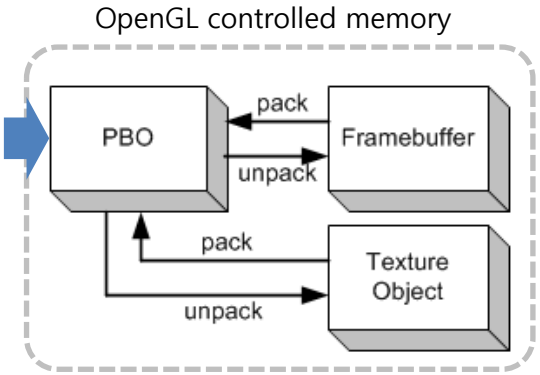
Graphics Programming, 13th Oct.

Graphics and Media Lab.
Seoul National University 2011 Fall

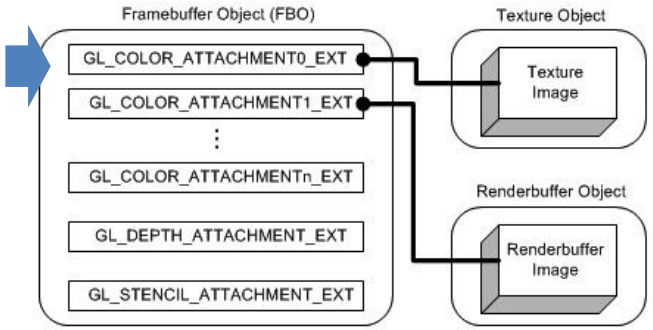
Abstract Buffer Objects

- 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*



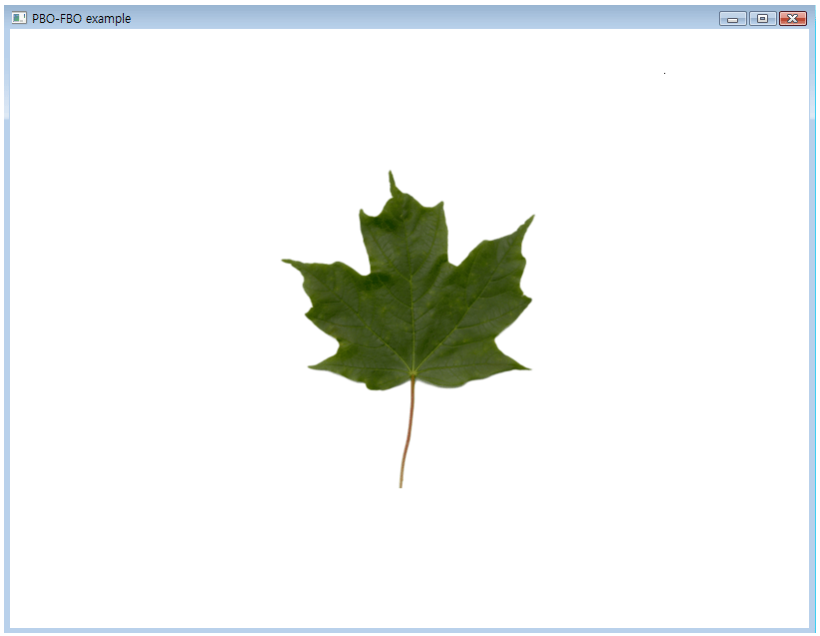
- Frame Buffer Object (FBO)
  - allows rendered contents (color, depth, stencil) to be stored in non-displayable framebuffers (e.g., texture object, renderbuffer object)
  - *GL_EXT_framebuffer_object*

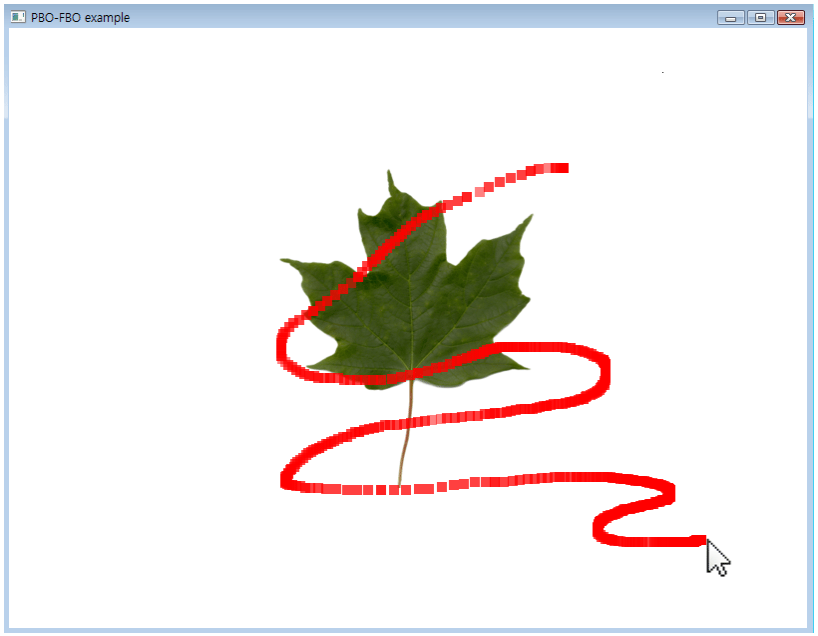


Pixel Buffer Object

- 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

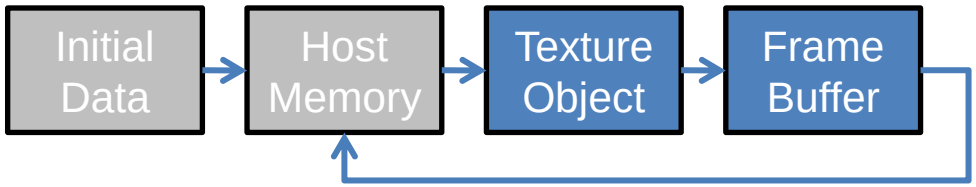
Example – Sketch Program

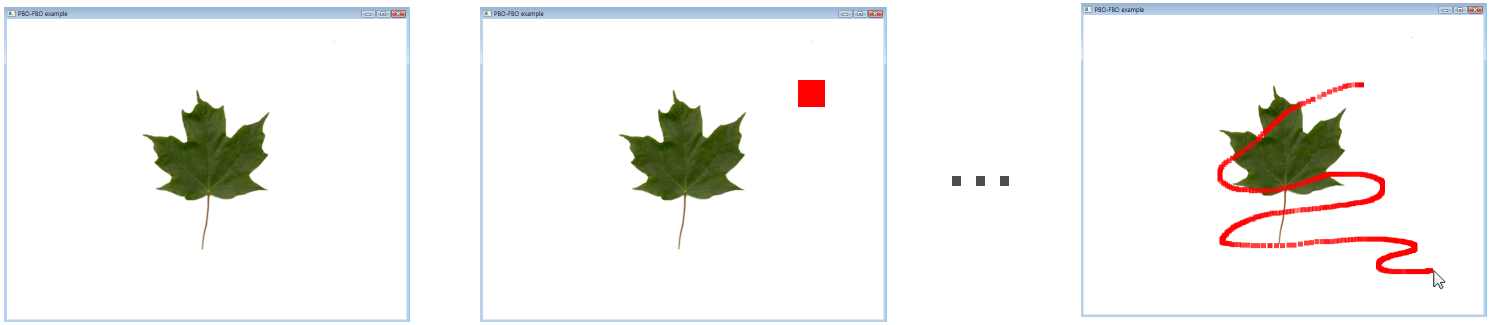




How to Implement it?

```
while(1) {
    Draw a textured rectangle (to framebuffer);
    Draw by blending the red square at the current mouse
    position (to framebuffer);
    Read pixels from the framebuffer (to CPU array);
    Use the read pixels to update the texture;
}
```



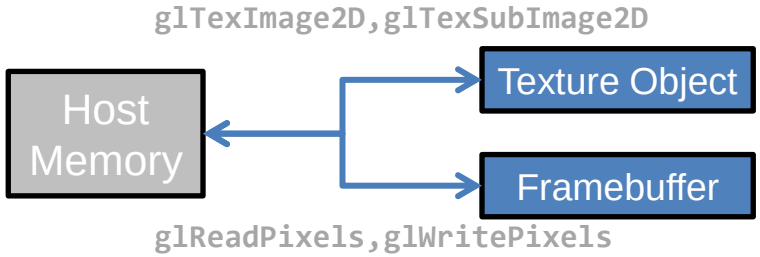


The Code w/o PBO or FBO

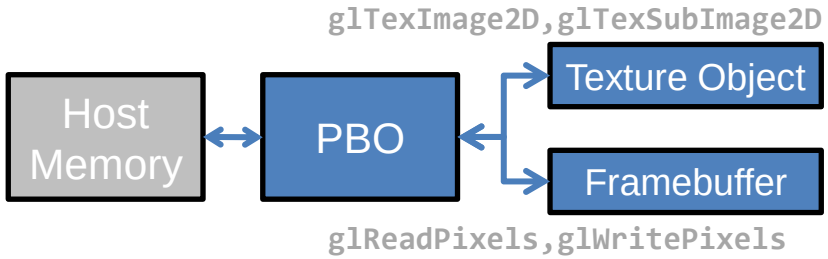
```
if(g_pboMode == 0) {
  glPixelStorei(GL_PACK_ALIGNMENT, 4);
  glReadPixels(0,0, g_winWidth,g_winHeight, PIXEL_FORMAT, PIXEL_TYPE, g_imageData);
  // reading framebuffer content to host memory
  glBindTexture(GL_TEXTURE_2D, g_texId);
  glTexSubImage2D(GL_TEXTURE_2D, 0, 0,0, g_winWidth, g_winHeight, PIXEL_FORMAT,
                  PIXEL_TYPE, g_imageData);
  glBindTexture(GL_TEXTURE_2D, 0);
}
```

Speeding up Pixel Data Transfer with PBO

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

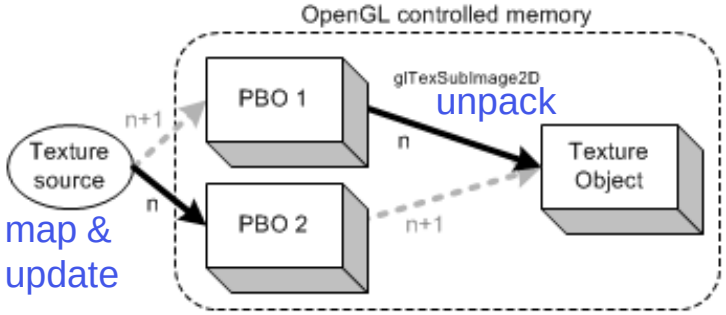


  - Using PBO

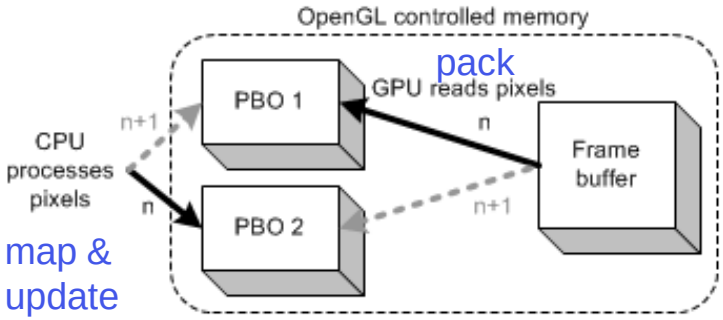


Use of Multiple PBOs

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

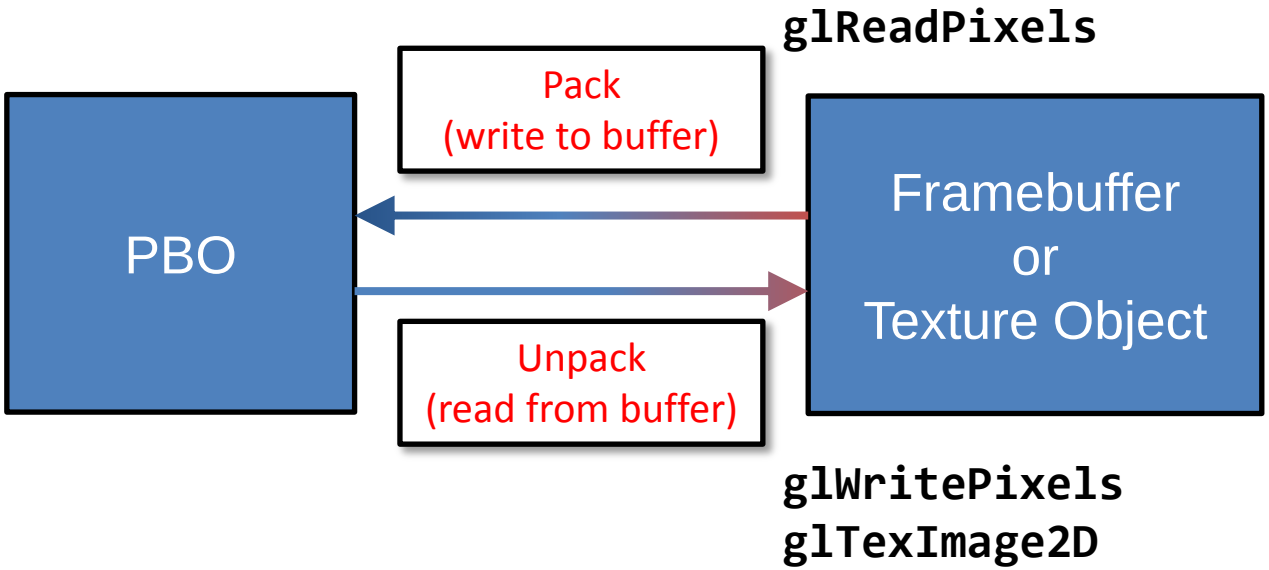


  - ex. Asynchronous read-back



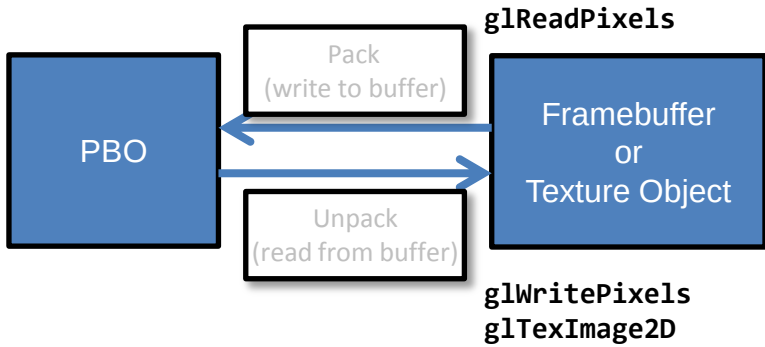
Usage of PBO

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



Usage of PBO

- PBO has two targets:
  - GL_PIXEL_PACK_BUFFER
  - GL_PIXEL_UNPACK_BUFFER

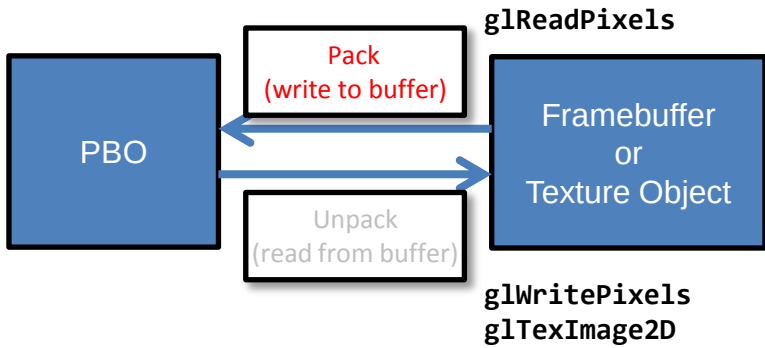


- Usage: Create & Delete

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

Usage of PBO

- PBO has two targets:
  - GL_PIXEL_PACK_BUFFER
  - GL_PIXEL_UNPACK_BUFFER

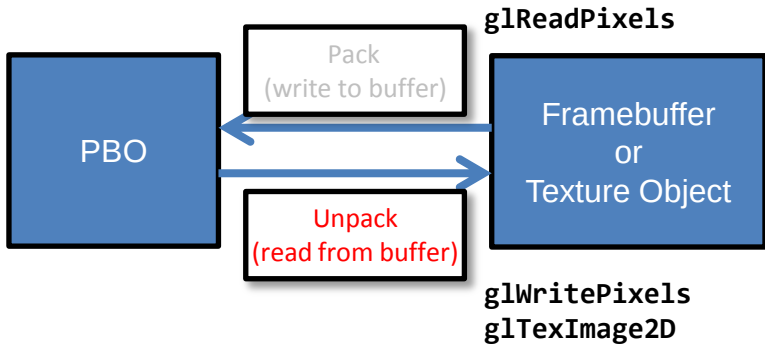


- Usage: PBO for reading

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

Usage of PBO

- PBO has two targets:
  - GL_PIXEL_PACK_BUFFER
  - GL_PIXEL_UNPACK_BUFFER



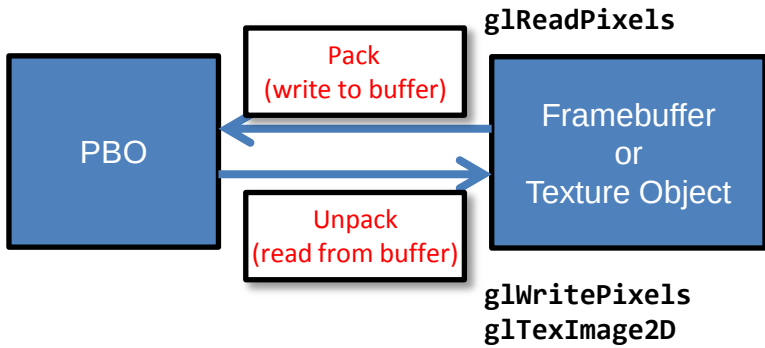
- Usage: PBO for writing

```
// 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_BGRA,
                GL_UNSIGNED_BYTE, 0);
```

With the current pbo context, 0 means the start of the pbo.

Usage of PBO

- PBO has two targets:
  - GL_PIXEL_PACK_BUFFER
  - GL_PIXEL_UNPACK_BUFFER



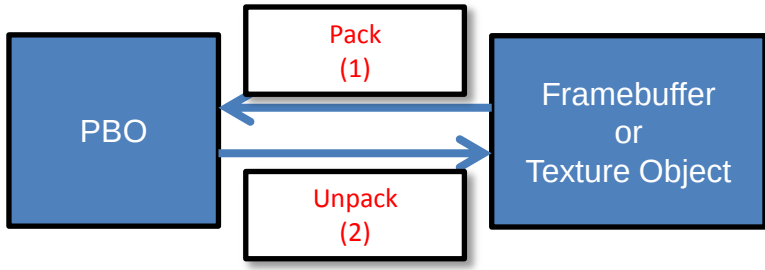
- Usage: Update PBO

```
// 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);
}
```

The Code with PBO

```
bool initMemory() {
  …
  // PBO
  if(g_pboSupported) {
    glGenBuffersARB(2, g_pboIds); Two pbos are created
    glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, g_pboIds[0]); Pbo0 is for pack
    // glBufferDataARB with NULL pointer only reserves the memory space.
    glBufferDataARB(GL_PIXEL_PACK_BUFFER_ARB, DATA_SIZE, 0, GL_STREAM_READ_ARB);
    glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0); Creation of Pbo0 is complete
    glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, g_pboIds[1]);
    glBufferDataARB(GL_PIXEL_UNPACK_BUFFER_ARB, DATA_SIZE, 0, GL_STREAM_DRAW_ARB);
    glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, 0);
  }
```

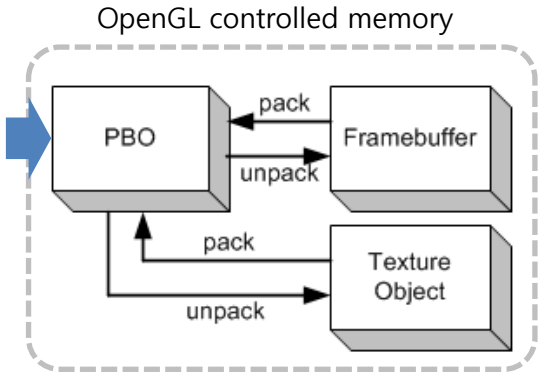
The Code with PBO

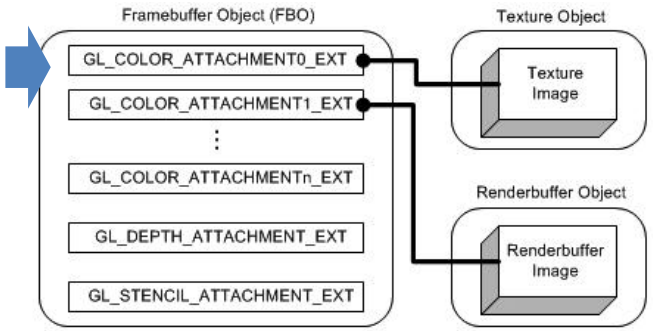



```
void callback_display() {
  Blend the current square on top of the previous texture content;
  if (g_pboMode == 1) {
      glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, g_pboIds[index]);
      glReadPixels(0,0, g_winWidth, g_winHeight, PIXEL_FORMAT, PIXEL_TYPE, 0);
      // (1) reading framebuffer content to pbo0
      glBindTexture(GL_TEXTURE_2D, g_texId);
      glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, g_pboIds[index]);
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0,0, g_winWidth, g_winHeight,
                              PIXEL_FORMAT, PIXEL_TYPE, 0);
      // (2) writing pbo0 content to texture object
      glBindTexture(GL_TEXTURE_2D, 0);
      glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
      glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, 0);
  }
}
// With FBO, the framebuffer content can be written directly to the texture obj
```

Abstract Buffer Objects

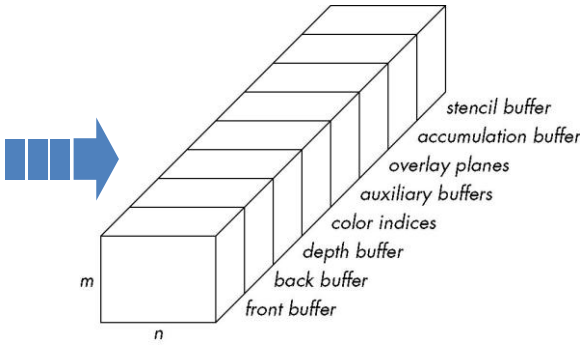
- Vertex Buffer Object (VBO)
  - allows vertex array data to be stored in the device memory.
  - *GL_ARB_vertex_buffer_object*
- Pixel Buffer Object (PBO)
  - allows pixel data to be stored in the device memory for further intra-GPU transfer
  - *GL_ARB_pixel_buffer_object*
- Frame Buffer Object (FBO)
  - allows rendered contents (color, depth, stencil) to be stored in non-displayable framebuffers (e.g., texture object, renderbuffer object)
  - *GL_EXT_framebuffer_object*



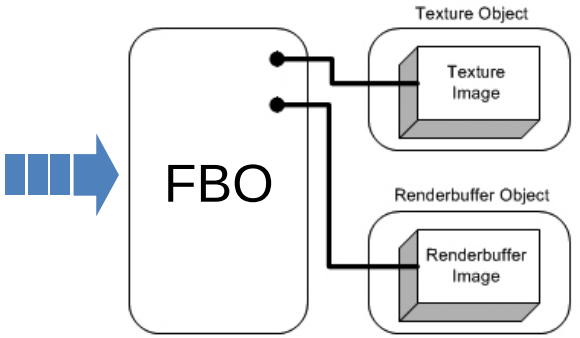


Frame Buffer Object

- 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.
  - 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)

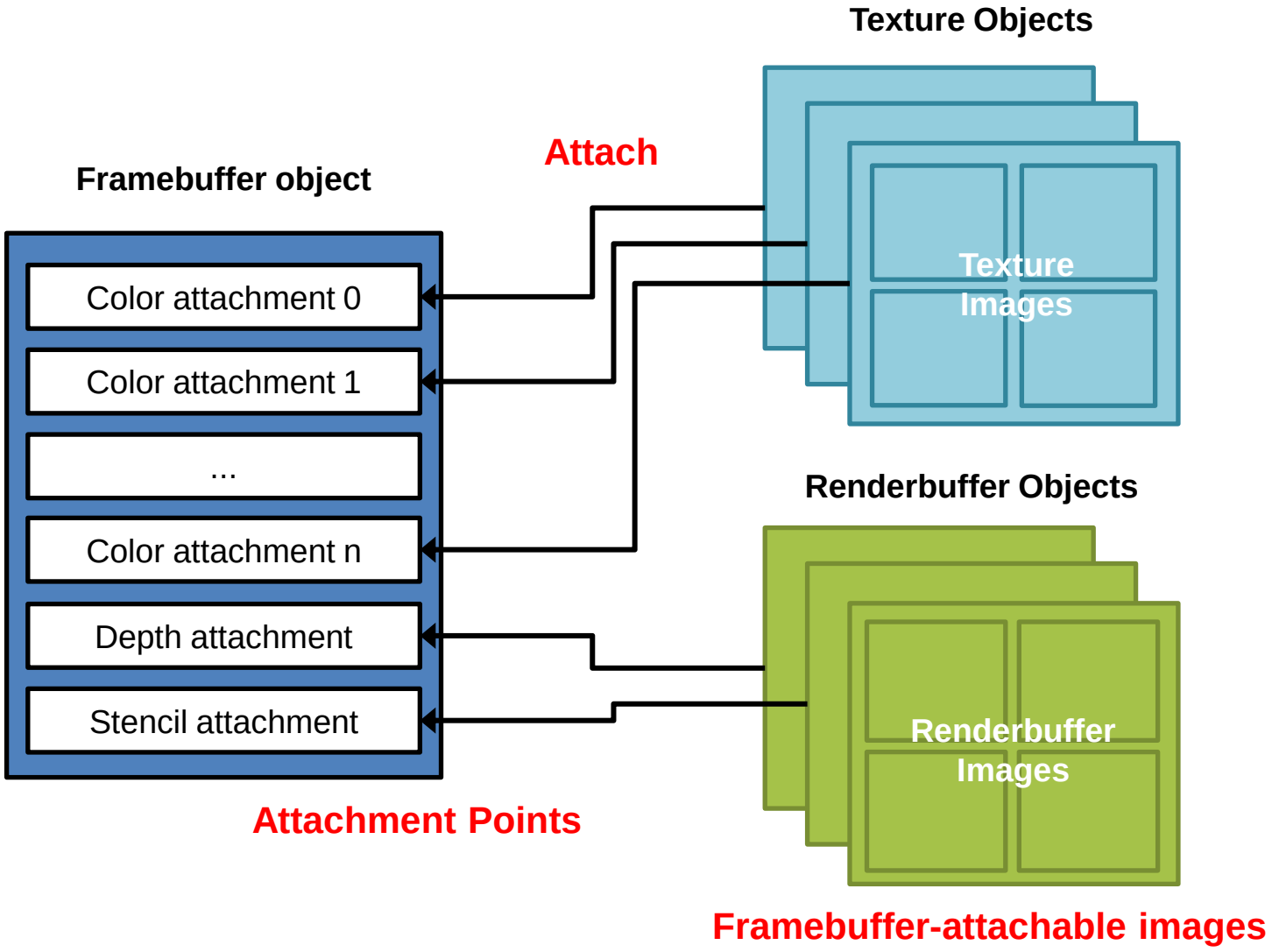


Attachment Points

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

- FBO supports color, depth, stencil attachment points.

FBO Architecture



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)

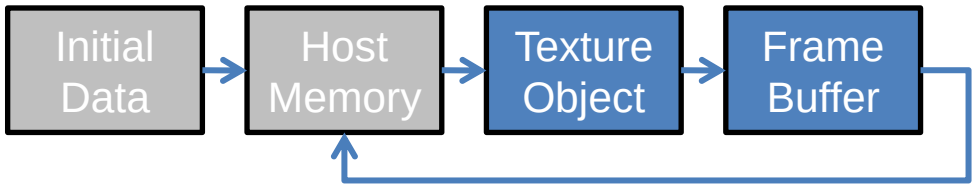
- More applications
  - Dynamic textures: procedurals, reflections
  - Multi-pass techniques: anti-aliasing, motion blur, depth of field
  - Image processing effects
  - GPGPU

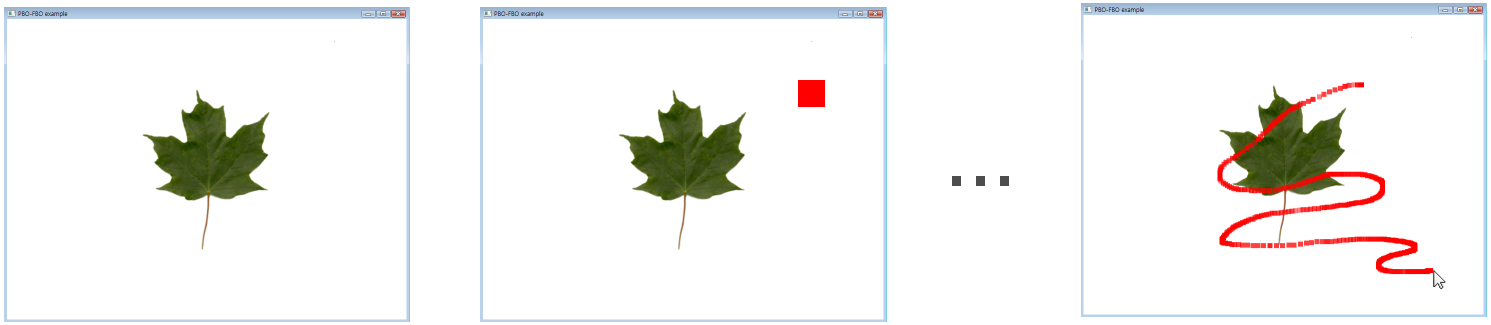
Renderbuffer Object

- Renderbuffer
  - 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.

Sketch Program – Without PBO/FBO

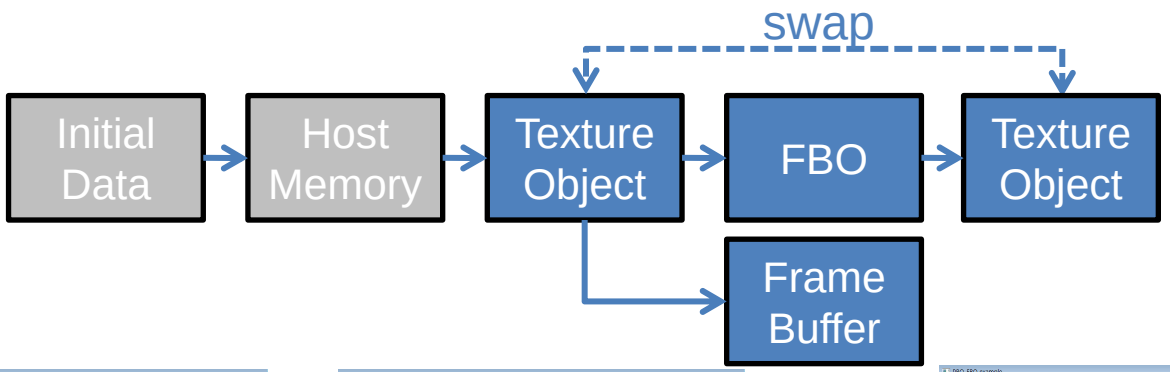
```
while(1) {
    Draw a textured rectangle (to framebuffer);
    Draw by blending the red square at the current mouse
    position (to framebuffer);
    Read pixels from the framebuffer (to CPU array);
    Use the read pixels to update the texture;
}
```

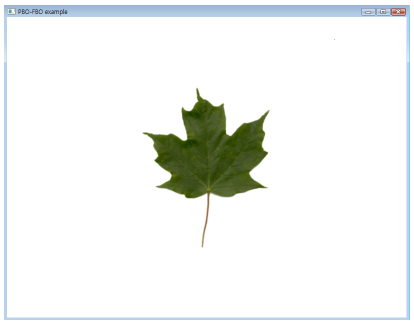


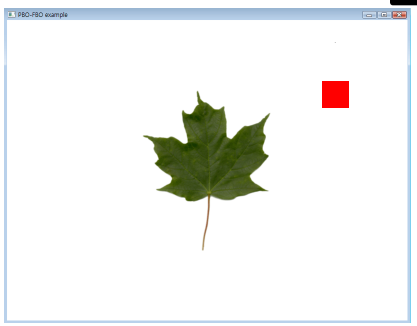


Sketch Program – With FBO

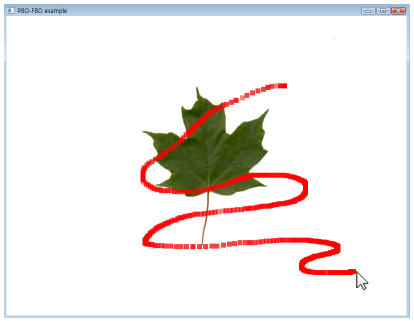
```
while(1) {
    Draw a textured rectangle (to fbo & framebuffer);
    Draw by blending the red square at the current mouse
    position (to fbo);
    Read pixels from the framebuffer (to CPU array);
    Use the read pixels to update the texture;
}
```







...



Initializing FBO

```
// Generate FBO ID
GLuint fboID;
glGenFramebufferEXT(1, &fboID);
// Bind FBO
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fboID);

// ...do something with this FBO

// unbind FBO
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```

Attach Texture Image to FBO

```
// Generate texture
GLuint texId;
glGenTextures(1, &texID);
// Attach texture for color drawing
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
                          GL_COLOR_ATTACHMENTn_EXT,
                          GL_TEXTURE_2D, texID, 0);
// or for depth drawing
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
                          GL_DEPTH_ATTACHMENT_EXT,
                          GL_TEXTURE_2D, texID, 0);
```

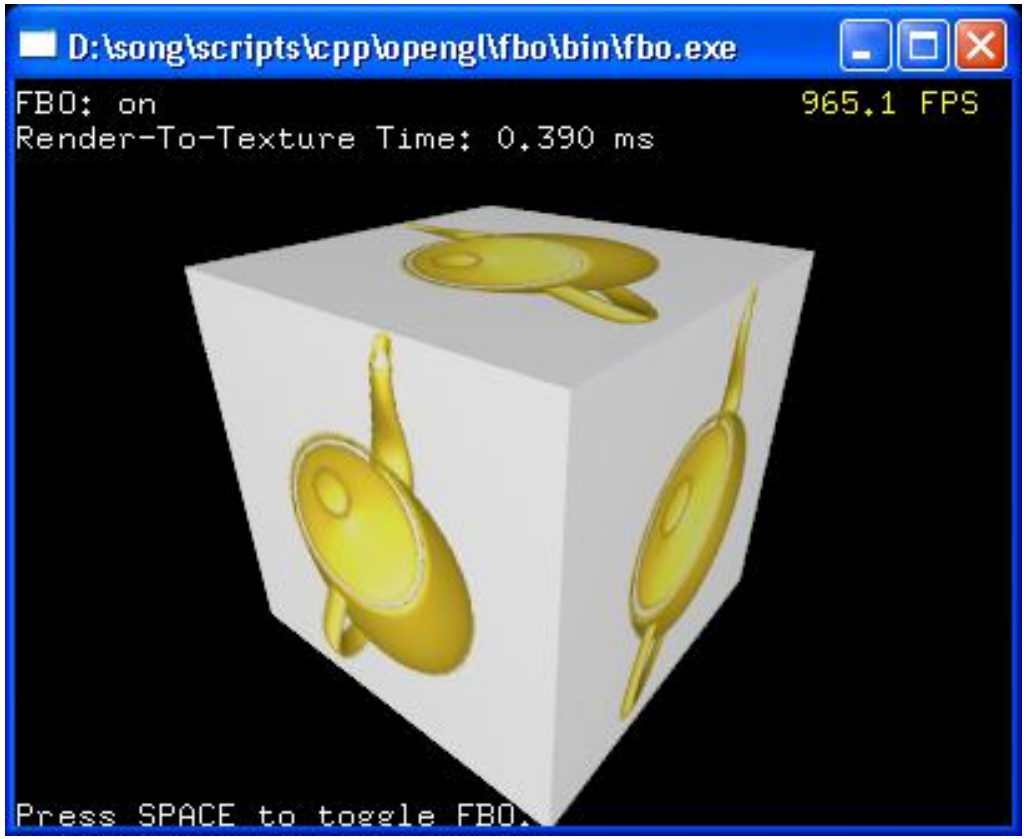
Attach Renderbuffer to FBO

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

// Attach renderbuffer to framebuffer
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT,
                             GL_DEPTH_ATTACHMENT_EXT,
                             GL_RENDERBUFFER_EXT,
                             rbID);
```

Check Completeness of FBO

```
// Get error status
Glenum status;
status = glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT);
// Check the status
switch(status) {
case GL_FRAMEBUFFER_COMPLETE_EXT: {... break;}
case GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT_EXT:
case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT_EXT:
case GL_FRAMEBUFFER_INCOMPLETE_DIMENSIONS_EXT:
case GL_FRAMEBUFFER_INCOMPLETE_FORMATS_EXT:
case GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER_EXT:
case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER_EXT:
Case GL_FRAMEBUFFER_UNSUPPORTED_EXT:
...
}
```

A Small Project with FBO



Normal Rendering of Teapot

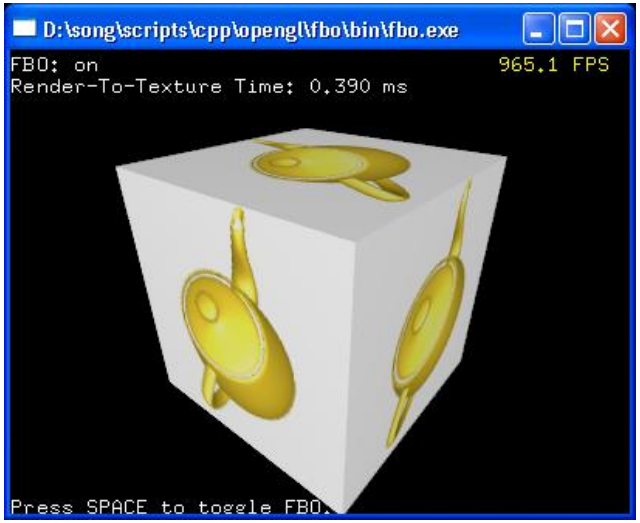
```
void display() {
    glClear(...);
    glViewport(...);
    applyTransform();
    glutSolidTeapot(...);
    glFlush();
    glutSwapBuffers();
}
```

Rendering it to FBO

```
void display() {
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fboID);
    glClear(...);
    glViewport(...);
    applyTransform();
    glutSolidTeapot(...);
    glFlush();
    glutSwapBuffers();
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
}
```

Drawing a Cube with Attached Texture

```
void display() {
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fboID);
    ...
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    glClear(...);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texID);
    glBegin(...);
    ...
    glEnd();
    glDisable(GL_TEXTURE_2D);
}
```



Any Questions ?