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 names for each material. In Blender this is usually "Model.001," "Model.002," etc... But you can give more descriptive names if you like.

Older versions of this code used special values in the BCF (baseColorFactor). This is a tedious process and error prone. While `glsModel` still supports this mechanism, material "names" are the recommended route.

Model Viewer

The Model Viewer is a Google-supplied plugin and is documented here: https://modelviewer.dev/.

There is also an excellent model editor here: https://modelviewer.dev/editor/

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

Also any `action` can be performed when a `material` is clicked. Currently, three actions exist:

- `update` will send user-defined HTML to a user-specified `<div>`.
- `debug` will report the material.name to the user-specificed `<div>`.
- `error` will report errors in read to the user-specified `<div>`.

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 and PHP servers are readily available but will need to be installed. Alternatively, there is a plugin for VSCode that will launch an HTML file on a local server. See this: https://www.youtube.com/watch?v=Wd7cVmtiFUU

HTML Code

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>
<div id="buttonOnBox_output"></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.

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 enclosed in curly braces `{}`, 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

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

```
    "model-viewer": {
        "src": "./images/buttonOnBox.glb",
        "id": "buttonOnBox",
        "div": "#buttonOnBox_div",
        "control-index": 1,

        "alt": "alt text",
        "auto-rotate": "",
        "camera-controls": "",
```

```
        "ar": "",
        "ar-status": ""
    },

...
}
```

- src: the URL to the .glb file - I put all of mine in the `images` folder
- id: the `id="..."` name of this model-viewer - IT MUST BE UNIQUE (don't reuse model-viwer names in the same HTML page).
- control-index: this is the 'index' into the baseColorFactor to extract the hidden (secret) values (defunct).
- 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 following fields are passed to the `model-viewer`. For more information about these and other tags, see the model editor: https://modelviewer.dev/editor/

- alt: the alternate text if the browser cannot display it
- auto-rotate: whether the model will rotate automatically (remove this tag to stop rotation)
- camera-controls: whether the model has camera controls
- ar: disabilities info
- ar-status: disabilities info
- (there are many more options that are not documented here. see the model editor).

The mapping section

For each `material.name` in your .glb file, 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 `material.name` value as specified in your Blender file.

- The `null` mapping is invoked if the user hits outside the defined model.
- The `default` mapping is invoked if the user hits the model, but no mapping exists.

```
"mapping": {
    "default": {
        "action": "error",
         "div": "#buttonOnBox_output"
    },
    "null": {
        "action": "update",
        "html": "<h2>You missed the target</h2>",
        "div": "#buttonOnBox_output"
    },
```

```
        "Material": {
            "action": "update",
            "html": "<h2>Try Again</h2>",
            "div": "#buttonOnBox_output"
        },
        "Material.001": {
            "action": "update",
            "html": "<h1 style='color:red'>This is the top button.  It does nothing useful a
            "color": "red",
            "div": "#buttonOnBox_output"
        }
    },
```

Within each mapping is an action to take and some parameters to pass to it.

- action: this is the action to take. It represents code in the `glsModel.js` file that will be invoked. Currently,three actions are defined: `update`, `debug`, and `error`.
- div: this is the `id` of the div to send HTML to
- html: this is HTML code to insert into the div
- color: this is the color to "flash" the material that was clicked on.

Color section

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 `color` 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.

```
    "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]
    }
}
```

Actions

The `mapping` section defines what action is performed on a mouse-click. Each `action` takes its own set of parameters. Actions are invoked as a result of a user clicking on some portion of a model.

The `update` Action

The `update` action will display some HTML on the screen whenever the user clicks a bit of material in the model. The material will also change color to give visual feedback to the user.

```
"mapping": {
    "Material.001": {
        "action": "update",
        "html": "<h1 style='color:red'>This is the top button.  It does nothing useful a
        "color": "red",
        "div": "#buttonOnBox_output"
    },
```

In this example, when the user clicks on the `Material.001` object , 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 HTML 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 `name` in the key (`"Material.001"` in this case), the `red H1` text "This is the top button. It does nothing useful at all." will be inserted at the `div` with an `id="buttonOnBox_output"` and the `material` will flash `red`.

The `debug` Action

The `debug` action is used to display the `material.name` whenever the user clicks on a material. It is used when you don't know the `material.name` of materials in your model. This is most useful in the `default` mapping while you're testing the model.

```
"mapping": {
    "default": {
        "action": "debug",
        "color": "red",
        "div": "#buttonOnBox_output"
    },
```

In this example, when the user clicks on an unknown object, the `debug` action will be invoked with the following parameters:

- 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. It will be updated with the `material.name` of the object.

To recap, when the user clicks on the `material` the `debug` action will show the material's name in the specified div.

The `error` Action

The `error` action is used to display the `material.name` in red. It is similar to the `debug` action but generates red text as if it were an error. It is used inside the `default` mapping.

```
"mapping": {
    "default": {
        "action": "error",
        "color": "red",
        "div": "#buttonOnBox_output"
    },
```

In this example, when the user clicks on an unknown object, the `error` action will be invoked with the following parameters:

- 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. It will be updated with the `material.name` of the object in `red`.

To recap, when the user clicks on the `material` the `error` action will show the material's name in red in the specified div.

Naming Conventions

I tend to name objects in the HTML and JSON files after the `.glb` file to be displayed. So, if the name of the .glb file is `ATK02_JOB_2.glb`, I will do the following:

1. Name my JSON file `ATK02_JOB_2.json`
2. Append the `id` of the `div` where the model is displayed with `_div`: `<div id='ATK02_JOB_2_div'></div>`
3. Append the `id` of the `div` where the output is displayed with `_output`: `<div id='ATK_JOB_2_output'></div>`
4. In the .json file, set the `id` of the `model-viewer` equal to the name of the file. This helps to keep the `id` names unique.

```
"model-viewer": {
    "src": "./images/buttonOnBox.glb",
```

```
"id": "buttonOnBox",
"div": "#buttonOnBox_div",
"control-index": 1,
```

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