Data Structure Explanation

Prior to the 2023-2024 capstone cycle, many of the variables and objects defined in the project were defined globally and had only a single instance. To remedy this and to allow expansion of the project's capabilities, many objects were moved into reusable structs. The 5 main structures in the project are:

- Pattern (globals.h)
- Pattern_History (patterns.h)
- Strip_Data (storage.h)
- Pattern_Data (storage.h)
- Config_Data (storage.h)

The final 3 structures, defined in storage.h, are utilized for both saving data and loading parameters. This reduces overall complexity as there are no need for additional structures for saving and loading, for example.

The Pattern struct contains the following items:

- index: The index for the given pattern.
- pattern_name: The name of the pattern that is presented to the user on the web app.
- enabled: If the pattern should be presented to the user on the web app.
- pattern_handler: The pointer to the function that generates the pattern.

There is an array of these structs, also in globals.h, that stores every single pattern currently run-able by end users. The description of all these patterns can be seen on the Nanolux wiki. To obtain and run a pattern, use code similar to the following:

```
mainPatterns[manual_pattern_idx].pattern_handler(&histories[0], config.length);
```

The Pattern History struct is essentially made up of a variety of objects used by various patterns. Most importantly, it contains an LED buffer. This LED buffer is intended to be used to house the current state of the LED strip. As each running pattern has an associated Pattern History object, this means each running pattern has a pattern buffer intended to be used only by it. This logic applies for all other variables associated with a pattern history, and also prevents conflicts around multiple patterns utilizing the same variables, such as the position and velocity of a virtual spring.

This is important as some patterns (for example, Hue Trail) utilize the previous state of the LED strip to generate the next state. So, if the current buffer is ever scaled or smoothed, this would mean the next iteration of the pattern would utilize the distorted data instead of the actual previous iteration's LED strip state.

The global Pattern History object is defined in main.ino:

```
Pattern_History histories[NUM_SUBPATTERNS];
```

The first of the 3 related storage structures is Strip Data. Strip Data contains visual and audio parameters that apply to the entire strip, including all running patterns. Strip Data also contains a number of instances of Pattern Data.

- **alpha**: How transparent the top layer is when running pattern layering.
- **noise_thresh**: The minimum threshold audio must cross to be considered non-zero during audio processing.
- **mode**: What mode the strip should be running in (strip splitting or pattern layering)
- **pattern_count**: The number of currently-running patterns.
- **pattern**: An array of Pattern_Data structures.

There are 2+ instances of Strip Data across the program, both defined within main.ino. The instance the main loop pulls from is

```
Strip_Data loaded_patterns;
```

There is an additional array of instances defined as:

```
Strip_Data saved_patterns[NUM_SAVES];
```

saved_patterns is the array that is saved to NVS. This process will be described in detail in a later section, but essentially, when the current Strip Data is to be saved to a slot, that particular array position is overwritten with the Strip Data in loaded_patterns. When a save slot is loaded, the inverse happens.

The second storage structure is Pattern Data. Pattern Data contains all data required to run a particular section of the LED strip.

- **idx**: The pattern index that this pattern is currently running
- **brightness**: A scalar applied to the entire pattern to decrease brightness (0-255)
- **smoothing**: How much of the previous LED strip state to include in the final processed output (0-175)

All instances should be contained within loaded or saved patterns.

The final storage structure is Config Data. Config Data stores configuration data such as strip length and target program loop times. It is sometimes referred to as "system settings."

- **length**: The number of LEDs on the current strip.
- **loop_ms**: The number of milliseconds to dedicate to a program loop.
- **debug_mode**: The debug state of the device (serial prints, simulator outputs).
- **init**: Set to 0 if loading from null data. Used to check if non initialized data has been loaded from NVS.