Master Interaction « Fundamentals of Virtual Reality »

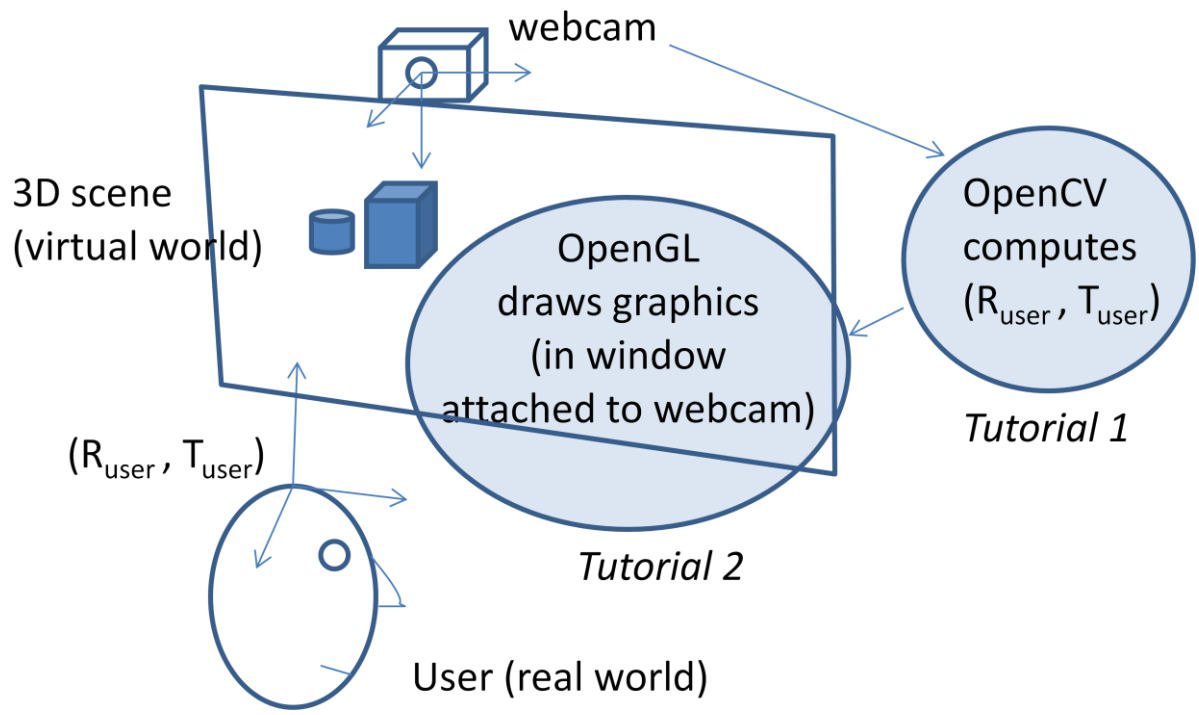
Tutored class 1

Frustum adaptation for Virtual Reality : a test case

Nicolas Ferey / Jean-Marc Vézien

The goal of this session is to illustrate how real-time data processing can be exploited to adapt 3D data visualization to the position of the user in front of the display.

Have a look at the corridor scene taken from the movie «Mission Impossible : Ghost Protocol » for a fun example on achieving this. In our more realistic case, a webcam attached to the visualization screen will detect the position and orientation of the user and feed it to a rendering program via message passing. The program will be written in Python, and will serve as an API to drive a dedicated image analysis routine implemented with the image processing library OpenCV. The rendering stage , performed in OpenGL is also written in Python and will be the subject of tutorial 2. Below is a figure summarizing the situation :



Let's get started ! The working directory should contains :

- A python file "tracking.py" containing the working script
- A set of 3 library files : `liblo.dll, liblo.pyd, pthreadVC2.dll`. Keep them in your working directory.
- A sample 3D file "glasses.txt" containing the description of the marker set detected by the webcam

The script will be run by executing :

```
>python tracking.py glasses.txt
```

The user detection will be color-based :

- a routine called "findBlob" detects image blobs of a given color. These color patches (four of them) form a distinctive 3D pattern that the user wears, typically on his/her glasses. Once detected , the centroids of the blobs are matched against the model provided in the 3D file, and the user position and orientation are recovered using the POSIT algorithm (see optional track on Augmented Reality). This position information is sent to a rendering program to draw a basic scene.

The `tracking.py` python script deals with the detection phase and thus computes a matrix named `pos_mat` , that it sends to the rendering program with calls to the `liblo` API.

Please proceed as follows :

1) Look at the script and try and figure out the general organization : which routine calls which and in what order.
2) Try and find out exactly how image processing is done. Note that all the analysis is performed on Hue-Saturation-Value images, or HSV. See the attached note on that subject. What are the parameters and how do you change them ?
3) Identify the routine that computes 3D information. What does it take as input ? As output ? Have a look at the `OpenCV` documentation describing that routine.
4) We don't send out the raw position matrix itself, but some by-products of it. Identify them and clarify how they are computed (make drawings !). Below is a figure summarizing the reference frames attached to the screen and the user (The user reference frame is arbitrary, you can choose a different one if you want).
5) Locate the calls to the `liblo` routines. Search on Internet how this API works.
6) Now make a 3D target with the supplied material and affix it to a pair of glasses. Create an appropriate 3D descriptor file and see if you can get tracked. Make the script output your position with respect to the screen.