Intermediate Graphics & Animation Programming

GPR-300
Daniel S. Buckstein

Frame Buffers & Off-Screen Rendering
Week 3

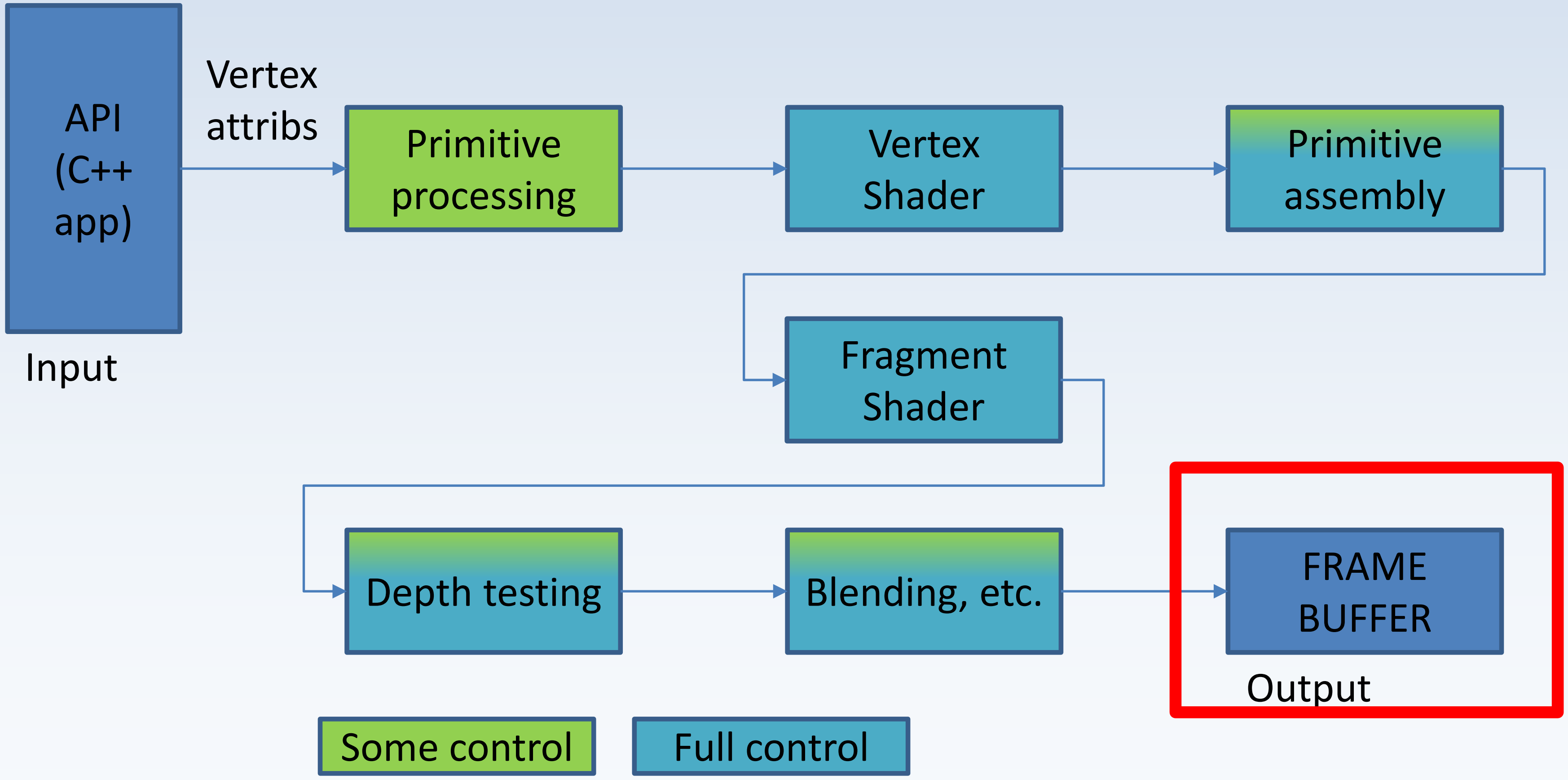
License

Frame Buffers

- Framebuffers and the display routine
  - Double-buffering
- Off-screen rendering
- Multiple-render targets
- Using FBOs

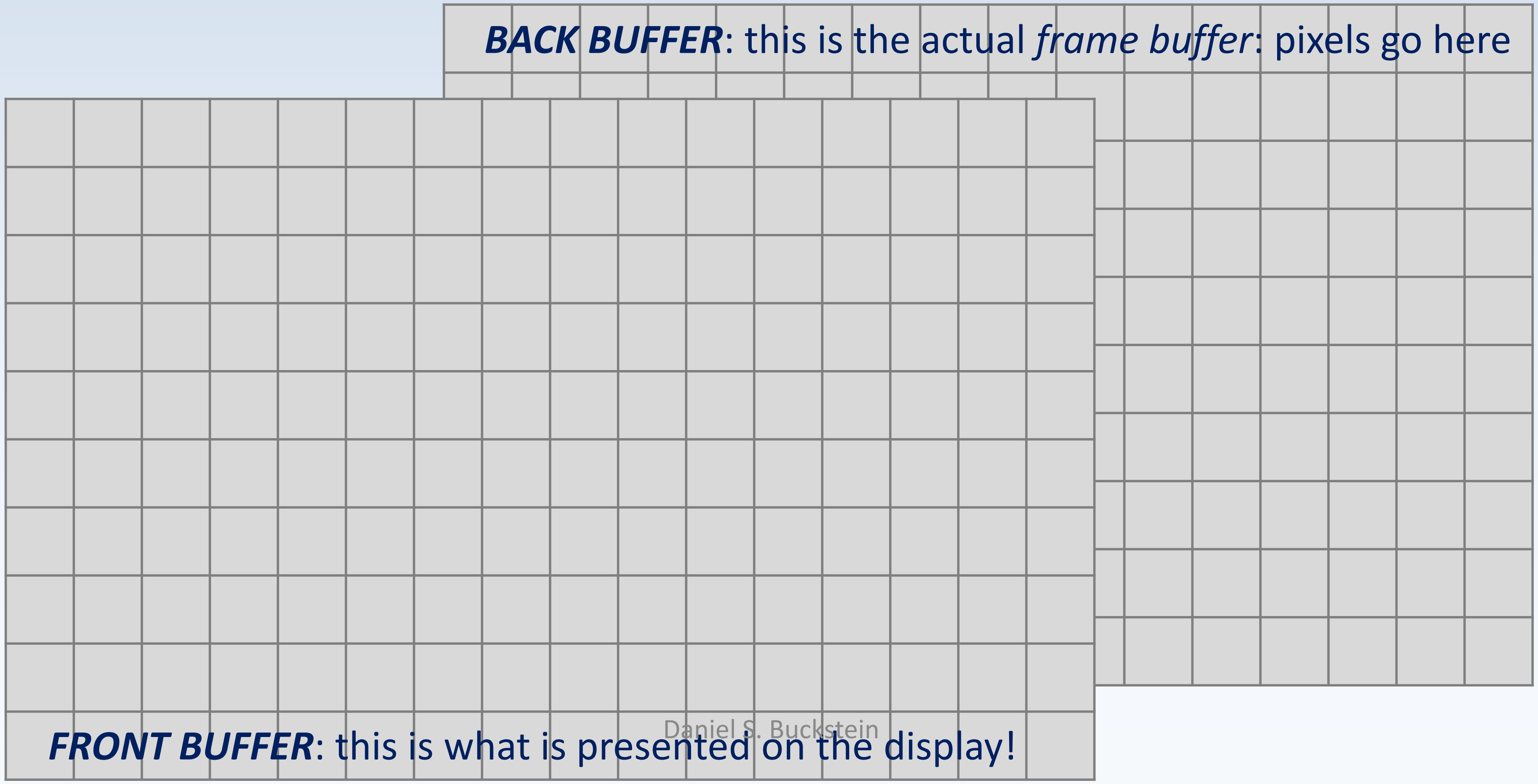
Frame Buffers

- The move towards programmable pipeline:



Frame Buffers

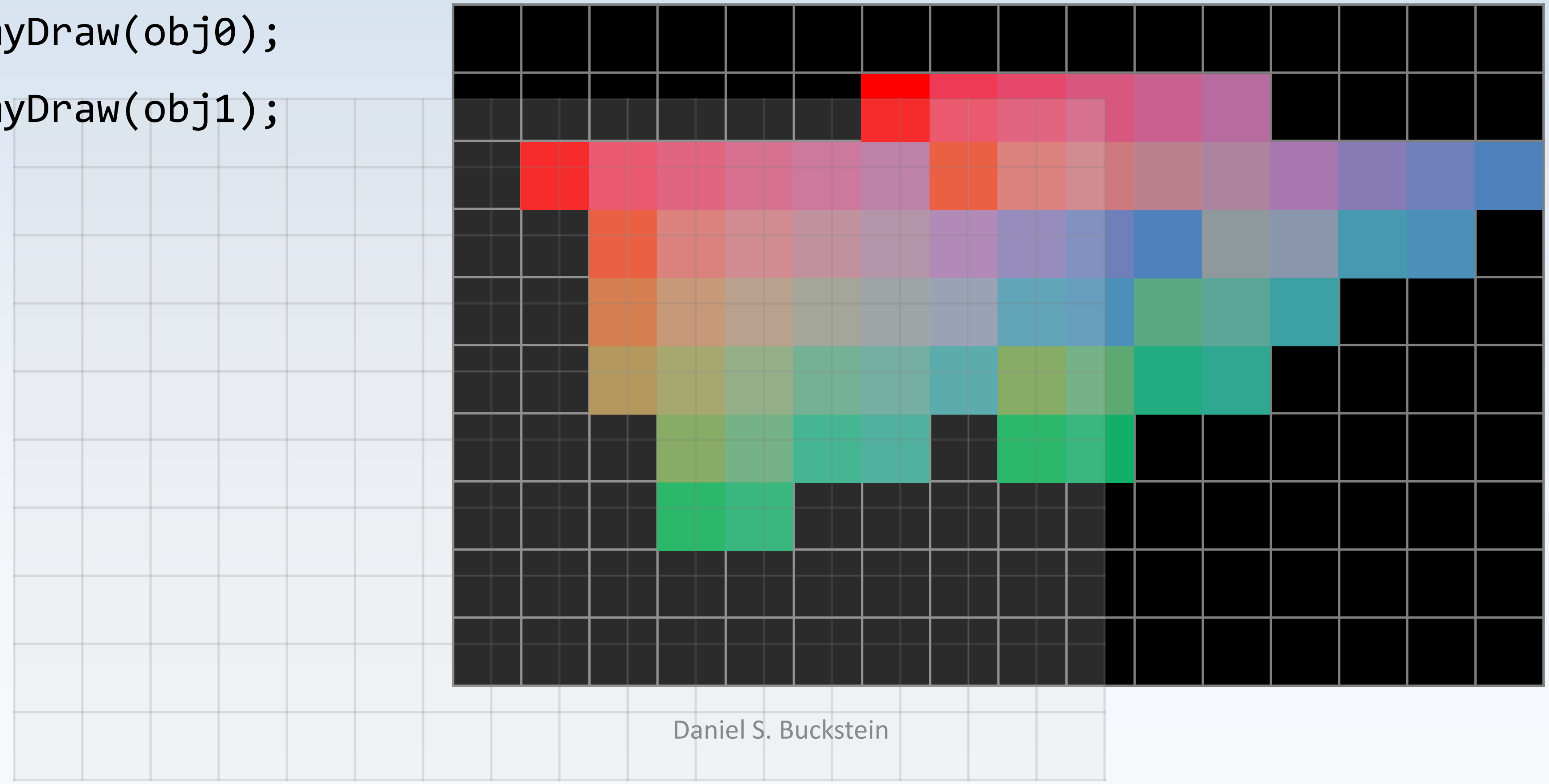
- Double buffering:


BACK BUFFER: this is the actual *frame buffer*: pixels go here
FRONT BUFFER: this is what is presented on the display!

Frame Buffers

- Double buffering: while rendering (1st frame)

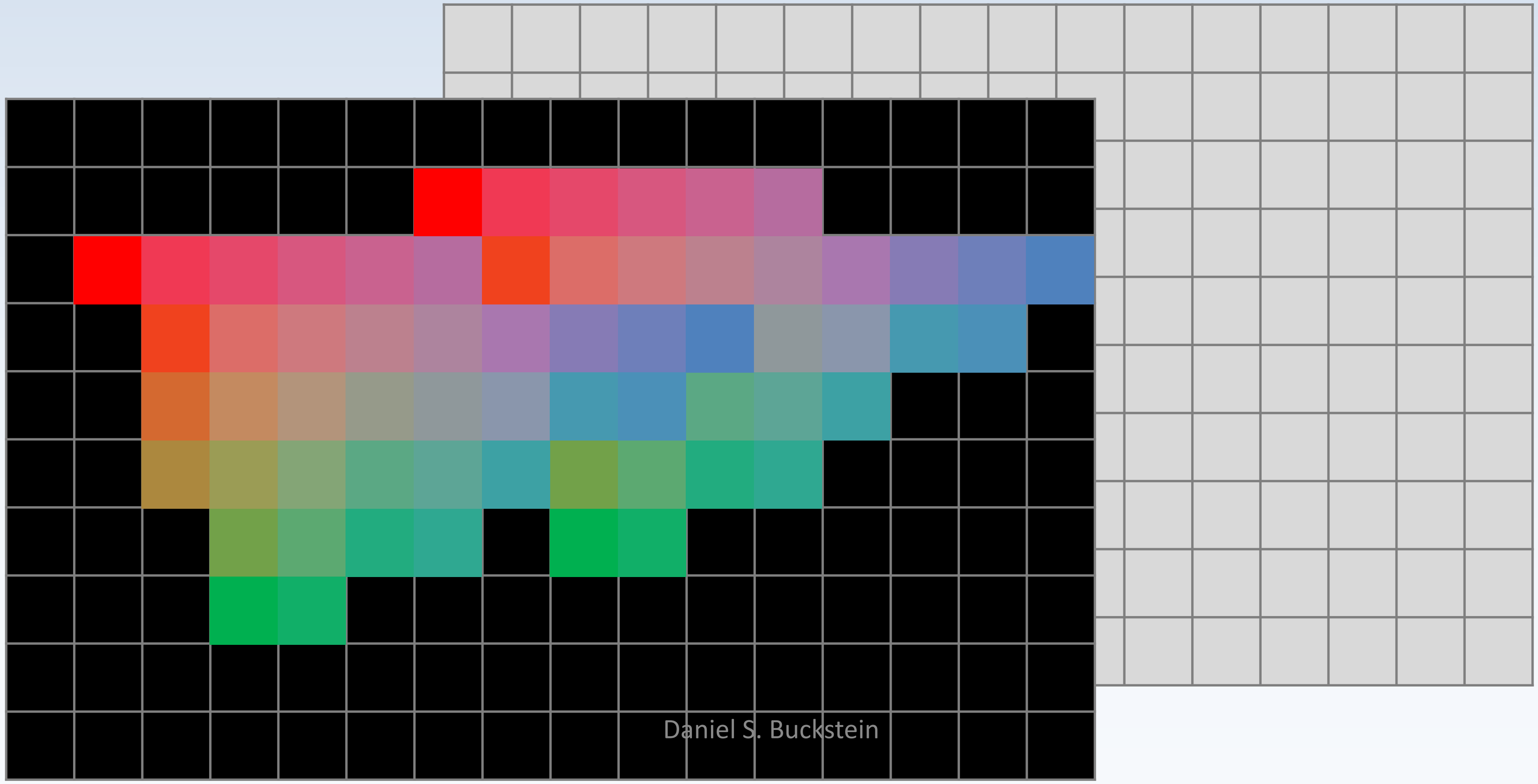
```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
myDraw(obj0);
myDraw(obj1);
```



Frame Buffers

- Double buffering: when finished rendering 1st

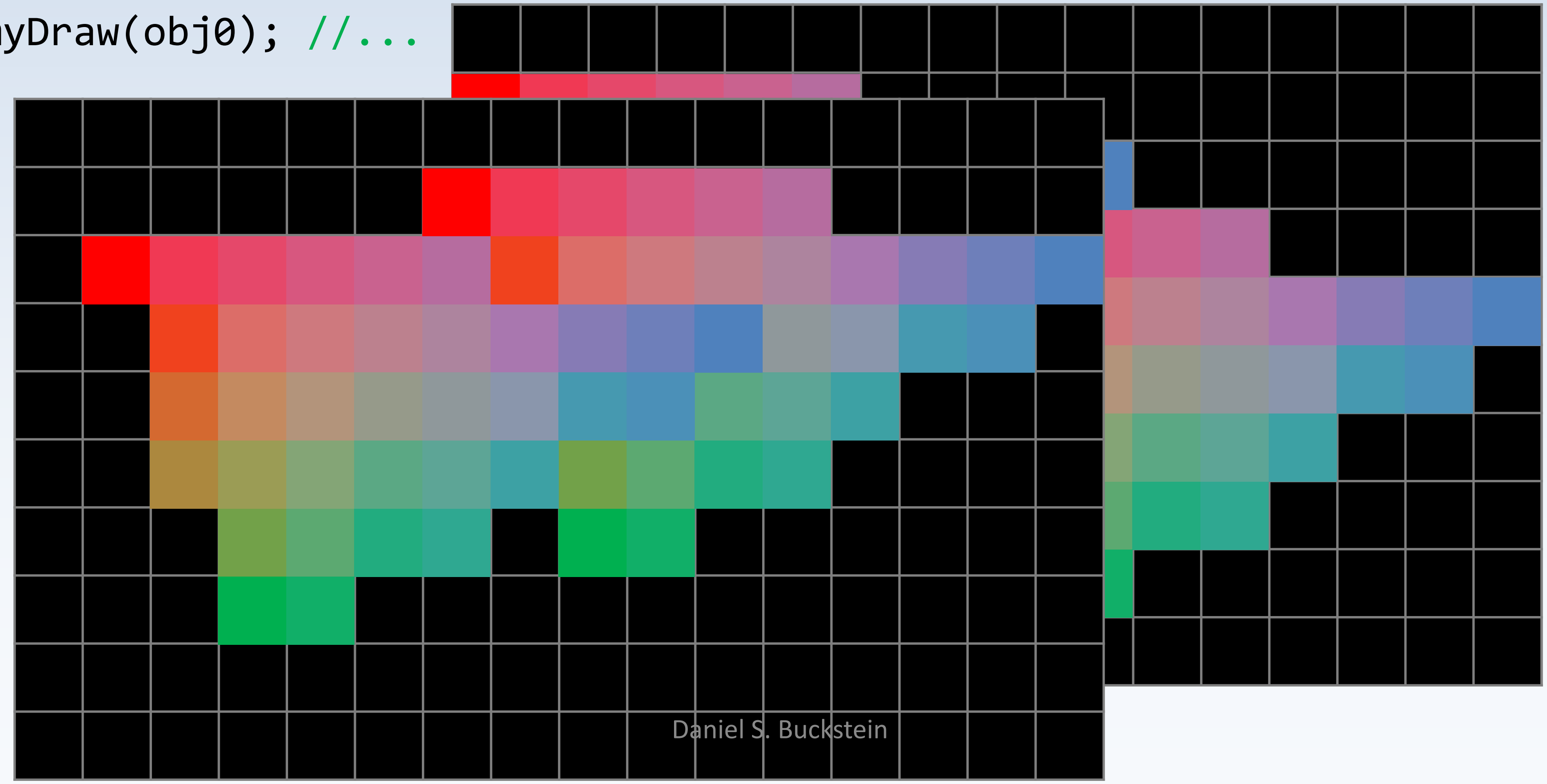
```
MY_SWAP_BUFFERS(); // e.g. glutSwapBuffers
```



Frame Buffers

- Double buffering: while rendering (2nd frame)

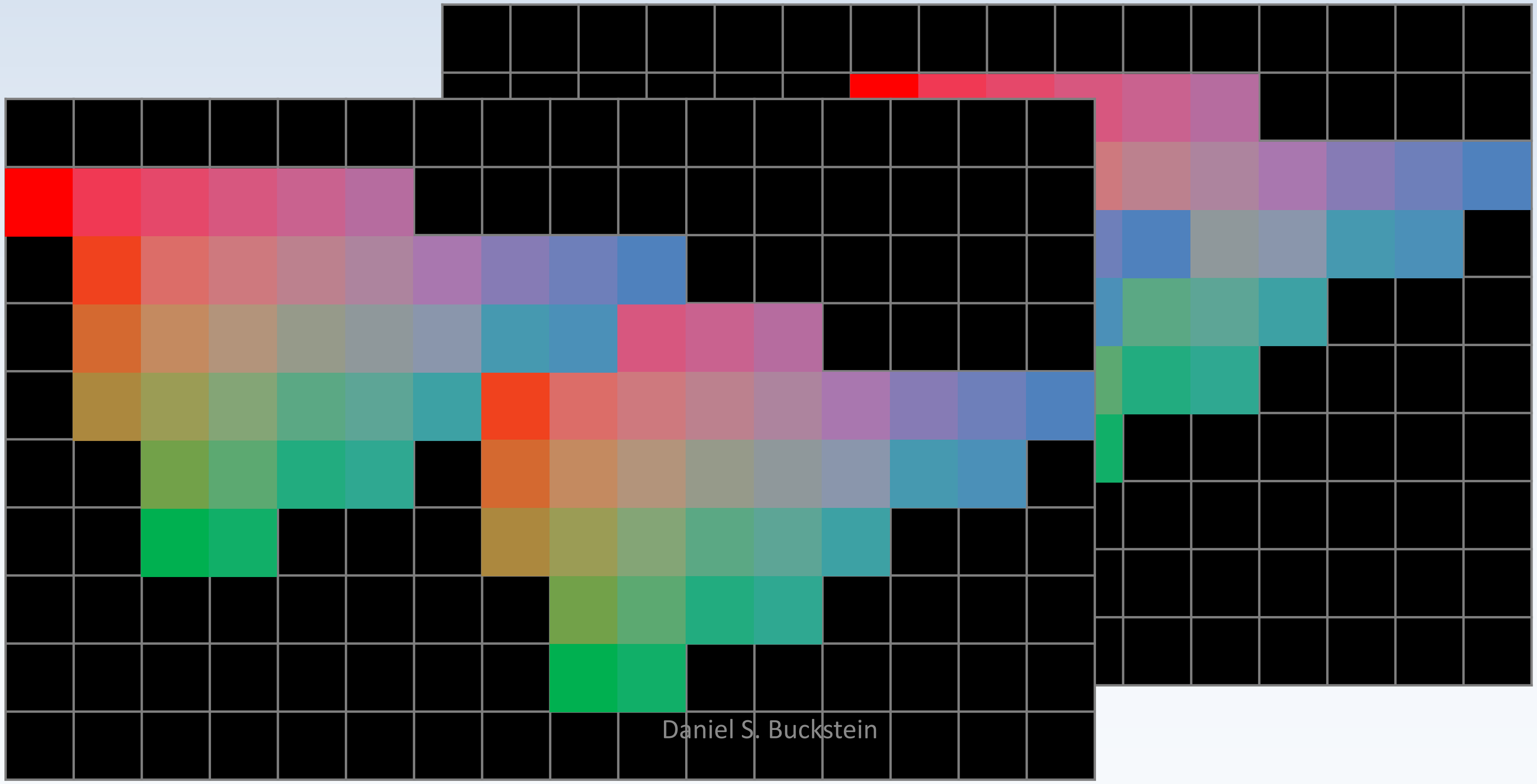
```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
myDraw(obj0); //...
```



Frame Buffers

- Double buffering: when finished rendering 2nd

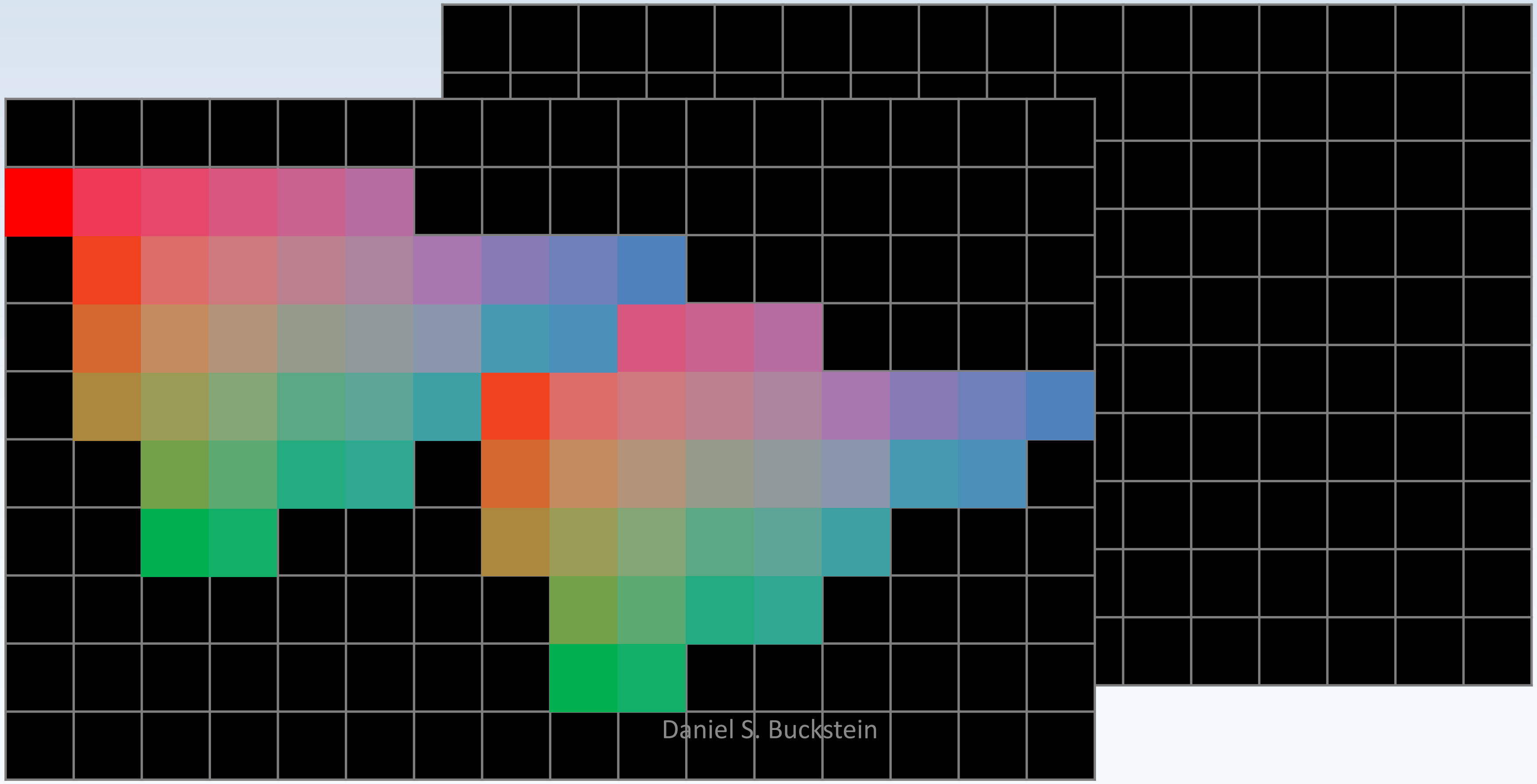
```
MY_SWAP_BUFFERS(); // e.g. glutSwapBuffers
```



Frame Buffers

- Double buffering: while rendering (3rd frame)

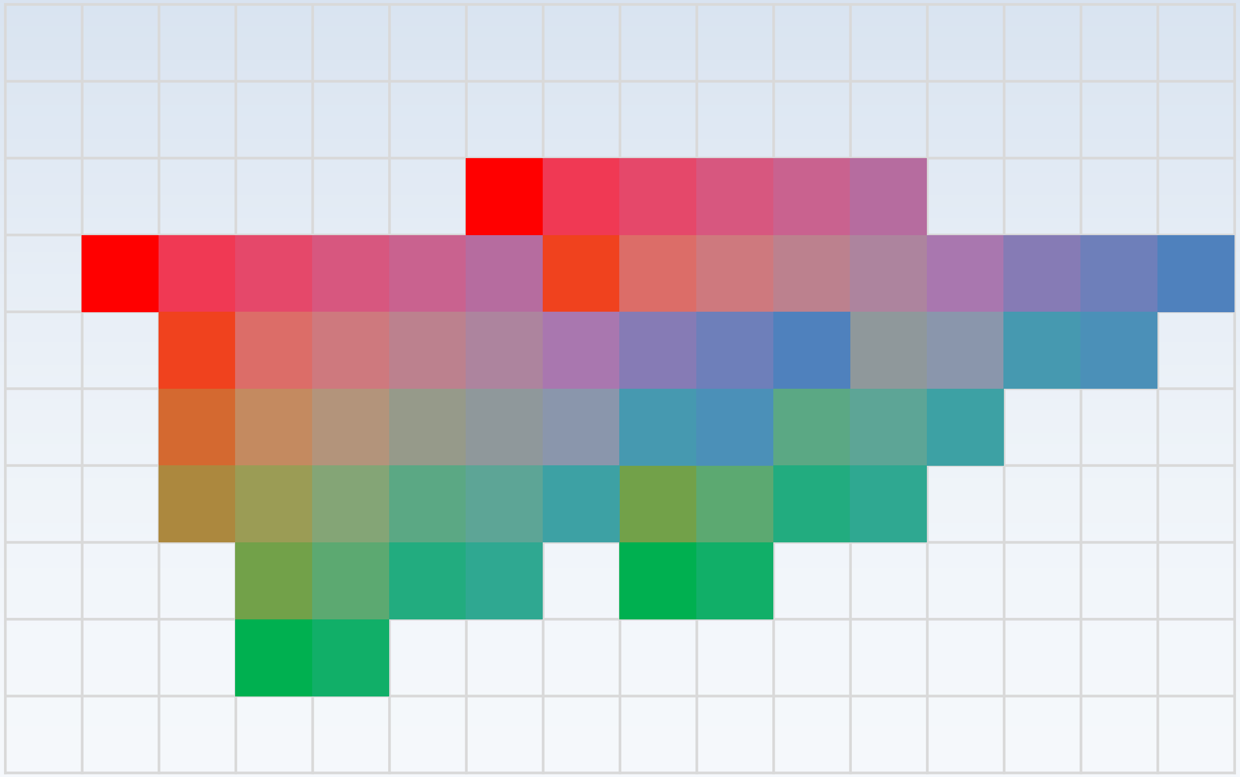
```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT ); //...
```



Frame Buffers

- Double buffering:

Back buffer is for drawing…



…front buffer is for display!

Frame Buffers

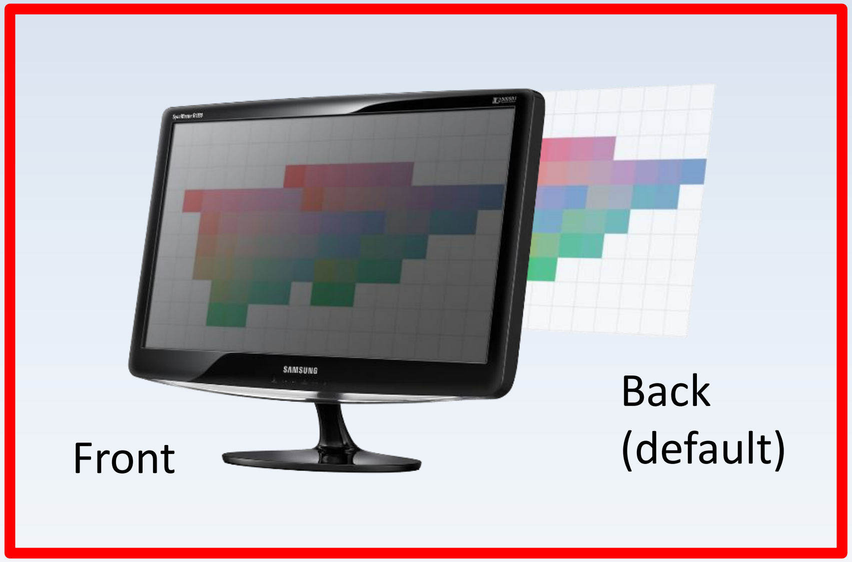
- The OpenGL back buffer is the default frame buffer, front is the default used to display…
- …but they are both valid ***render targets***:
- `glDrawBuffer( GL_FRONT ); // bad`
- `glDrawBuffer( GL_BACK );  // good`
- `glDrawBuffer( GL_NONE );  // what.`

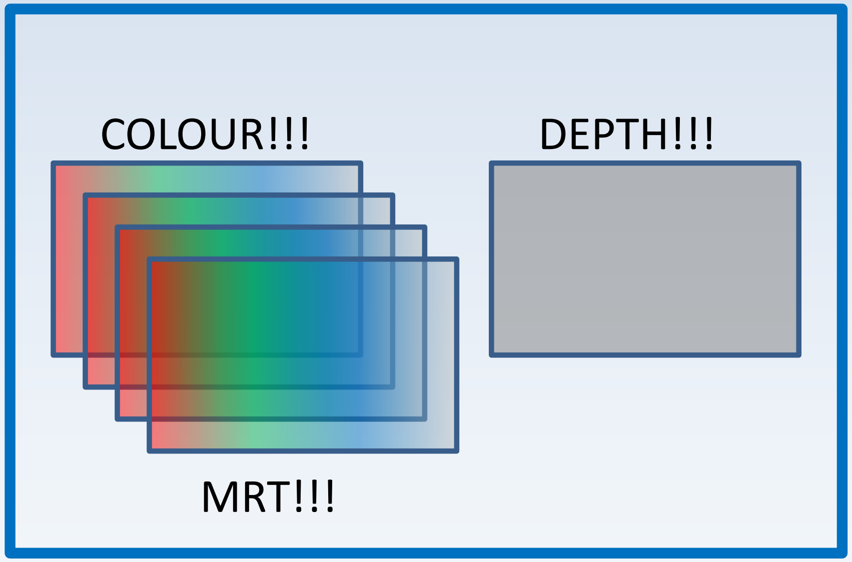
Frame Buffers

- ***Huge problem???***
- Cannot access the ***data*** stored in these buffers
- …nothing to ***process!!!*** ☹
- Post-processing requires additional buffers that we can control
- ***Frame Buffer Objects (FBOs)***

Frame Buffer Objects & Off-Screen Rendering

- "On-screen rendering": targeting the default buffers





Default buffers: on-screen target
→OpenGL-managed
→minimal control over data

FBO: off-screen target(s)
→User-managed:
→fully accessible… but how???

Frame Buffer Objects & Off-Screen Rendering

- ***Multiple Render Targets*** (MRT):
- A single frame buffer has several components:
- Colour, depth, stencil
- ***One*** target for depth OR depth/stencil combo
- … ***colour*** can have *many targets*
- Query from GL:

```
glGetIntegerv( GL_MAX_COLOR_ATTACHMENTS, &i_maxCount);
```

Frame Buffer Objects & Off-Screen Rendering

- ***Render to Texture*** (RTT):
- The output of off-screen rendering (targets) can be stored in textures!!! :D
- …yes, the same kinds of textures you use to give things colour!
- …which means we can sample from them!!!

- It's still just data!

Frame Buffer Objects & Off-Screen Rendering

- Setting up a FBO with MRT and depth:

```
// create and bind for configuration
glGenFramebuffers( 1, &fboHandle );
glBindFramebuffer( GL_FRAMEBUFFER, fboHandle );


// generate a SET OF TEXTURES for colour
glGenTextures( count, texHandleArray);
for ( int i = 0; i < count; ++i ) {
    // next slide: attach textures to FBO
    //  i.e. make them targets for output!
```

Frame Buffer Objects & Off-Screen Rendering

- Setting up a FBO with MRT and depth:

```
    glBindTexture( GL_TEXTURE_2D, // tex target
        texHandleArray[i] );      // handle
    glTexImage2D( GL_TEXTURE_2D, 0,
        GL_RGBA8, width, height, 0,
        GL_RGBA, GL_UNSIGNED_BYTE, NULL );
    glFramebufferTexture2D( GL_FRAMEBUFFER,
        GL_COLOR_ATTACHMENT0 + i, // attachment
        GL_TEXTURE_2D, texHandleArray[i], 0 );
} // end for loop
```

Frame Buffer Objects & Off-Screen Rendering

- Setting up a FBO with MRT and depth:

```
// set up depth component
glGenTextures( 1, &depthTexHandle );
glBindTexture( GL_TEXTURE_2D, depthTexHandle );
// configure texture, not RGBA format!
glTexImage2D( GL_TEXTURE_2D, 0,
    GL_DEPTH_COMPONENT24, width, height, 0,
    GL_DEPTH_COMPONENT, GL_UNSIGNED_INT, NULL);
glFramebufferTexture2D( GL_FRAMEBUFFER,
    GL_DEPTH_ATTACHMENT, // attachment
    GL_TEXTURE_2D, depthTexHandle, 0 );
```

Frame Buffer Objects & Off-Screen Rendering

- Setting up a FBO with MRT and depth:

```
// validate that the FBO is configured properly
if ( glCheckFramebufferStatus( GL_FRAMEBUFFER )
    == GL_FRAMEBUFFER_COMPLETE ) {

    // YOU ARE GOOD TO GO!!!  Handle success
}


// as always, unbind when you’re all done
```

Frame Buffer Objects & Off-Screen Rendering

- Now we have an off-screen target that we could render to if we wanted to!
- How to draw to an FBO instead of the back buffer:
- 1) Bind FBO
- 2) Tell OpenGL which targets to use (MRT)
- 3) Set viewport to match the size of targets
- 4) RENDER AWAY!!!

Frame Buffer Objects & Off-Screen Rendering

- Rendering to FBO:

```
// 1. bind FBO
glBindFramebuffer( GL_FRAMEBUFFER, fboHandle );
// 2. set render targets
const unsigned int targets[] = {
    GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1,
    GL_COLOR_ATTACHMENT2, // etc...
};
glDrawBuffers( count, targets );
// 3. set viewport
glViewport( 0, 0, width, height ); // x,y,w,h
```

Frame Buffer Objects & Off-Screen Rendering

- When you're done with your FBO… revert!

```
// 1. unbind FBO
glBindFramebuffer( GL_FRAMEBUFFER, 0 );

// 2. reset render target to back buffer
glDrawBuffer( GL_BACK );

// 3. reset viewport
glViewport( 0, 0, mainWidth, mainHeight );
```

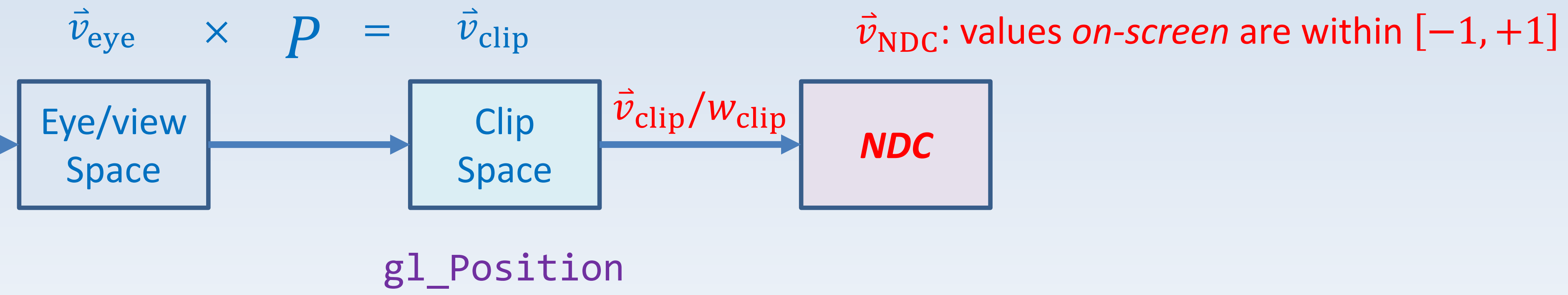
Frame Buffer Objects & Off-Screen Rendering

- We know about buffers and off-screen targets
- How do we *use* render textures?
- Our main focus:

- ***POST-PROCESSING***

Intro to Post-Processing with FBOs

- Normalized Device Coordinates (NDC)



$\vec{v}_{\text{eye}} \times P = \vec{v}_{\text{clip}}$
$\vec{v}_{\text{NDC}}$: values *on-screen* are within $[-1, +1]$
Eye/view Space
Clip Space
$\vec{v}_{\text{clip}}/w_{\text{clip}}$
NDC
gl_Position

*** If w_{clip} is equal to 1, then clip space and NDC are identical!

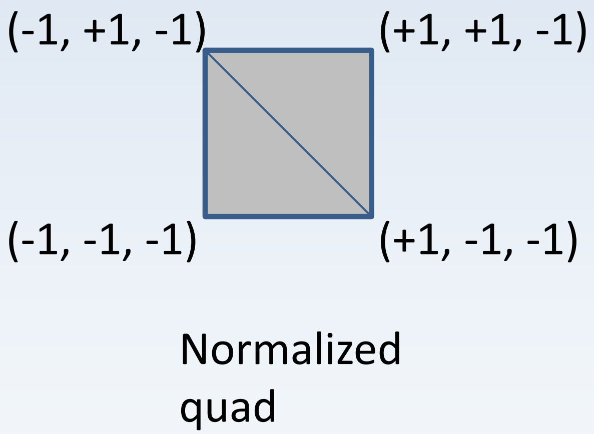
Intro to Post-Processing with FBOs

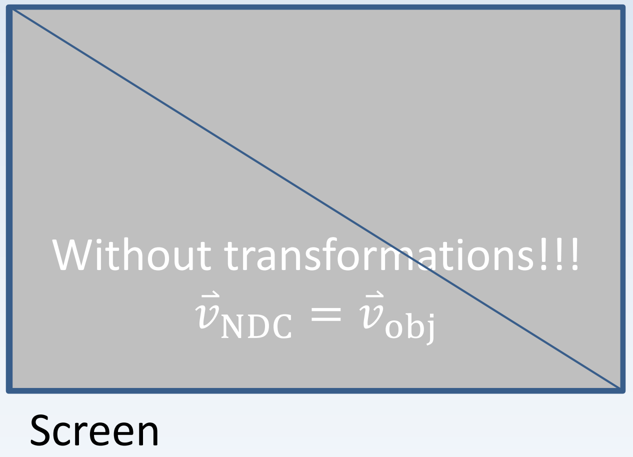
- If we know that vertices must end up in NDC…
- …and we know that the limits of NDC represent the edges of the screen…

- …then what happens if we draw a quad with values [-1, 1] with ***no*** transformations applied?

Intro to Post-Processing with FBOs

- Drawing a normalized quad with and without transformations applied:



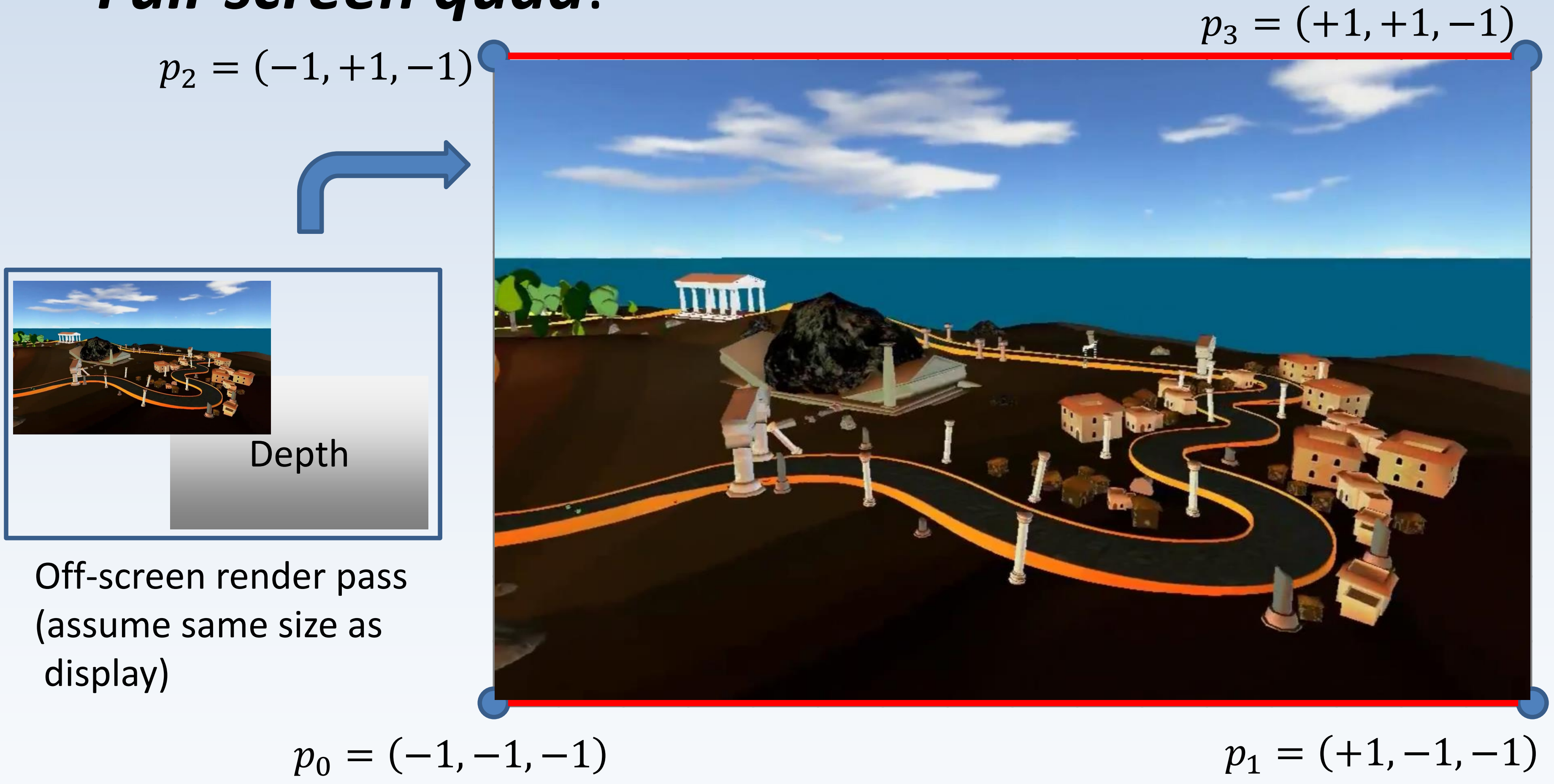

Without transformations!!!
$\vec{v}_{\mathrm{NDC}} = \vec{v}_{\mathrm{obj}}$
Screen

Intro to Post-Processing with FBOs

- Drawing a ***full-screen quad*** allows us to present the contents of our FBO!
- Ultimately, this completes the concept of a ***rendering pass***
- Enable FBO
- Draw scene
- Disable FBO, bind texture
- Draw FSQ***

Intro to Post-Processing with FBOs

- ***Full-screen quad***:



Off-screen render pass (assume same size as display)

Intro to Post-Processing with FBOs

- Full-screen quad shader program:
- Vertex shader: no transforms
  - Output vert as NDC
  - Convert NDC to screen-space sampling coordinate
- Fragment shader: ***post-processing algorithm***
- Use this principle for ***any and all post-processing techniques*** ☺
- I.e. this is your core tool!

Intro to Post-Processing with FBOs

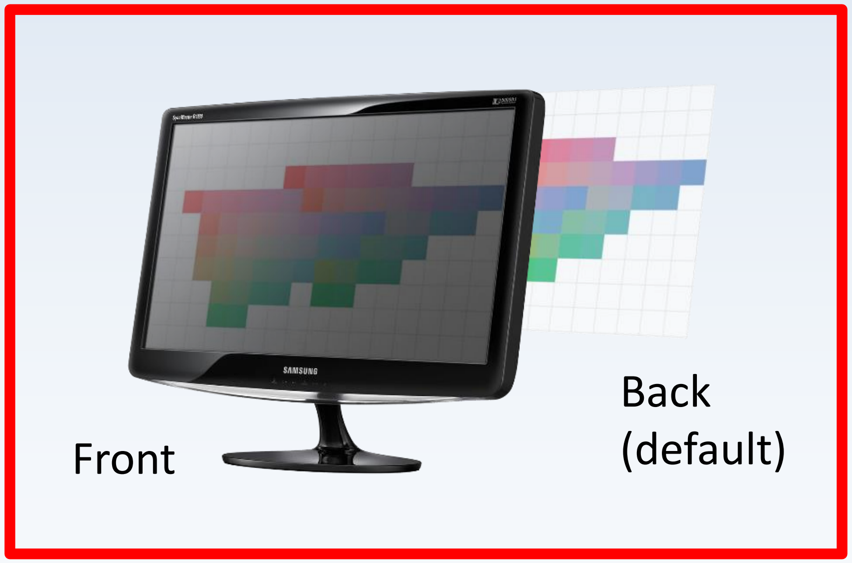
```
// vertex shader (GLSL 1.2)
attribute vec4 position; // xyz=[-1, 1] and w=1
varying vec2 screenCoord;
void main() {
    gl_Position = position; // no transforms
    // convert normalized pos to screen-space
    //  texture coordinates :)
    screenCoord = SERIALIZE(position.xy);
}
```

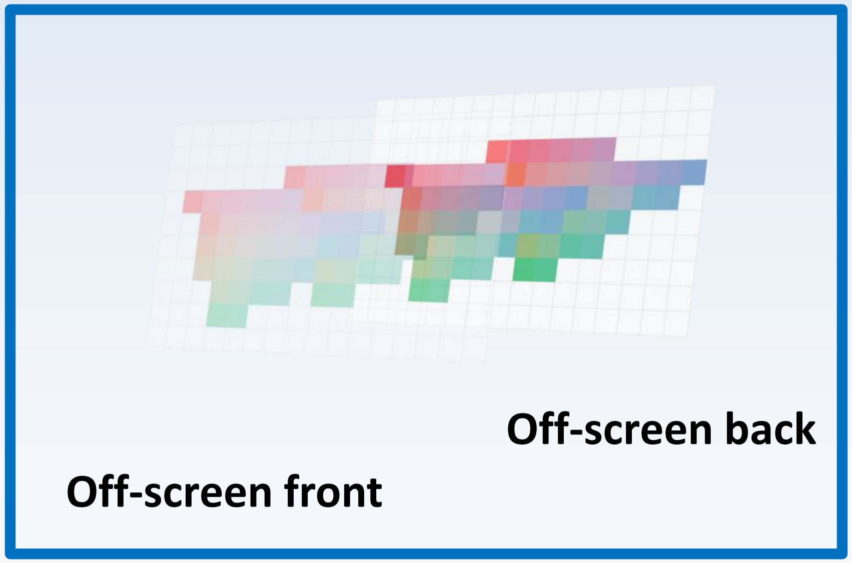
Intro to Post-Processing with FBOs

```
// fragment shader: time to do post-processing!
varying vec2 screenCoord; // xy=[0, 1]
uniform sampler2D myFBOtexture; // from FBO
void main() {
    // sample from input texture
    vec4 pixelColour =
        texture2D(myFBOtexture, screenCoord);
    gl_FragColor.rgb = pixelColour.rgb;
}
```

FBO Pro Tips

- How many FBOs do you need for a 3-pass algorithm???
- One per pass???





FBO Pro Tips

- What does this line mean???

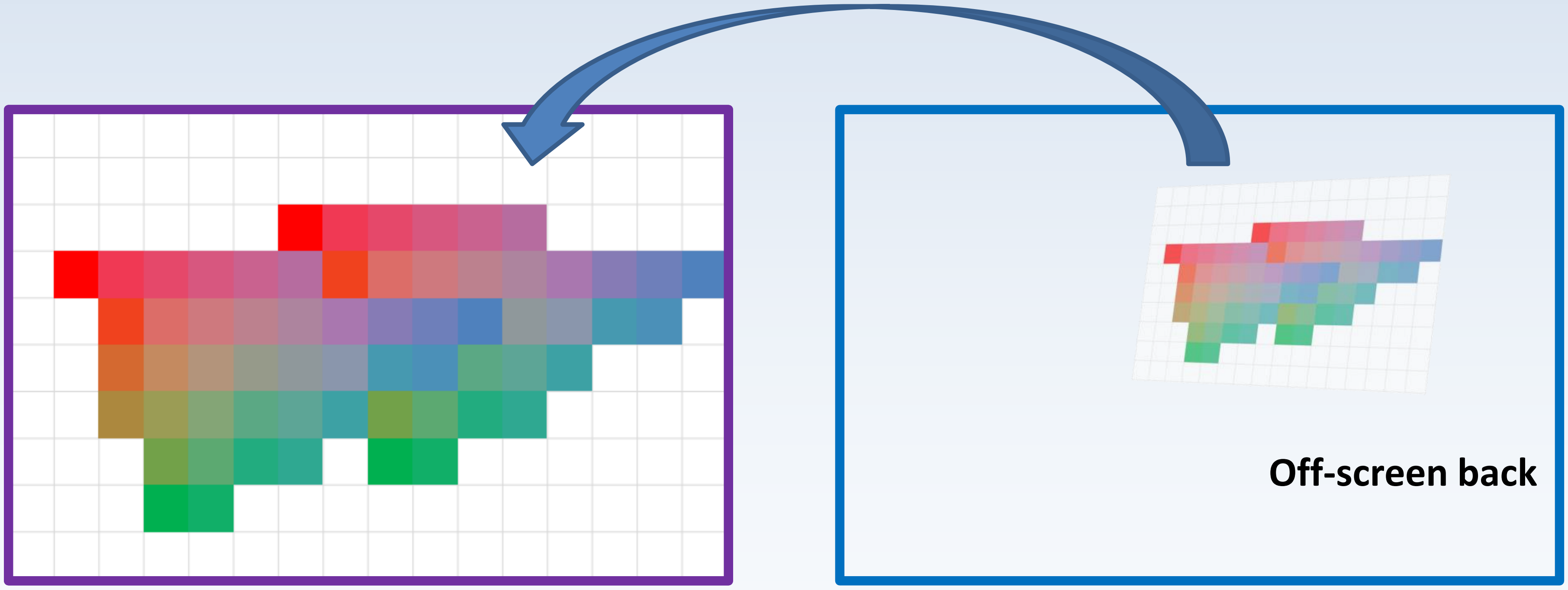
```
glClear( GL_COLOR_BUFFER_BIT );
```

- How about this one?

```
glClear( GL_COLOR_BUFFER_BIT |
         GL_DEPTH_BUFFER_BIT |
         GL_STENCIL_BUFFER_BIT );
```

FBO Pro Tips

- When should you use `glClear` ?
- When should you ***not*** use it?



Receiving buffer
(NO DEPTH TEST)

FBO Pro Tips

- Some platforms ***only use FBOs***
- Control over hardware back buffer is ***locked***
- How do you ensure you can draw to it???

FBO Pro Tips

- Typical game rendering loop (mobile or not):

1) Draw final pass from last frame
   - (presented to display at the end of this frame)
2) Update
3) Execute all rendering operations using FBOs

The end.

- Questions? Comments? Concerns?