# Project `MATRIX` : On-set 3D Reconstruction Tool

Louis Viot
louis.viot@etu.enseeiht.fr

Department of Applied Mathematics and Computer Science, ENSEEIHT

**Keywords** : *VORTEX, Mikros Image [**?** ], POPART, MATRIX, On-set 3D Reconstruction Tool, 3D, openMVG, Cinema, CGI, Reconstruction*

## Abstract

This project, named MATRIX [**?** ] but officialy named "On-set 3D Reconstruction Tool", aims at developing a tool to do a 3D reconstruction of a scene from pictures of this scene. The reconstruction is based on an already existing library called openMVG [**?** ] that people are already using for the real time previzualisation of special effects during the shooting of a movie. What we did was building a graphical user interface around this library using different languages as Qt [**?** ], QML, PyQt, Python, C++ so that user can automatize the whole reconstruction process. While reading this abstract, you have to keep in mind that this project wasn't about ending up with something really stable and usable, but more about building some sort of a prototype that people could use to rebuild a real software. Indeed, what we did is now almost usable, the user can launch a 3D reconstruction and import pictures from his computer or his camera, but more important, we have build a solid architecture and a re-usable code and that what the project was all about.

## 1 Context

Right now, on a shooting for the cinema, people in charge of special effects would really benefit from a tool allowing them to have the real time pre-visualization of special effects. That is what the European Project POPART [**?** ] is all about. Our project is casted in this context : we had, first, to collect a dataset of images from the scene and do the 3D reconstruction, and then, build a graphical user interface to automatize the whole process. The 3D reconstruction of a scene from images is a well known problem in computer vision so there are already existing library : we have used the openMVG library. openMVG uses tools that implement the "Structure From Motion" techniques. Those techniques are working that way : from a dataset of images, openMVG computes the matching points between images and then computes the 3D scene (coordinates of cameras and points in the scene). So the 3D reconstruction is not a real problem for us, the main goal of the project is to build the GUI that automatize the reconstruction because today, people on a shooting have to use openMVG as a command line and to launch again the reconstruction each time they have a new picture. So the GUI must allow user to automatically import pictures, then automatically launch the reconstruction, and then do a 3D rendering of the scene. The project initially aimed at improving the openMVG library so that the user could add pictures while a reconstruction is being done, preventing the reconstruction to be relaunched and started from scratch. But this was totally out of our means and we wouldn't have had the enough amount of time.

## 2 Difficulties and Choices

First of all, before even thinking about programming language, we had to think of a solid and re-usable architecture for our application. Because we had to build a GUI that would launch openMVG executable, we quickly choose to use the model/view/controller pattern. This pattern is pretty simple to understand : the user uses a controller which will manipulate the model which will update the view that the user sees. In our context, the model correspond to the openMVG executables for example. What we actually choose was to modify a bit this pattern to perfectly fit our needs : we would merge the controller and view together, we will explain later why.

Once we had built a solid architecture, and that the client confirm the architecture was what he needed, we thought about programming languages. We firstly choose to use C++ for the model code for two reasons : because nobody knew at this time how to use this language and because it is a very powerful and low level language. To create the GUI, we choose

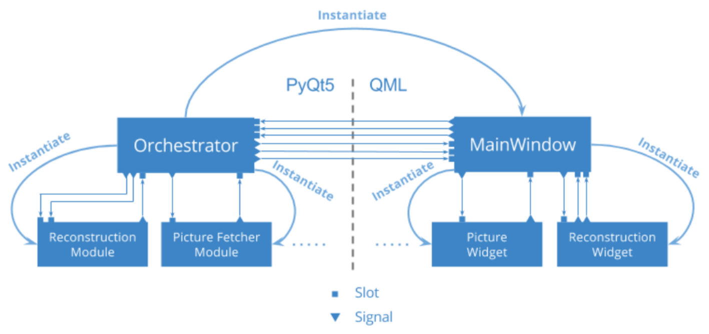


Figure 1: Architecture of the application (`PyQt5` for the model, `QML` for the view)

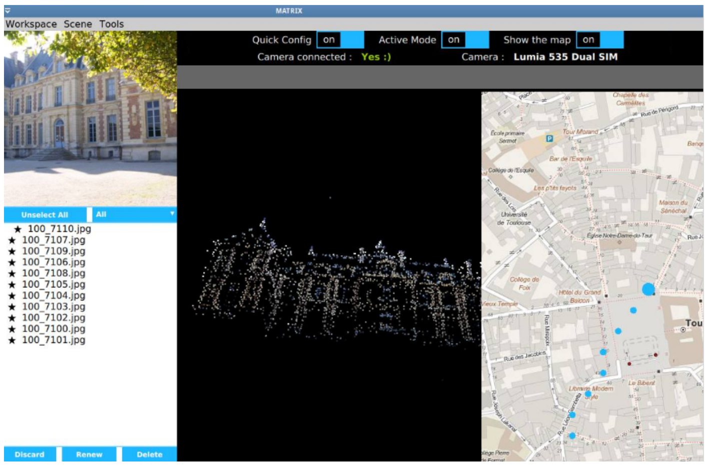
to use Qt, which is a "cross-platform application for developpers using C++" according to their website, so that it perfectly fits our needs. But later in the project, the client told us that he would rather us to use Python instead of C++. Python was also a challenge for us because no one knew it at this time. After choosing python, we had to restart almost everything, forget what we learned and start over. That was a important milestone in our project. We choose to stay with Qt and QML for the GUI, but we had to adapt it in order to make it work with Python : that's why we used PyQt, a "binding" between Qt, coded to use with C++, and Python, that allows us to communicate between Python et Qt.

When we were set on the programming languages, we did a prototype of our application to test communication between the model, in python, and the view/controller, in PyQt and QML. The power of PyQt and Python is the communication in signal/slot : one can emit signal in a module and another module can receive this signal in it's slot. So module, in PyQt or Python, can exchange any objects through this system. Throughout the project, we always kept in mind that our goal was to build a easily reusable application. That's why we used the architecture described above but adapted to the signal/slot communication (see figure **??**) : only two "modules", the orchestrator and the main window, can communicate in our application so that whenever a developper wants to add a GUI or a model module, he only has to connect it to the orchestrator or the main window. Modules are then connected to either the orchestrator or the main window, and they handle the communication.

The last really important milestone we had was the 3D renderer : because python is more high level than C++, there are not really any optimised openGL libraries for python. But one important application requirement was a performant renderer to render big point clouds. So instead of using openGL with python, we used openGL with C++ and then we created a QML plugin to import in PyQt. Though we first thought that this was a huge problem, it turns out to be a really good example on how our application was highly cutomizable and modular : one can easily import plugin coded in any language they want as long as they convert it into QML plugin.

## 3 Results and possible improvements

Results are really satisfying given the fact that we knew absolutely none of the languages we used at the beginning and that the application is supposed to be a prototype. We ended up having more than 7000 lines of code in the languages described previously. There are still a few improvements to be made but for the moment one can import pictures from either his camera or his computer, sort them, delete them, launch a 3D recontruction on those images, and then see the result on the renderer and see cameras positions on a map. We also implement interaction between the map and the photo list : if a user click on a picture, it will center the map on the GPS position of the camera used to take this picture. The user can also manage a workspace, even multiple workspaces, in which he can add scenes. One scene has one or multiple pictures, and it is from one scene that the user can launch a 3D reconstruction. The application comes with



Quick Config on    Active Mode on    Show the map on

Camera connected : Yes :)    Camera : Lumia 535 Dual SIM

★ 100_7110.jpg
★ 100_7107.jpg
★ 100_7109.jpg
★ 100_7106.jpg
★ 100_7108.jpg
★ 100_7105.jpg
★ 100_7104.jpg
★ 100_7103.jpg
★ 100_7102.jpg
★ 100_7100.jpg
★ 100_7101.jpg

Figure 2: Screenshot of the final application

installation instructions and even a documentation (see [**?** ]). To conclude on those results, we have, at the end of this project, a pretty good prototype and our clients were pretty satisfied with the result. Here is a screenshot of the application :

In further version, we could add the same type of interaction between the 3D renderer and the photo list. We could also launch the reconstruction in another thread to not block the entire application while a reconstruction is processed.