Sea2D – A 2D Sprite and Game Engine for OS X

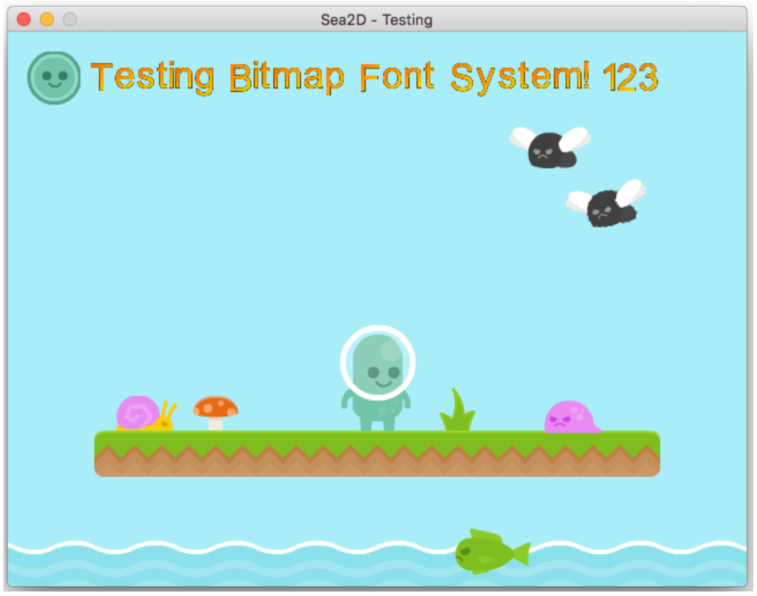
Jibran Syed

Intro

The goal with this research was to make a 2D graphics/game engine that utilized sprites as the main resource to render (as opposed to meshes in a 3D engine). Definition of sprite data would be defined in external files instead of inside code, allowing for little code modification to add new sprites. Sea2D stands for "Sprite Engine Apple 2D".

The other main goal of this engine was to make a graphics engine that would run on Apple's OS X platform. Most graphics code I have written was aimed at Windows and found it useful to learn the nuances of implementing similar systems on a different platform like OS X. This would also allow me to understand how to code C++ applications in OS X. These pursuits are useful in case a developer ever wishes to port their engine to another platform, or for multiplatform engine development.

Other minor goals included a simple scene-graph system and a bitmap font system. Most fonts, like TrueType are vector-based fonts, which is far beyond the scope of this engine. With a bitmap-based font, any sprite can be made to represent characters in a string of text.



C++ Development on OS X

One of the most common methods of developing C++ applications on OS X is to use Xcode. Xcode facilitates the development of apps in Objective-C and Swift, but is slightly lacking when helping write C++.

Of course, since C++ is generally a multiplatform language, the full C++11 feature set is available on OS X, although most of those features were not used in this project.

C++ can be used to make different kinds of software, from windowed apps, to command line utilities, and libraries. The most common way to start a C++ project in Xcode is to choose the "Command Line Tool" template when starting a new project. Many online tutorials choose this route whether they want to develop command line apps or windowed apps.

But what most people don't know is there is actually another project template that is better suited for GUI apps written in C++. It doesn't look obvious, but you can make a C++ windowed application starting from the "Cocoa Application" template. This template does not offer C++ as a language option, which is why it doesn't look so obvious as a choice for C++ projects.

So to make a C++ windowed application in Xcode, you choose the "Cocoa Application" template and select "Objective-C" as the language for the project ("Swift" is the only other option). The reason you would choose Objective-C is because Objective-C code can allow C++ code to be written on top of it. And Objective-C's `main()` function is basically in C, allowing the developer to replace the Objective-C code in `main()` with C++ code.

The library used for windowing was GLFW3. Setting up the library was mostly straight forward, and the setup code is very similar in Windows, but with one slight difference. On OS X, the following line needs to be added with the GLFW setup code otherwise OpenGL 3+ features cannot be used:

```
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
```

The OpenGL library used was GL3W, and image loading used a C library called stb_image.

Game Data Assets in YAML

Game engines usually compose of files known as “assets” which compose of resources needed to create a game. Most well known types of assets include code, art, and audio.

But there is also another kind of asset needed that is usually defined in pure data form; this will be dubbed a “data asset”. Data assets are very useful for the developer to define custom properties of the game without modifying code.

An example would be defining shader properties from a data file without hard-coding those properties in code. Perhaps you have a shader than can take in a particular tint color as a uniform value. Instead of setting that color directly in code, you can set that property in a data file, which the engine will load and automatically assign that property to the shader.

There are many methods to create data assets, using various binary or text formats. Sea2D utilizes a text format for data assets called YAML, which allows loading of data in whatever list or hierarchy the developer desires.

YAML files are used extensively in defining data needed to render sprites and bitmap fonts.

Sprite System

In Sea2D, a “Sprite” is defined as a collection of “ImageFrames” that are defined from a texture atlas (aka spritesheet).

Textures in Sea2D that are meant to be spritesheets have a spritesheet definition data asset associated with the texture that defines the list of ImageFrames. An ImageFrame is a “sub-image” of the texture given as a rectangular partition of that texture. In turn, Sprites are a collection of at least 1 ImageFrame.

The distinction between an ImageFrame and Sprite is needed because an ImageFrame is always a single image, while a Sprite could be composed of multiple images, if it’s animated. The idea was that regardless of whether the Sprite was animated or not that it would have a reference of ImageFrames that it can use to draw, and animated sprites had the extra logic of determining the next ImageFrame to draw.

Sprites have a data file per texture defining which ImageFrames compose of a sprite, and the name of a sprite.

Every sprite renders with the same quad mesh, but the ImageFrame dimensions are sent in to a special "sprite shader" that draws sprite images. An ImageFrame composes of an X,Y pair and the width and height of the sprite, all in pixels. These values are normalized before being sent into the shader, because OpenGL deals with normalized texture coordinates, not pixel coordinates.

Bitmap Font System

The font system used in Sea2D is a basic form of raster-based font glyph rendering. Typical (advanced) font systems utilize vector-based fonts like TrueType have employ advanced typographical parameters like kerning. These things are beyond the scope of this research and it is only concerned with a simple raster-based font system suitable for one line of text.

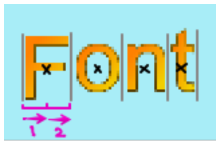
The font system in this engine utilizes the Sprite system in order to render a string of glyphs in the scene. A glyph here is defined as a sprite that represents a single character of text.

Font glyphs, like sprites themselves, are placed onto a texture atlas and dissected in the same way spritesheets are. Thus a collection of ImageFrames is defined representing every glyph found in the texture atlas.

After the ImageFrames for the glyphs are defined, there is a font definition data asset file that associates an ImageFrame glyph with a character value, as in a character in a string of text.

There is an object called a BitmapFont that stores a lookup table associating a glyph sprite with a character value. When rendering a string of bitmap text, this BitmapFont retrieves the characters needed for the string and draws on glyph at a time.

Rendering a string of bitmap text requires only one world matrix in Sea2D, because the displacement of different glyphs is calculated while iterating the characters in the string to draw and sent into the shader, which handles the individual glyph displacement without tampering with the world matrix.



The glyph is first displaced half its width. This displacement is sent to the shader and the glyph is drawn. And then the current text displacement moved half the glyph's width again before drawing the next glyph.

Ending Remarks

Sea2D also has useful math utilities, a 2D camera view system, and a basic scene graph system, along with the other features mentioned in this report. All of these features together give Sea2D much potential already for making 2D games.

Considerations in the future would include a tile system, which would optimally draw many static sprites in different locations at very high performance. Optimizations in general would be an ideal pursuit, especially in the graphics pipeline, which does a lot of state changes in order to draw sprites.

The scene graph system currently doesn't have the ability to dynamically reorientate the scene hierarchy, which could be a very useful feature in defining advance game scene logic. Currently, a branch of the hierarchy has to be deallocated before it can exist somewhere else in the hierarchy.

The current sprite-rendering engine draws every individual sprite without a sense of grouping. Sea2D would benefit from a sprite batch system that can allow sprite instances to be placed into groups that have different drawing orders and allow the enabling or disabling of rendering entire groups of sprites.

Sea2D started on OS X, but there is potential for the engine to be ported to other platforms. A Windows port could easily be made by syncing both an Xcode and Visual Studios project in version control and letting them use the same code base. Platform specific code could be included as separate files or wrapped in preprocessor declarations. With extra work and minor refactoring, a Linux version of Sea2D could also be possible, and even iOS.