U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Interactive Graphics Systems

Interface

Alexandre Valle | DEI | FEUP

Pictures: two details of HCI from movie T*he Dark Knight (credits BLIND, Ltd.)*

Requirements

- WebGCF already contains a class to support interface management: CGFinterface
- The class uses dat.gui:
  - https://github.com/dataarts/dat.gui (File API.md contains detail on the API)
  - https://sbcode.net/threejs/dat-gui/
- What CGFinterface does:
  - Manages keyboard, touch, mouse input for the current camera
- What CGFinterface does not do:
  - Setup the 2D interface overlay with the help of dat.GUI (or other 2D interface technology)
- You will have to costumize the interface by creating a new class extending CGFinterface. The application and scene should reference the interface. CHECK main.js

Requirements

Excerpt from main.js

```
…
var app = new CGFapplication(document.body);
var myInterface = new MyInterface();
var myScene = new XMLscene(myInterface);

app.init();

app.setScene(myScene);
app.setInterface(myInterface);

myInterface.setActiveCamera(myScene.camera);
…
```

Excerpt from CGFinterface

```
…
init(application) {
   console.log("Initializing Interface");
   this.scene = application.scene;

…
```

By the end of initialization, scene should know interface and vice-versa

Interface initialization

```
// assuming this is an object extending class CGFinterface
// create a new dat.GUI object at initialization
this.gui = new dat.GUI();

// also call key management initialization. Method is described later
this.initKeys()
```

Key management

```
// assuming this is an object extending class CGFinterface
initKeys() {
    this.processKeyboard = function(){}; // unhook default processKeyboard method from CGFinterface
    this.activeKeys = {}; // will hold the key state
}

// process event in case of key pressed. Overrided method
processKeyDown(event) {
    this.activeKeys[event.code]=true;
};

// process event in case of Key released. Overrided method
processKeyUp(event) {
    this.activeKeys[event.code]=false;
};
```

At any time, inspection of this.activeKeys for a particular key code will return true or (false | undefined)

Checkbox creation and hook

// assuming *this* is an object extending class CGFinterface

// example for a ‘Lights visible checkbox’ that, onChange, calls methos

// method1 from this.scene

```
this.gui.add(this.scene, 'displayLights’)
        .name('Lights Visible’)
                .onChange(this.scene.method1.bind(this.scene));
```

Add folder, add checkbox, manage change

```
// Create a GUI folder
var folder = this.gui.addFolder('Lights’);

// Add items to the folder, set name, bind on change to a function ‘method2’ in scene
for(let i = 0; i < this.scene.lights.length; i++)
        folder.add(this.scene.lights[i], 'enabled’)
        .name(this.scene.lights[i].name)
                .onChange(this.scene.method2.bind(this.scene))
```

Add selection box and manage on change

```
// list of value labels that should be presented in the list. Necessarily string values
// example: [“perspective camera 1”, “perspective camera 2”, “ortho camera”]
var listValues =
// object containing an attribute that holds the selected value label
var someObject = this;
// in the next line we consider this to hold an attribute called selection
this.selection = “perspective camera 2”
// in the next command we add a list of values based on a) an array of strings ()listValues
// and b) the object that holds the selection (someObject.selection)
// .name and .onChange set the selection list label and the onChange callback
this.gui.add(someObject, ‘selection’, listValues)
        .name(‘theSelectionBoxLabel’)
        .onChange((value) => {
                var item = someArray[value];
                // do something with item
        });
```

Updating the active camera for interface purposes

// Assuming *this* is an object of a class extending CGFinterface

// the following function needs to be called on camera change, to hook mouse and keyboard input to a new active camera

// DO NOT FORGET TO DO THIS ON CAMERA CHANGE!!

```
this.setActiveCamera(some CGFcamera object)
```