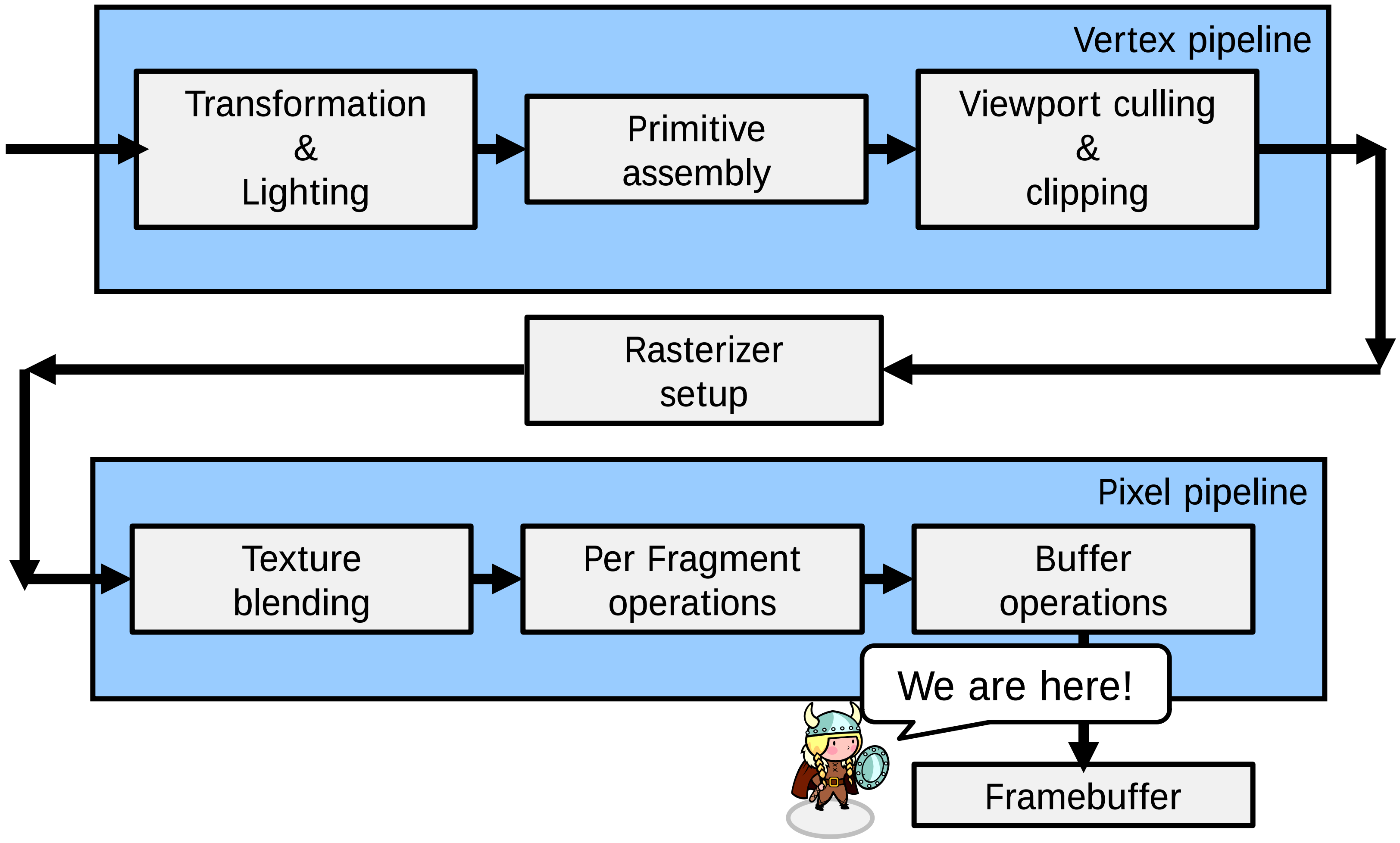
OpenGL Framebuffer Object Extension

Motivation

- Render-to-texture
  - Allow results of rendering to framebuffer to be directly read as texture
- Better performance than glCopyTexSubImage
- Application
  - Dynamic textures, GPGPU, …

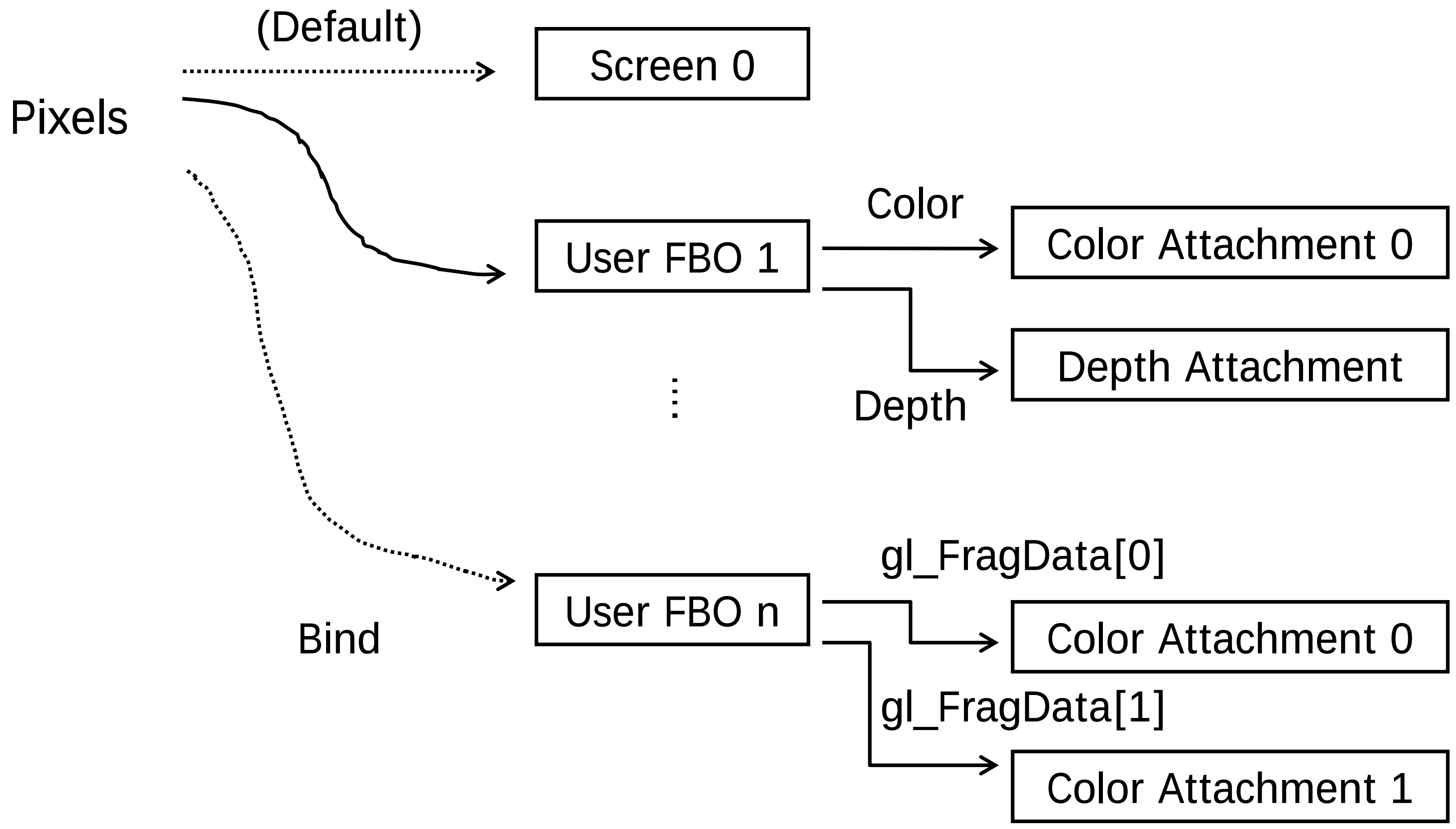
Problem position



EXT_framebuffer_object

- A collection of logical buffers
  - Color, depth, stencil
  - Provides a new mechanism for rendering to destinations other than those provided by window system
  - Framebuffer-attachable images
    - Off-screen buffers (Renderbuffers)
    - textures

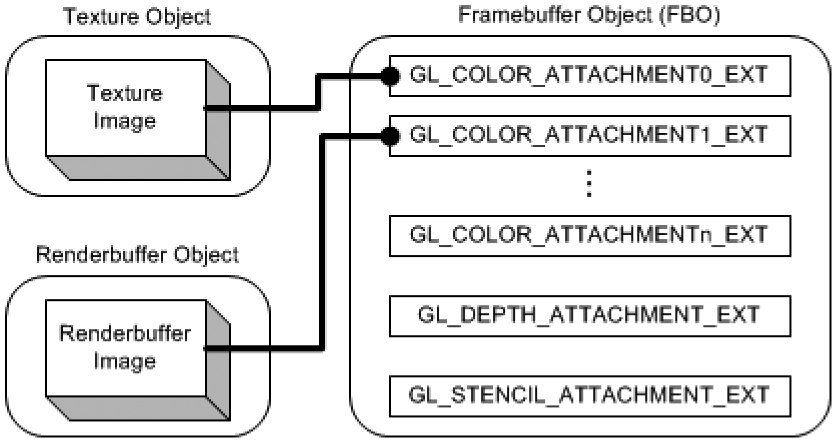
Replace the Screen



Idea (0/2) (ref)

- Similar to *window-system-provided* framebuffer, a FBO contains a collection of rendering destinations; color, depth and stencil buffer. *(Note that accumulation buffer is not defined in FBO.)*
- These logical buffers in a FBO are called *framebuffer-attachable images*, which are 2D arrays of pixels that can be attached to a framebuffer object.

Idea (1/2) FBO



- There are two types of framebuffer-attachable images; texture images and renderbuffer images.
  - If an image of a texture object is attached to a framebuffer, OpenGL performs *"render to texture"*.
  - If an image of a renderbuffer object is attached to a framebuffer, then OpenGL performs *"offscreen rendering"*.

Idea (2/2) Renderbuffer

- By the way, renderbuffer object is a new type of storage object defined in GL_EXT_framebuffer_object extension. It is used as a rendering destination for a single 2D image during rendering process.

Functions and Attachment

- Create and Destroy FBO
  - void glGenFramebuffersEXT(GLsizei n, GLuint* ids)
  - void glDeleteFramebuffersEXT(GLsizei n, const GLuint* ids)
- Bind FBO
  - void glBindFramebufferEXT(GLenum target, GLuint id)
- Texture
- Render Buffer
  - You cant send it to shader
  - It supports some format that texture doesn’t.

Render Buffer

- Create
  - void glGenRenderbuffersEXT(GLsizei n, GLuint* ids);
- Destroy
  - void glDeleteRenderbuffersEXT(GLsizei n, const Gluint* ids);
- Bind
  - void glBindRenderbufferEXT(GLenum target, GLuint id);
- Storage
  - void glRenderbufferStorageEXT(GLenum target, GLenum internalFormat, GLsizei width, GLsizei height)

FBO and RB

- Framebuffer Object (FBO)
  - Collection of framebuffer-attachable images (color, depth, stencil)

- Renderbuffer (RB)
  - Contain a simple 2D image (no mipmap, …)
  - Store pixel data resulting from rendering
  - Cannot be used as textures!

Setting Up FBO

```
// Setup our FBO
glGenFramebuffersEXT(1, &fbo);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);

// Create the render buffer for depth
glGenRenderbuffersEXT(1, &depthBuffer);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depthBuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, width, height);

// Now setup a texture to render to
glGenTextures(1, &img);
glBindTexture(GL_TEXTURE_2D, img);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8,  width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// And attach it to the FBO so we can render to it
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, img, 0);

// Attach the depth render buffer to the FBO as it's depth attachment
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, depthBuffer);

GLenum status = glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT);
if(status != GL_FRAMEBUFFER_COMPLETE_EXT)
    exit(1);

glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);    // Unbind the FBO for now
```

Using FBO$_{1/2}$

```
void display(void)
{
    // First we bind the FBO so we can render to it
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);

    // Save the view port and set it to the size of the texture
    glPushAttrib(GL_VIEWPORT_BIT);
    glViewport(0,0,width,height);

    // Then render as normal
    glClearColor(0.0f, 0.0f, 0.0f, 0.5f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear Screen And Depth Buffer
    glLoadIdentity();

    // rendering (to FBO) ...

    // Restore old view port and set rendering back to default frame buffer
    glPopAttrib();
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```

Using FBO$_{2/2}$

```
    glClearColor(0.4f, 0.4f, 0.4f, 0.5f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear Screen And Depth Buffer
    glLoadIdentity();

    // Now bind the texture to use it
    glBindTexture(GL_TEXTURE_2D, img);
    glEnable(GL_TEXTURE_2D);

    // render with the FBO texture ...

    glDisable(GL_TEXTURE_2D);

    glutSwapBuffers ( );
    // Swap The Buffers To Not Be Left With A Clear Screen
}
```