API for the g3.js library

Overview

This library allows you to efficiently display and interact with dynamic text, procedurally defined diagrams and images in 3D in XR, using an API that is similar in spirit to the canvas 2d API.

Usage

```
import { G3 } from "../util/g3.js";

export const init = async model => {
      …
      let g3 = new G3(model, draw => {
          // code to draw one g3 animation frame goes here.
      });
      …
      model.animate(() => {
         g3.update();
      });
}
```

Methods to set properties

```
draw.color(color)              // SET THE DRAWING COLOR
draw.font(font)                // SET THE TEXT FONT
draw.lineWidth(lineWidth)      // SET LINE THICKNESS
draw.textHeight(th)            // SET TEXT HEIGHT
```

Query methods

```
draw.distance(p)               // PERPENDICULAR Z DISTANCE TO A POINT
draw.finger(hand,i)            // POSITION OF USER'S FINGERTIP
draw.pinch(hand,i)             // CHECK FOR A PINCH GESTURE
draw.view()                    // EYE VIEW CURRENTLY BEING RENDERED
```

Methods to draw 3D objects independent of the viewer

```
draw.draw(path)                // DRAW A PATH IN 3D SPACE
draw.fill(path)                // FILL A PATH IN 3D SPACE
draw.line(a,b)                 // DRAW A 3D LINE FROM a TO b
```

Methods to draw objects in 3D that always turn to face the viewer

```
draw.draw2D(path,center)                          // DRAW A 2D PATH
draw.fill2D(path,center)                          // FILL A 2D PATH
draw.image(image,center,x,y,w,h,sx,sy,sw,sh)      // DRAW AN IMAGE
draw.text(text,center,alignment,x,y,rotation)     // DRAW TEXT
```

NOTE: Methods to set properties or draw objects return `draw`, so they can be chained.

Detailed description of all methods, in alphabetical order

`draw.color(color)` **// SET THE DRAWING COLOR**

These are all valid ways of describing red: `'red'` , `'#ff0000'` , `[1,0,0]`
You can also describe transparent red like this: `'#ff000080'` or `[1,0,0,.5]`

`draw.distance(p)` **// PERPENDICULAR Z DISTANCE TO A POINT**

Returns perpendicular distance along the eye gaze axis from the user's eye to point `p`.

`draw.draw(path)` **// DRAW A PATH IN 3D SPACE**

The `path` argument is in the form: `[ [`x_0`,`y_0`,`z_0`] , [`x_1`,`y_1`,`z_1`] , … ]`

`draw.draw2D(path,center)` **// DRAW A 2D PATH**

The drawing always faces the viewer.
The `path` argument is in the form: `[ [`x_0`,`y_0`] , [`x_1`,`y_1`] , … ]`
In the path, the x and y coordinates are generally in the range `[-1.0 … 1.0]`.
The path is centered at `center`, where `center` is a 3D point in space.

`draw.fill(path)` **// FILL A PATH IN 3D SPACE**

The `path` argument is in the form: `[ [`x_0`,`y_0`,`z_0`] , [`x_1`,`y_1`,`z_1`] , … ]`

`draw.fill2D(path,center)` **// FILL A 2D PATH**

The filled shape always turns to face the viewer.
The `path` argument is in the form: `[ [`x_0`,`y_0`] , [`x_1`,`y_1`] , … ]`
In the path, the x and y coordinates are generally in the range `[-1.0 … 1.0]`.
The path is centered at `center`, where `center` is a 3D point in space.

`draw.finger(hand,i)` **// RETURN POSITION OF A FINGERTIP**

`hand` must be either `'left'` or `'right'`.

`i` must be `0`,`1`,`2`,`3` or `4`, to indicate thumb, index, middle, ring or pinkie, respectively.
If using a controller, `draw.finger(hand)` returns the virtual ping pong ball position.

`draw.font(font)` **// SET THE TEXT FONT**

Valid arguments are `'Helvetica'`, `'Courier'` and `'Times'`.
The default font is Helvetica.

`draw.image(image,center,x,y,w,h,sx,sy,sw,sh)` **// DRAW AN IMAGE**

The displayed image always faces the viewer.
The displayed image is centered at `center`, where `center` is a 3D point in space.
Offset within the 2D image plane by `x` and `y`.
If `width` is positive, set the width of the image to `width`.
If `height` is positive, set the height of the image to `height`.
If either `width` or `height` is omitted, preserve the image's aspect ratio.
Optionally select a rectangle of pixels `sx,sy,sw,sh` as the source sub-image.

`draw.line(a,b)` **// DRAW A 3D LINE FROM a TO b**

`draw.lineWidth(lineWidth)` **// SET LINE THICKNESS**

`draw.pinch(hand,i)` **// CHECK FOR A PINCH GESTURE**

`hand` must be either `'left'` or `'right'`.
`i` must be `1`,`2`,`3` or `4`, for thumb touching index, middle, ring or pinkie, respectively.
Returns either `true` or `false`.
If using a controller, `draw.pinch(hand)` returns whether the trigger is pressed.

`draw.text(text,center,alignment,x,y,rotation)` **// DRAW TEXT**

The displayed text always turns to face the viewer.
The displayed text is centered at `center`, where `center` is a 3D point in space.
Multiple lines of text are specified by including `'\n'` within the text string.
Offset within the 2D plane of the text by `x` and `y`.
Optionally set `alignment` to `'left'` or `'right'`. Default alignment is centered.
Optionally specify `rotation`. Each unit represents 90º counterclockwise.

`draw.textHeight(th)` **// SET TEXT HEIGHT**

`draw.view()` **// EYE VIEW CURRENTLY BEING RENDERED**

Returns `0` if this is the left eye view, or `1` if this is the right eye view.