IEVA - Adaptive 3d Interface

Eric Maisel - Pierre Deloor

Octobre 2023

Over the course of the various laboratory sessions, we will seek to implement a means of exploring a database in a way that is relevant to a user by means of a 3d metaphor. The aim here is to get to grips with some of the concepts associated with this 3d representation.

1 Appropriation

1.1 Launch program

The system is based on aclient-server architecture.

To start the server :

```
> python serveur/serveur.py &
```

To execute the client :

```
> firefox client/index.html
```

WARNING to render 3d objects correctly you need to modify the `security.fileuri.strict_origin_policy` option and set it to `false` (by accessing the `about:config` Web page).

1.2 Add objects to a scene

The client and server interact by sending messages (requests) from the client to the server. . This returns instructions acting on scene objects, contained in frames in JSON format.

Currently, the server can respond to two types of message. These reactions are implemented in the `serveur/serveur.py` file in the `init` and `click` functions.

Some of these objects are created in the world when the client is initialized (in the `init` function), while others are created in response to the occurrence of events (for example, in the `click` function).

Initially, you will work in the `init` function.

First, create the scene:

```
scene = Scene()
...
return jsonify(scene.jsonify())
```

The following instruction creates an actor known to the server as "toto" and of type actor".

```
scene.actor("toto", "actor")
```

This actor is completely abstract (no incarnation and not located in space) and impossible to render. We can associate an object with this actor. Here, a green sphere with a diameter of 20cm:

```
scene.actor("toto", "actor").add(sphere("toto",0.2, "vert"))
```

This instruction illustrates the architecture of an entity-component 3d application: it's made up of entities (actors) to which components are added to describe the shape and appearance of objects, as well as their behavior.
We'll see later that sections of component and entity code are regularly executed to modify the state of the entities.

Programming the world is therefore :

1. creating/deleting entities
2. Add/remove components to these entities (at initialization or during runtime)

See appendix for a list of components corresponding to 3d primitives that can be used in this work. .

For the moment, the 3d objects used are placed at the origin of the scene reference frame. Components are used to position and orient them in space:

```
a =scene.actor("lulu", "actor").add(box("lulu",3,1,1, "rouge"))
a.add(position(5,2,3)).add(rotation(0,math.pi/4,0))
```

Here we create an actor embodied by a red box. We apply

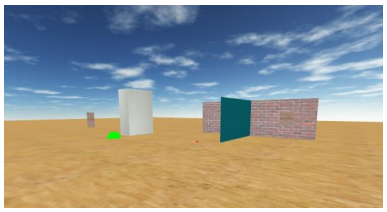
- a rotation parametrized by the following Euler angles: 0 around the x axis, $\frac{\pi}{4}$ around the y axis and 0 around the z axis.
- a translation of vector $(5, 2, 3)^t$

It can be useful to place objects not in relation to the scene reference frame, but in relation to an object in the scene. For example, we want to place a sphere on a box at some point in the scene.

```
scene.actor("boite01", "actor").add(box("boite01",1,1,1, "rouge"))
scene.getActor("boite01").add(position(2,0.5,3)).add(rotation(0,math.pi/4,0))

# Ajoût de la sphère

scene.actor("sphere01", "actor").add(sphere("sphere01",1, "vert"))
scene.getActor("sphere01").add(position(0,0.5,0))
scene.getActor("sphere01").add(anchoredTo("boite01"))
```



Exercise try executing the code given above without the last instruction. Then with the last instruction.

- What do you observe?
- What can you deduce?

Exercise using this principle hang two paintings (component created by calling the function `poster`) on a wall (component created by calling the function `wall`).

Exercise propose the instructions which allow you to obtain a scene corresponding to the image above:

2 A gallery of tables

We want to represent the paintings contained in a database in a 3d world. These tables will be placed as if they were in a gallery, i.e. a "corridor" in the context of this work.

2.1 Modeling

The gallery is represented in a "simplified" way by two parallel walls of equal length L and width l. On each wall are n different paintings.

Exercise propose a sequence of instructions to create the gallery.

2.2 Automatically populating a gallery

The `Musee` class defined in the file `serveur/serveur02.py` is used to load a database which contains a set of tables described by the following elements:

- the painter who created this painting
- the name of this table
- the year it was completed
- its height (in cm)
- its width (in cm)

Run the server implemented in the `serveur/serveur02.py` file. Its role is to randomly place representations of specified tables in the `base.json` file.

Exercise Propose a subclass of the `Museum` class which allows you to visualize a gallery dedicated to a particular painter. In this gallery you will only find paintings by this painter arranged in chronological order.

3 A museum

Using the available 3d primitives, provide instructions that allow you to model an architectural environment composed of identical square rooms (10m*10m*3.5m) placed in a matrix layout ($n * nmatrix$). Each room communicates with its 4 neighbors. Place paintings in these rooms.

Exercise propose a series of instructions allowing you to create such a 3d representation applied to the base `base.json`.

Exercise labels are associated with each work. They are accessible through the `tags` attribute of the `Tableau` class. Propose a procedure to classify the paintings in the different rooms according to the different attributes (painter, year of completion, tags, etc.). The passage from one room to one of its neighbors must follow an "understandable" logic.

4 Appendices

4.1 Appendix A: description of 3d primitives

- `sphere(name, diameter, material)`
- `box(name,dx,dy,dz, material)`
- `wall(name,dx,dy,dz,material)`
- `poster(name,l,h,image access)`

4.2 Appendix B: materials

- "rouge"
- "vert"
- "bleu"
- "blanc"
- "murBriques"
- "murBleu"
- "parquet"