Create the Framebuffer Object (FBO)

- ☐ Generate FBO descriptor using `glGenFramebuffers()`
- ☐ Bind `fbod` to be active using `glBindFramebuffer()` and `GL_FRAMEBUFFER`

Attach Renderbuffer Objects (RBO)

- ☐ Generate RBO descriptors using `glGenRenderbuffers()`
- ☐ Attach Depth Buffer Object (DBO)
  - ☐ Bind renderbuffer using glBindRenderbuffer()
  - ☐ Allocate memory for depth buffer using `glRenderbufferStorage()` using internal format of `GL_DEPTH_COMPONENT`
  - ☐ Attach renderbuffer to framebuffer using `glFramebufferRenderbuffer()` `GL_FRAMEBUFFER`, `GL_DEPTH_ATTACHMENT`, `GL_RENDERBUFFER`, `dbod`
- ☐ [Optional] Attach Stencil Buffer Object (SBO)
  - ☐ Bind renderbuffer
  - ☐ Allocate memory for depth buffer using `glRenderbufferStorage()` using internal format of `GL_STENCIL_INDEX`
  - ☐ Attach renderbuffer to framebuffer using `glFramebufferRenderbuffer()` `GL_FRAMEBUFFER`, `GL_STENCIL_ATTACHMENT`, `GL_RENDERBUFFER`, `sbod`
- ☐ [Optional] Attach Depth and Stencil Buffer Objects together at once (DSBO)
  - ☐ Bind renderbuffer
  - ☐ Allocate memory for depth buffer using `glRenderbufferStorage()` using internal format of `GL_DEPTH_STENCIL`
  - ☐ Attach renderbuffer to framebuffer using `glFramebufferRenderbuffer()` `GL_FRAMEBUFFER`, `GL_DEPTH_STENCIL_ATTACHMENT`, `GL_RENDERBUFFER`, `dsbod`
- ☐ [Optional] Attach additional Color Buffer Objects (CBO)
  - ☐ Bind renderbuffer
  - ☐ Allocate memory for depth buffer using `glRenderbufferStorage()` using appropriate internal format `GL_RGB`, `GL_RGBA`, etc.
  - ☐ Attach renderbuffer to framebuffer using `glFramebufferRenderbuffer()` `GL_FRAMEBUFFER`, `GL_COLOR_ATTACHMENT`*i*, `GL_RENDERBUFFER`, `cbod`

Attach Texture Image to FBO

- ☐ Generate texture handle using `glGenTextures()`
- ☐ Bind texture handle to be active using `glBindTexture()`
- ☐ Allocate space for texture of framebuffer size using `glTexImage2D()`
- ☐ Set texture filters and texture coordinate wrap parameters
- ☐ Attach texture to corresponding renderbuffer attachment using `glFramebufferTexture2D()`
  - ☐ Note: This will generally by attached to one of the CBOs using `GL_COLOR_ATTACHMENT`*i* but we could attach to the DBO using `GL_DEPTH_ATTACHMENT`
  - ☐ Note: Texture does not need to be limited to a 2D texture, can use a 1D texture, 3D texture or Cube Map as the texture object

Check Framebuffer Status

- ☐ Query status using `glCheckFramebufferStatus()`
- ☐ Check if returned value equals `GL_FRAMEBUFFER_COMPLETE`
- ☐ If returned value is something else, then check relevant error code

When rendering the scene – First Pass

- ☐ Bind `fbod` using `glBindFramebuffer()`
- ☐ Set viewport to framebuffer size using `glViewport()`
- ☐ Render scene as normal using shaders and VAOs
- ☐ Ensure OpenGL has finished drawing by calling `glFlush()`

When rendering the scene – Second Pass

- ☐ Detach framebuffer to render to screen by calling `glBindFramebuffer( 0 )`
- ☐ Use postprocessing shader program using `glUseProgram()`
- ☐ Bind framebuffer texture using `glBindTexture()`
- ☐ Set viewport to window size using `glViewport()`
- ☐ Set projection matrix to a 2D orthographic projection
- ☐ Render a textured quad that fills the screen

When cleaning up memory

- ☐ Delete the FBO using `glDeleteFramebuffers()`
- ☐ Delete the Renderbuffers using `glDeleteRenderbuffers()`