glsModel Demo

I rewrote the excellent example by TL Ford to create a more declarative method of embedding models and clicking on them in the model viewer.

Model Files

The `glsModel` requires that the model files be created with special values in the BCF (baseColorFactor). The glsModel will detect errors and report them if the file is not properly formatted.

Model Viewer

The Model Viewer can be placed anywhere on the HTML page by specifying an `id="name"` attribute. Any tag can be used, but the `<div>` is the best choice.

Also any `action` can be declared to be performed when a `material` is clicked. Currently only an `update` action is offered which will send user-defined HTML to a user-specified `<div>`. Other actions can be added easily.

Running the example program

Sadly, modern browsers don't allow running local file examples. You'll need to run a `server` of some sort. Python an PHP servers are readily available but will need to be installed. Alternatively, there is a plugin for VSCode that will launch a HTML file on a local server. See this: https://www.youtube.com/watch?v=Wd7cVmtiFUU

JSON data files

The definition of where to display a model and the actions to perform are defined in a JSON file. For more information on the format of a JSON file, visit: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

It's not that scary, I promise. The file begins and ends with curly braces `{}`. Every element within the file is a set of key/value pairs separated by a colon and ending with a comma. The format is recursive so a value can be either a string in double-quotes or another "object" in curly braces.

In the following sub-sections I reference `./data/buttonOnBox.json`

model-viewer section

```
{
    "model-viewer": {
        "src": "./images/buttonOnBox.glb",
        "id": "buttonOnBox",
        "alt": "alt text",
        "auto-rotate": "",
        "camera-controls": "",
        "ar": "",
        "ar-status": "",
        "control-index": 1,
        "div": "#buttonOnBox"
    },
...
}
```

This section defines everything about the `model-viewer` including its `id`, `src`, and which `div` to insert it in.

- src: the URL to the .glb file - I put all of mine in the `images` folder
- id: the `id="..."` name of this model-viewer
- alt: the alternate text if the browser cannot display it
- auto-rotate: whether the model will rotate automatically
- camera-controls: whether the model has camera controls
- ar: disabilities info
- control-index: this is the 'index' into the baseColorFactor to extract the hidden (secret) values.
- div: the id of the div to insert the `model-viewer` into. (You can use `query` specifiers `.` and `#` for class or id. If no query specifier is used, the `#` is assumed).

The mapping section

```
    "mapping": {
        "null": {
            "action": "update",
            "html": "<h2>This isn't a properly formatted model :/</h2>",
            "div": "#buttonOnBoxOutput"
        },
        "0": {
            "action": "update",
            "html": "<h2>Try Again</h2>",
            "div": "#buttonOnBoxOutput"
        },
        "1": {
            "action": "update",
            "html": "<h1 style='color:red'>This is the top button.  It does nothing useful a
            "color": "red",
            "div": "#buttonOnBoxOutput"
        },
        "0.99": {
            "action": "update",
            "html": "<h1 style='color:orange'>This is the middle button.  It screams as you
            "color": "orange",
            "div": "#buttonOnBoxOutput"
        },
        "0.98": {
            "action": "update",
            "html": "<h1 style='color:cyan'>This is the bottom button.  It wants to be the t
            "color": "cyan",
            "div": "#buttonOnBoxOutput"
        }
    },
```

For each value that the BCF can hold, an entry is made in the `mapping` section. When the user clicks on the model, one of these mapping sections is invoked based on the BCF value.

- The `null` value is invoked if the user hits outside the defined model.
- The `0` value is invoked if the user hits the model, but no special BCF value was defined.
- action: this is the action to take. It represents code in the `glsModel.js` file that will be invoked. Currently only one action is defined; `update`. This will be described below.

The rest of the values are `action-dependent` and can be considered parameters passed to the action function. As more actions are added, these values will be specific to them.

Color section

```
     "colors": {
        "red": [1, 0, 0, 1.0],
        "orange": [1, 0.45, 0.125, 1.0],
        "yellow": [1, 1, 0, 1.0],
        "green": [0, 1, 0, 1.0],
        "blue": [0, 0, 1, 1.0],
        "cyan": [0, 1, 1, 1.0],
        "violet": [1, 0, 1, 1.0],
        "black": [0, 0, 0, 1.0],
        "white": [1, 1, 1, 1.0]
     }
}
```

Since the `model-viewer` uses RGBA values to determine how things are colored, I've created a color map. The name ("red", "orange", etc. . .) can be used in the definition of `actions` (see above). The four values represent red, green, blue, and alpha (brightness). They must be between 0.0 and 1.0. You can add as many color entries as you like and name them however you see fit.

Actions

Actions are invoked as a result of a user clicking on some portion of a model. The `mapping` section defines what action is performed on a mouse-click. Each `action` takes its own set of parameters. Currently, only the `update` action is defined. This will update the HTML with a bit of text and flash the selected `material` object a specific color.

The `update` Action

```
"mapping": {
    "0.98": {
        "action": "update",
        "html": "<h1 style='color:cyan'>This is the bottom button.  It wants to be the t
        "color": "cyan",
        "div": "#buttonOnBoxOutput"
    }
```

In this example, when the user clicks on the `material` object that has a BCF value of “0.98”, the update action will be invoked with the following parameters:

- html: a line of HTML to be inserted into a div. This can be as simple as some text, or as complex as formatted text, images, styles, and classes. Special care must be taken to avoid double-quotes in the HTML string. If double quotes are needed a backslash-double-quote `\"` can be used, or the HTML glyph code `&quot;`.
- color: this is the color you want the `material` object to flash when the user clicks it. (If no color is specified, the `material` will not change color.)
- div: this is the `<div>` tag to update. The `div` in question must have an `id="..."` attribute.

To recap, when the user clicks on the `material` with the BCF value in the key (`"0.98"` in this case), the `cyan` `H1` text “This is the bottom button. It wants to be the top button” will be inserted at the `div` with an `id="buttonOnBoxOutput"` and the `material` will flash `cyan`.

HTML Stylings

You must include both the `model-viewer` and `glsModel` scripts in the `<head>` portion of your HTML.

```
<script type="module" src="https://unpkg.com/@google/model-viewer/dist/model-viewer.min.
<script src="./js/glsModel.js"></script>
```

To facilitate inserting the `model-viewer` and its associated model into your HTML page, you must add `div` tags with appropriate `id="..."` attributes. Here is a sample.

- DIV WITH ID

```
<div id="buttonOnBox"></div>
<div id="buttonOnBoxOutput"></div>
```

You must also add a call to the `glsModel.load()` function at **the very end** of your HTML.

- glsModel.load()

```
  <script>glsModel.load("./data/buttonOnBox.json");</script>
  </body>
</html>
```

Finally, you can add as many `model-viewers` to your HTML page as you like. Just repeat the `DIV WITH ID` and `glsModel.load()` bits of HTML wherever you want the model to show up.

Conclusions

I hope this script will make it easier to embed models and the `model-viewer` into your code.

If you need more `actions` than currently supplied, please feel free to reach out to me and suggest add-ons.

Greg Smith greg@agilefrontiers.com