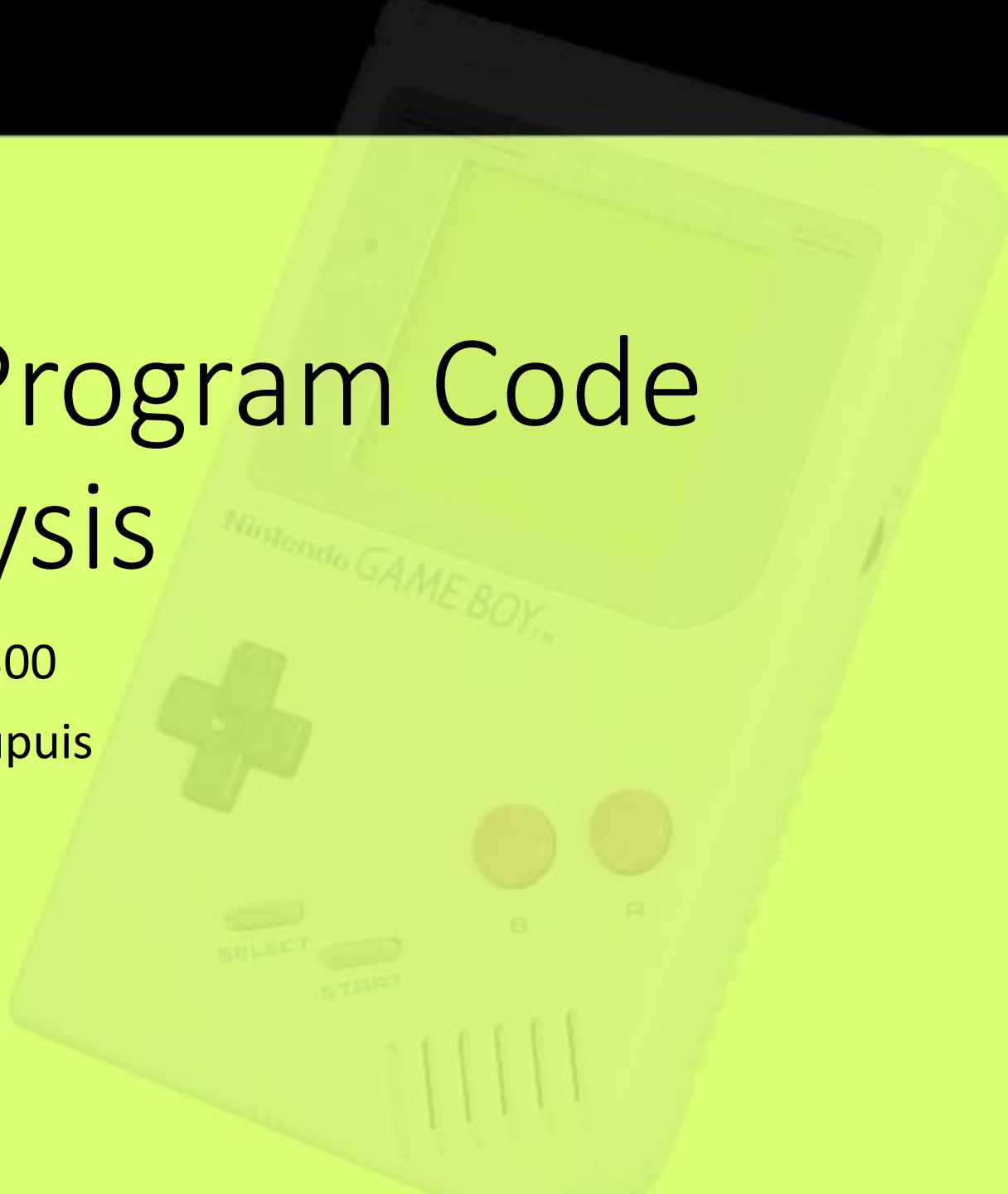
Basic OpenGL Program Code Analysis

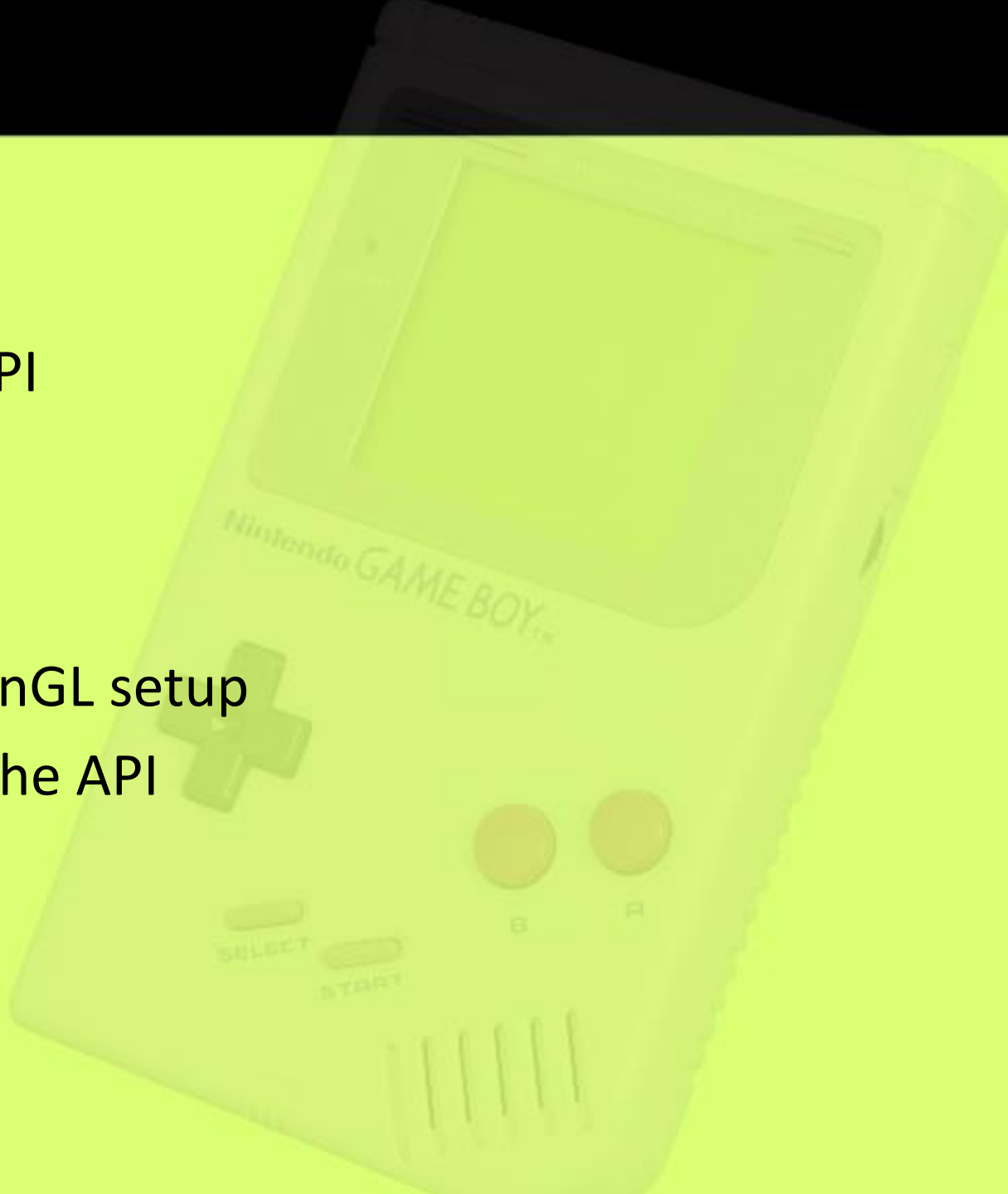
Game 300

James Dupuis



OBJECTIVES

- Add a Library to an existing project
- Understand common terms for an API
  - Versioning
  - Deprecation
- Understand the context code
- Understand how we handle the OpenGL setup
- Understand naming conventions in the API



Adding OpenGL to our Project

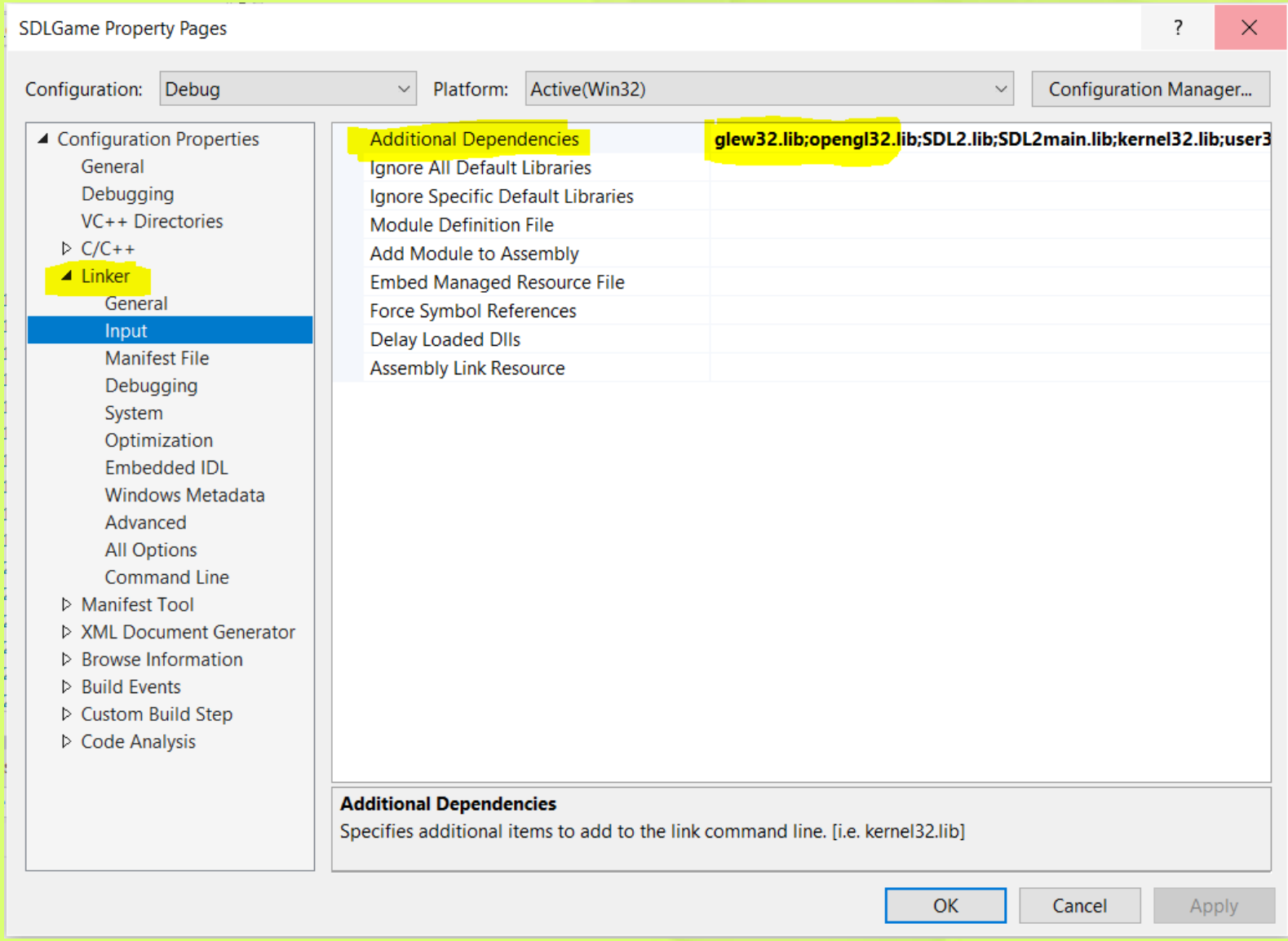
- OpenGL is an additional API **(Application Program Interface)** to SDL.
  - There are 3 separate parts to a API library
    - a dynamic link library (.dll)
    - A library file (.lib)
    - a header (.h)
  - To add OpenGL to our project we need to place **glew.h** into our lib folder where libraries are contained.
  - **Glew32.dll** and **glew32.lib** need to be placed in the main project folder.
    - GLEW stand for OpenGL Extension Wrangler.

Adding OpenGL to our Project

- A .**DLL and .lib** are dynamic link library files.
  - Essentially this is a prebuilt set of code that you have access to (the API)
  - You can call functions inside of here, however, you do not see how the internal function works.
    - This is known as **abstraction**.
  - The glew.h file supplies you with the function headers so you can see what the function expects for parameters and what it returns.

Adding OpenGL to our Project II

- When adding in a library to your project you must link them.
- To do this, access your project settings and select Linker>Input> Additional Dependencies.
  - Here you can add the new libraries placed in your project so that the project itself knows to link them when you compile.



OpenGL Context

- Once we've linked to the main libraries we can include and Initialize glew.h (openGL).

```
16      //new OpenGL/GLEW headers:
17      #define GL3_PROTOTYPES 1
18      #include "glew.h"
```

- We next need to tell our SDL window that is will also be used to now render OpenGL by supplying SDL_WINDOW_OPENGL as the last parameter.

```
// create our basic SDL window, this is why we have both a console window and a window for our graphical game.
// we will use this for both 2D and 3D rendering.
window = SDL_CreateWindow("Basic SDL Project",
    SDL_WINDOWPOS_UNDEFINED,
    SDL_WINDOWPOS_UNDEFINED,
    WINDOW_WIDTH, WINDOW_HEIGHT,
    SDL_WINDOW_OPENGL);
```

OpenGL Context

- OpenGL has a Context
  - Represents the current state of the window to display the graphics on
  - Can have multiple contexts established.
    - Only one context per CPU thread at a single time.
  - Contexts can be created and destroyed similar to an object in C++
  - When a context is destroyed so is all it’s data.
  - Data can be transferred from one context to another
  - You can create a new context using the SDL API function SDL_GL_CreateContext()

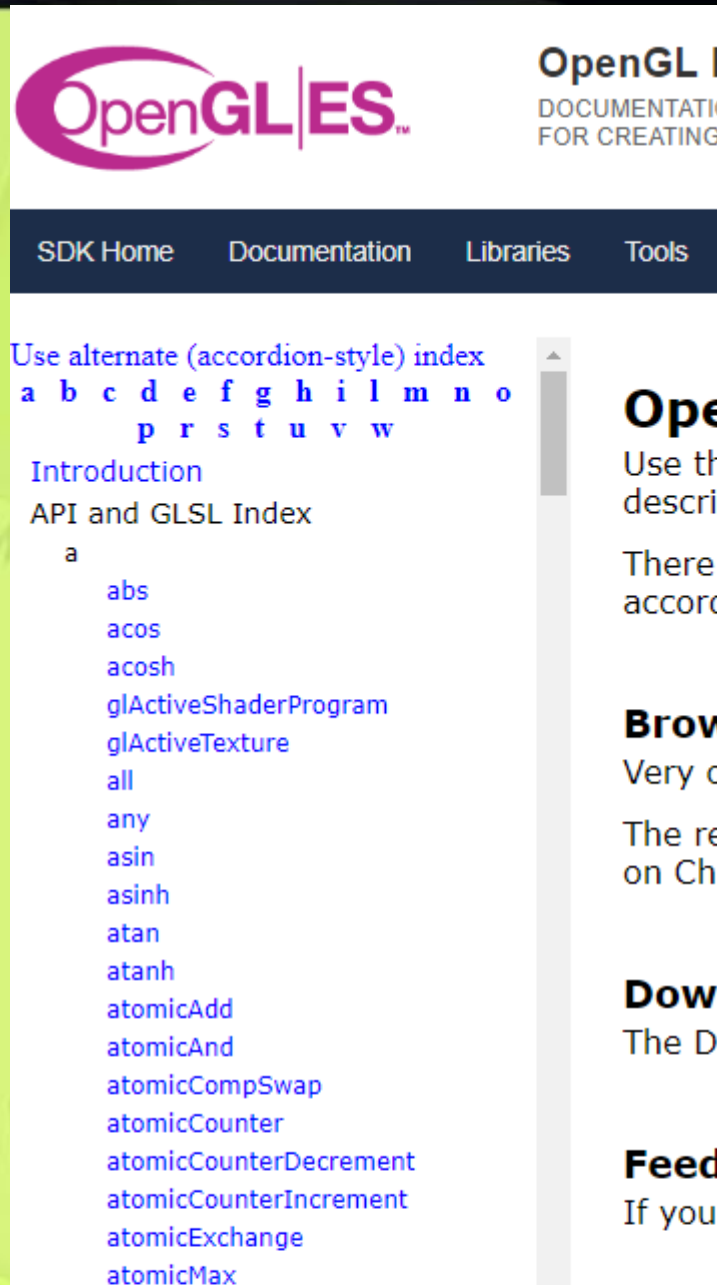
```
// initialize our context using our window as the source
SDL_GLContext GLContext = SDL_GL_CreateContext(window);
```

Initializations

- Additionally we will need to initialize glew so that it knows we want access to all the OpenGL API calls with glewInit().

```
glewInit();
```

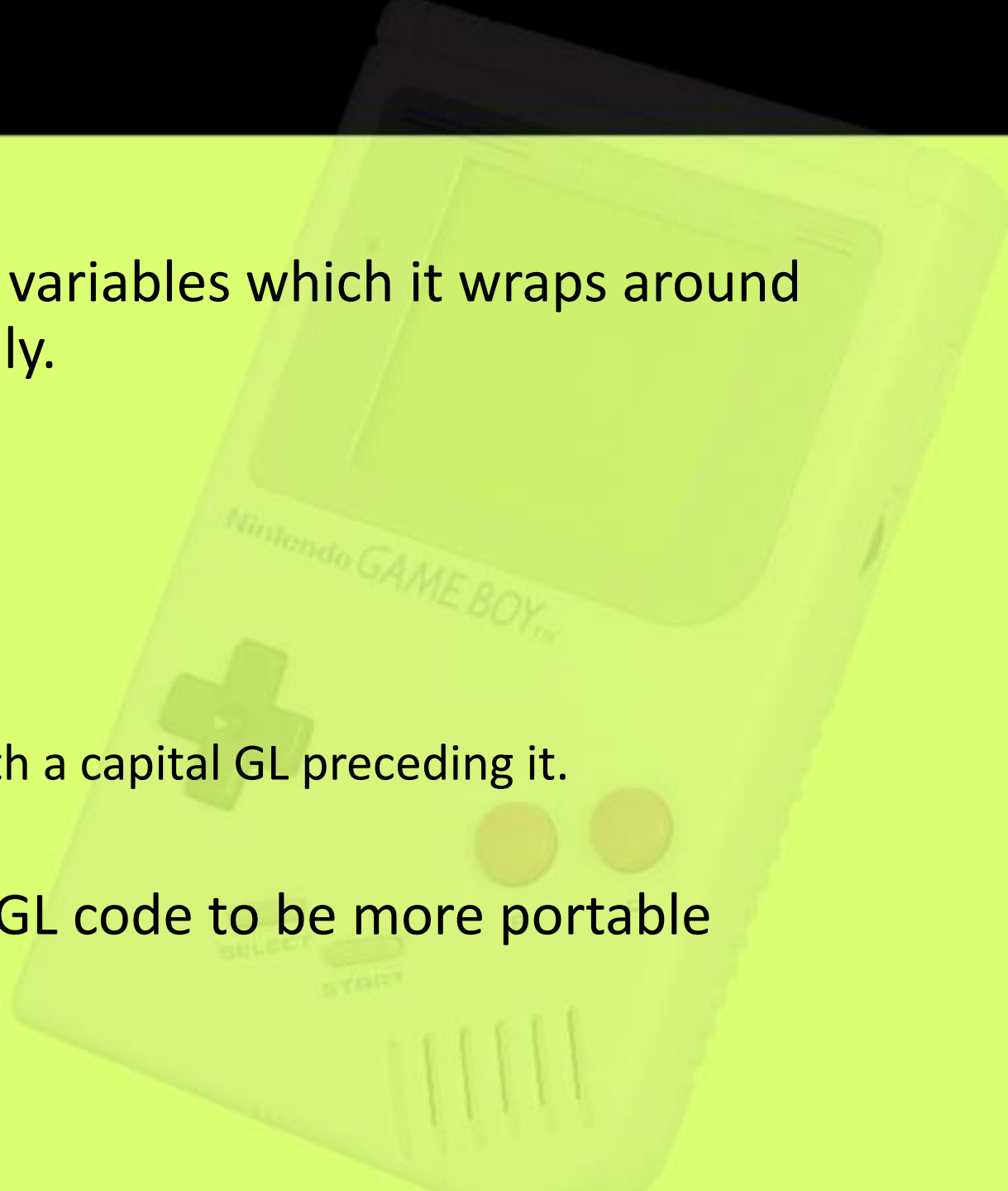
- We can now take a look at the OpenGL API which we have complete access to:
  - https://www.khronos.org/registry/OpenGL-Refpages/es3.1/



OpenGL Variables

- The OpenGL library includes a set of variables which it wraps around standard variable types used normally.
  - int => GLint
  - uint => GLuint
  - float => GLfloat
  - bool => GLbool
  - etc…
  - Notice it’s essentially just the same with a capital GL preceding it.

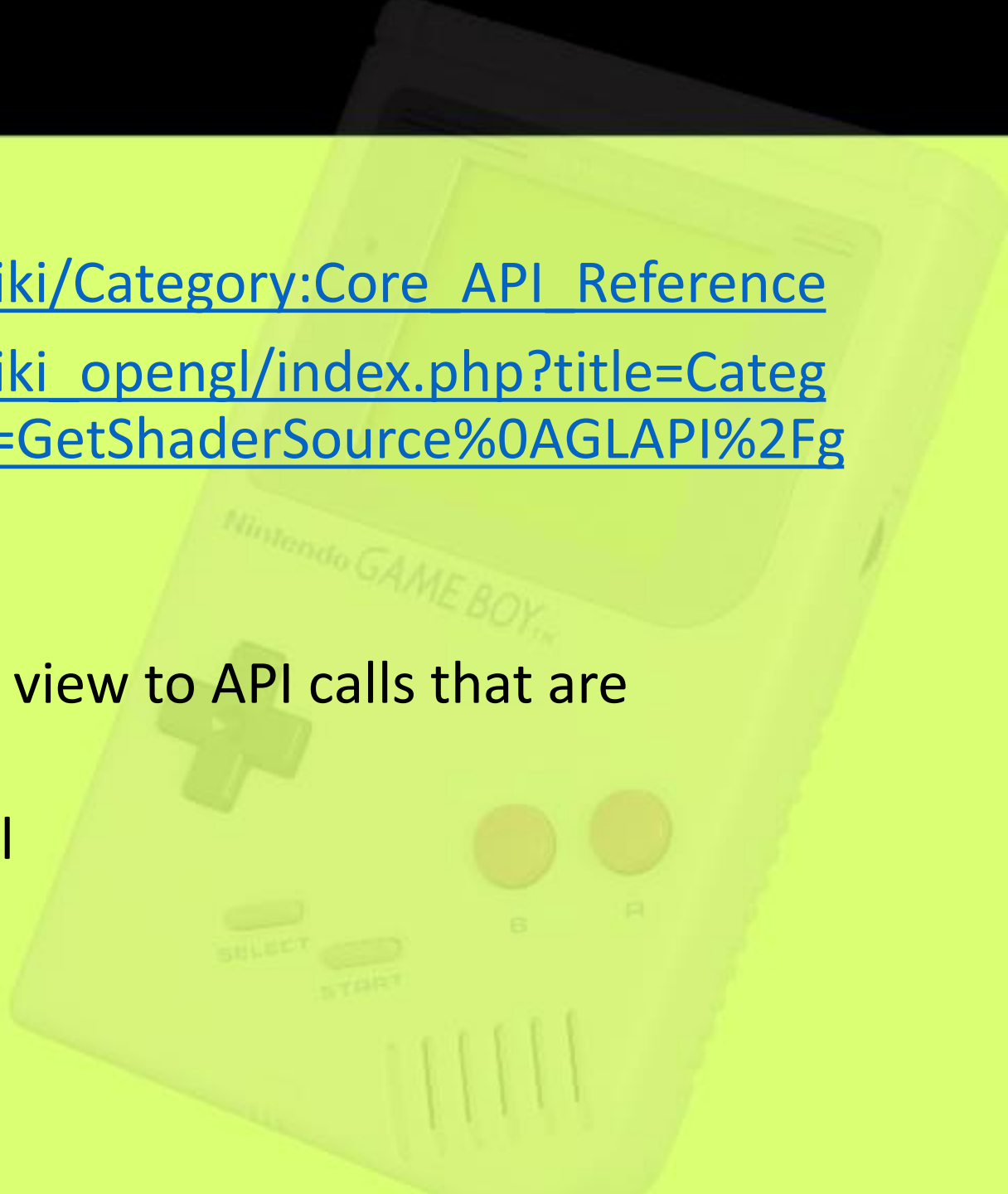
- Wrapping the variables allows OpenGL code to be more portable between consoles, devices and PC.



OpenGL functions

- https://www.khronos.org/opengl/wiki/Category:Core_API_Reference
- https://www.khronos.org/opengl/wiki_opengl/index.php?title=Category:Core_API_Reference&pagefrom=GetShaderSource%0AGLAPI%2FglGetShaderSource#mw-pages

- Use the subcategories to restrict the view to API calls that are relevant to the current task.
- All OpenGL API calls also start with gl
  - Lowercase gl indicates a function
  - Uppercase GL indicates a variable type



Depreciation

- API's often have a **deprecation** model.
  - As time goes on API's improve by adding in new functionality.
    - This is known as **versioning**
  - Occasionally functionality becomes obsolete or restructured.
  - The removal of functions falls under what's known as a deprecation model.
- Not all API's follow a hard set rule on how to handle deprecation.
- Some API's just remove the function, others ease them out over time allowing developers to adapt.
- OpenGL falls under the easing category.
  - In general, they attempt to keep pre-existing functionality in place as long as possible unless it interferes with new functionality.
  - In version 3.3 of OpenGL a large overhaul was made to the way things functioned.

First Functions

- To begin with, the first thing we need to learn is to clear the screen.
  - To do this use the glClear command:

```
glClear(GL_COLOR_BUFFER_BIT);
```

  - Note the GL_COLOR_BUFFER_BIT.
    - This is a defined bitmask which represents the color buffer OpenGL is managing.
  - Using the glClear command on it's own will always clear the screen to black.
  - To set the color we want to clear the screen with we can use the glClearColor function, prior to calling the main clear function.
  - glClearColour takes four parameters representing Red, Green, Blue, and Alpha.

```
glClearColor(1.0, 0.0, 0.0, 1.0);
glClear(GL_COLOR_BUFFER_BIT);
```

Window Update

- Lastly to make anything actually render, we need to tell SDL to update the screen using:

```
SDL_GL_SwapWindow(window);
```

- https://wiki.libsdl.org/SDL_GL_SwapWindow

- We pass in the SDL window we created so it knows where to apply the current context.

Summary

- Topics Covered:
  - Add a Library to an existing project
  - Understand common terms for an API
    - Versioning
    - Deprecation
  - Understand the context code
  - Understand how we handle the OpenGL setup
  - Understand naming conventions in the API

- Read Chapter 1 & 2 of book for additional support.