MMI - Project Report **CammVJ**: *Collaborative And Multi-Modal VeeJay*

Ilya Cassina,
MultiModal Interfaces,
Spring 2010, Université de Fribourg

June 5, 2010

1 Idea

The basic concept is to let a group of users collaborate together in building and animate a dynamic 3D scene to be rendered in real time. The scene is a set of 3D objects with different properties.

A group of users manipulate the objects on the scene by moving and rotate fiducial markers on a table, a users interface is projected on the work surface to give them a feedback of their actions. This group of users control all the properties of the 3D objects of the scene.

Another user stays in front of a camera handling two objects of different color, one for each hand. The two objects are tracked and their coordinates are used to manipulate the camera of the rendered 3D scene.

2 Architecture

2.1 Overview

The application is divided in three parts:

- **Users Interface**: the interface projected on the table;
- **Hands Tracker**: the software tracking hands movements;
- **Render Server**: the software rendering the scene.

The architecture overview diagram can be seen in 1. The **Users Interface** is built with `pymt`[6], a `Python`[7] library for multitouch applications. It is used to handle the input from `reacTIVision`[8], a fiducial tracker software. The **Hands Tracker** is implemented using the (`OpenCV`[2] based) `Movid`[1] framework, a very impressive tool that allows building recognizers by connecting different modules together, without writing a single line of code. The **Render Server** is also written in `Python` using `pymt`.

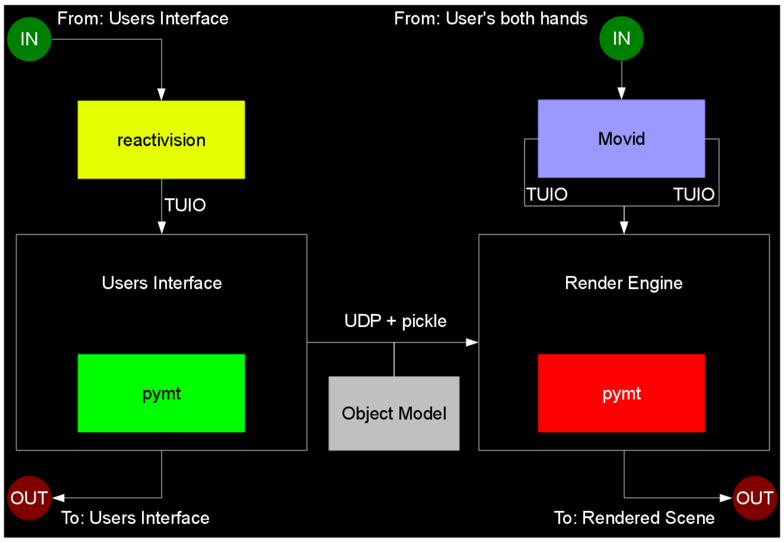

Figure 1: Architecture overview

2.2 Communication

The communication protocol used is `UDP`, in particular the `TUIO`[9] protocol (itself based on the `OSC`[4] protocol):

- from `reacTIVision` to `pymt`'s **Users Interface**;
- from `Movid`'s **Hands Tracker** component to `pymt`'s **Render Server**.

UDP (but not TUIO) is also used for passing the objects' state from the **Users Interface** to the **Render Server**. Both applications use the same `Python` class for the objects, that information is extracted from the object, passed to `pickle`[5] (a marshaler), transmitted, re-passed through `pickle` and the object is rebuilt on the other side.

This distributed architecture allows to use different computers for the different tasks. It is particulary important, because each component is very resource consuming (the interface and the rendering engine uses graphic acceleration to draw on the screen, the hands tracker also needs a lot of resources for image recognition). Moreover, it can be fun to have the people handling the table on one side of the room, while the person manipulating the camera is sitting on the opposite side of the room and four other computers are rendering the same scene in other rooms! It is also a necessity: in order to project the users interface, give a feedback for the hands tracking and render the scene on a single computer, the latter will need three video outputs (i.e. it is impossible to do it with a laptop, as two graphic cards are a minimum).

3 Implementation

3.1 Users Interface

The user interface is written with `pymt`[6], a multitouch widget library. It allows to build complex user interfaces with the traditional widget paradigm: layers, parent/children etc. What is interesting is that it allows many input providers (mouse, wii, touch, tuio, among others) and, of course, it is not limited to a single input at a time. Another positive aspect (not used in this project) is gesture recognition. I didn't use it because it is implemented for "touches", i.e. a set of points with a begin and an end, while fiducial markers usually stay on the table. Finally, a widget can be moved, rotated and resized, as `pymt` is based on the `OpenGL`[3] library for the rendering part.

Classes The components of the users interfaces are:

- `WorldObject`, class containing information about an object (position, color, shape, size, ...);
- `ActionItem`: a class that can be "actioned";
- `MenuItem`: a class extending `ActionItem`, an item of a `Menu`;
- `Menu`: a class extending `MenuItem`, containing many `MenuItems`;
- `MenuSystem`: a class extending `ActionItem` containing a root `Menu` and methods to navigate through a menu hierarchy;
- `Circular`: a class providing some methods to translate "bottom-left" to "orgin" relative coordinates.
- `CircularWidget`: a class extending from from `MTScatterWidget` and `Circular`, implementing drawing itself and childrens from the origin (instead of from bottom-left);
- `CircularMenuItem`, `CircularMenu`, `CircularMenuSystem`: classes extending from `CircularWidget` and the corresponding menu class, implementing their visual part.
- `CircularMenuCombobox`, extending from `CircularMenu`, for selecting items from a menu (i.e. items cannot be menus themselfs);
- `CircularMenuSpinner`, extending from `CircularMenuItem`, for manipulating a numeric value.
- `ObjectMenu`: extending from `CircularMenu`, implementing the main menu for a single `WorldObject`. Contains the following submenus:
  - `ObjectMenuShape`: combobox for selecting the shape;
  - `ObjectMenuMode`: combobox for selecting the rendering mode (wire, solid, both);
  - `ObjectMenuColor`: menu containing four `ObjectMenuColorSpinner` (a different flavour of circular spinner, more colorful!). The four spinners control the *hue*, *saturation*, *light* and *alpha channel* of the object's color.

  - `ObjectMenuRotation`: menu containing six spinners, for controlling the rotation angle around the three axes or the respective angular speeds;
  - `ObjectMenuScale`: menu containing four spinners: three of them controlling the scale factor along each axis, the fourth controlling the overall scale factor of the object;

- `Tracker`: class following a particular fiducial marker, keeping information about it. Extends `CircularWidget`;
- `GestureTracker`, extending from `Tracker` and containing a "tracked" widget: an `ActionItem` to be manipulated by the tracker gestures.
- `CircularMenuSystemGestureTracker`: simply a `GestureTracker` requiring a `TrackedCircularMenuSystem` as the tracked widget;
- `TrackedCircularMenuSystem`: extending CircularMenuSystem. Draws feedbacks about available gesture-actions depending on its state;
- `ObjectsTracker`: extending `CircularMenuSystemGestureTracker` implementing methods to add and remove `WorldObject` to the tracked widget;
- `ObjectsTrackedMenu`: extending `TrackedCircularMenuSystem` and implementing the specific logic of the main menu (i.e. 'LEFT' and 'RIGHT' actions *in primis*, the ability to change the `WorldObject` being manipulated);
- `ObjectsMainMenu`: a `CircularMenu` that will contain `ObjectMenu`s as its items; implements methods to add and remove `WorldObjects`.
- `Shape`s and `ShapeRender`s classes implements the `OpenGL` commands to render their corresponding shape.
- `ShapeViewer` draws or render a `WorldObject` according to its position, rotation, color, scaling factors and `Shape`.
- `UI` is the main class managing the `Users Interface`: instantiating new `WorldObjects` when new fiducials are on the table, or instantiating new `ObjectsTracker`s when fiducials with a specific id are on the table; updating their position when they move and deleting them when removed from the table. On changes it also send a message to the `Render Server` component with the needed informations.

3.2 Hands Tracker

The hand tracker consists in a particular configuration of `Movid` software, as it can be seen in Figure 2, showing the "pipeline" network of the recognizer.

The `Camera` module is in charge of open a video from a camera. We connect its output to the `Mirror` input, because this configuration is intended for a webcam mounted in front of the user. The mirrored output is then connected to two `YCbCr` modules, one configured to recognize **yellow** and one for **green**. We build then the same pipeline for the two colors: first the `Smooth` module, that, as its

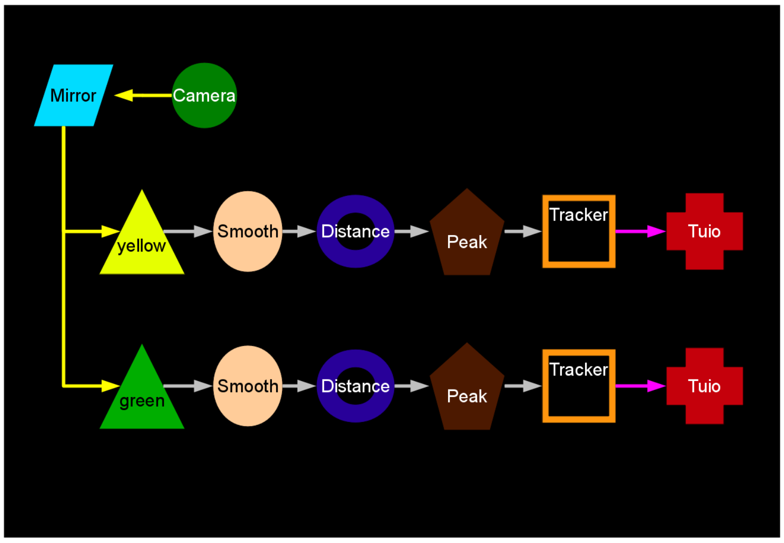

Figure 2: Movid's pipeline for two-colors tracking.

name suggests, smooths the resulting image; secondly the `DistanceTransform` module, that computes the distance from each pixel to the nearest contour, drawing a sort of a "shape". The following module is `PeakFinder`, that recognizes peaks, discarding the rest. Its output is connected to the `BlobTracker` module, that recognizes "blobs" and generate its corresponding "touch" (x,y coordinates and eventually it's size). Finally we connect the touches stream to the `Tuio` module, in charge of sending TUIO messages to the specified host and port. This powerful two-colors tracker is already finished!

3.3 Render Server

The `Render Server` is composed in two parts:

- `Object Receiver`: UDP server in charge of receiving messages from the `Users Interface` and keep the list of "live" `WorldObjects` up-to-date;
- `Render Engine`: `pymt` widget which extensively uses `OpenGL`, drawing the objects list according to their properties and rotate/scale the scene according to the `Hands Tracker` inputs.

4 User Guide

4.1 Users Interface

4.1.1 Basic concepts

The Users Interface is manipulated through *Fiducial Markers*. We differenciate two kind of fiducial markers:

- **Object markers**: used to control objects on the scene;
- **Controller markers**: used to control the menus.

Controller markers are those with a fiducial id 4,5,6 and 7, all the others are **Object markers**.

Objects An object represent a 3D shape in the scene space, it has the following properties:

- **Position**: *x, y, z* coordinates;
- **Size**: scale factors on the three axes and an overall scale factor;
- **Rotation**: degrees around the three axes, eventually also the rotational speed;
- **Color**: *hue, saturation, light* and *alpha* components;
- **Shape**: type of shape: *cube, cone, sphere*, etc;
- **Mode**: rendering mode: *solid, wire* or *both*.

The **position** is controlled directly with the **Object markers**: the (x,y) coordinates by moving the fiducial on the table surface, the y coordinate by rotating it.

In figures 3 you can see a possible effect of placing eight Object markers on the table: a (pseudo) random 3D shape with random color will be tied to each marker.

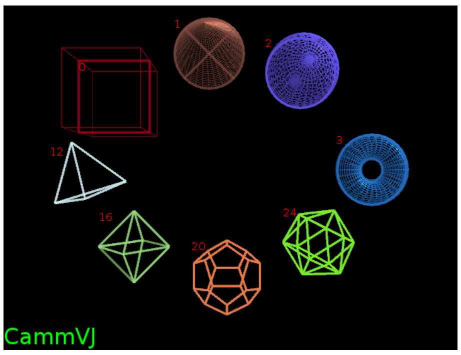

Figure 3: (Almost) all the shapes.

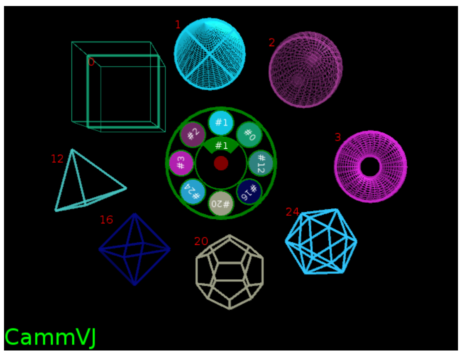

Figure 4: Shapes with the main menu.

Controllers Controllers implements a menu system to control all other properties of objects. The Main Menu of a Controller consists in a (circular) list of all the objects on the table.

When an object is selected from the Main Menu, the Object Menu substitutes the Main Menu. The Object Menu has five submenus, namely **Shape**, **Mode**, **Color**, α, β, γ (to access the “Rotation” properties) and the **Size** submenu.

Actions The controller knows about six different actions: *up*, *down*, *left*, *right* and *increase*, *decrease*. The first four actions are triggered by moving the controller marker on one of the four corresponding regions and wait a little inside the region. When the action is about to be performed, a yellow circle blinks. Increase and decrease operations are triggered simply by rotating the fiducial clockwise or counter-clockwise respectively.

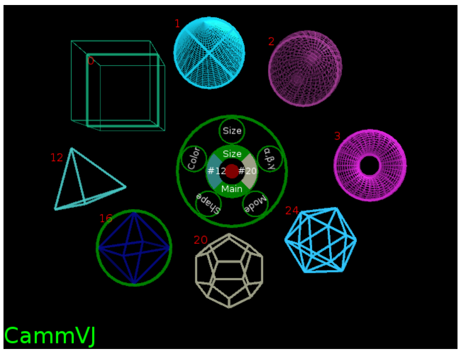

Figure 5: Shapes with an Object Menu.

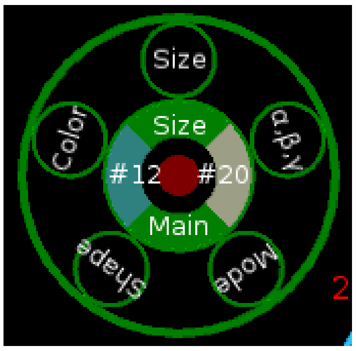

Figure 6: An Object Menu.

In figures 5 and 6 the controller has all the six actions enabled. The *increase* operation will rotate the menu counter-clockwise and select the following item, the *decrease* operation will rotate the menu clockwise and select the preceding item. The *down* operation will close this menu and go back to Main Menu, while the *up* operation will activate **Size** menu of the current object, as it is the one currently selected.

Action	Menu	MenuItems
up	enter selected menu	activate/expand
down	go to parent menu	deactivate/collapse
incr.	next item	increase value (spinner)
decr.	prev. item	decrease value (spinner)
left	track prev. object (unless in Main Menu or only one obj.)	
right	track next. object (unless in Main Menu or only one obj.)	

Table 1: Actions regions and operations for Menu and MenuItems.

In Table 1 you can see the default behaviour of each action. If the current widget is an object menu (i.e. an object is selected, the controller is "tracking" that object) and there are other objects around, then the *left* and *right* regions are activated. Their action is a special one: it allows you to activate another object while keeping the position on the menu, i.e. if you were manipulating the hue color component on an object X, when you pass to next object using *right*, say Y, you will be on the hue spinner of Y object.

4.1.2 Setup

Unpack the `pymt` library :
Unpack the `3rdparty/pymt-latest.tar.bz2` archive into the `libs/` directory.

Unpack the `reacTIVision` application :
Unpack the `3rdparty/reacTIVision-1.4-src.tar.bz2` archive into the `apps/` directory.

Unpack the `TUIO_Simulator` application :
Unpack the `3rdparty/TUIO_Simulator-1.4.zip` archive into the `apps/` directory.

4.1.3 Usage

To launch the **Users Interface**, you need first to launch either `reacTIVision`, if you are using a real table, or `TUIO_Simulator`, if you are just testing it out or don't have such equipment.

Launch reacTIVision

1. Go inside the "`apps/reacTIVision-1-4-src/`" directory;
2. depending on you Operating System enter the `linux/`, `macosx/` or `win32/` directory;
3. launch `reacTIVision` and calibrate it[1].

If you plan to run the **Users Interface** on a host different than the one running `reacTIVision`, please adjust the configuration file accordingly (`reacTIVision.xml`).

Launch TUIO_Simulator

1. go inside the "`apps/TUIO_Simulator/`" directory;
2. launch the simulator (a Java `jar`): `java -jar TuioSimulator.jar`.

Note: to rotate markers, you need to hold down the right button while moving the mouse. Also, as this is a Java application, it does not support multiple mouse inputs. If you plan to run the **Users Interface** on a host different than the one running `TUIO_Simulator`, use the `-host` commandline option to specify the correct host where to send TUIO packets.

Launch CammVJ Users Interface Now that a TUIO client is up and running, it's time to launch the **Users Interface**. Type (unix shell only)[2]:

```
export PYTHONPATH="libs/pymt"
python src/cammvj-ui.py -p react:tuio,0.0.0.0:3333
```

Note that if you are running the **Render Server** application on another machine, you should edit the source file and change the `HOST` variable, i.e. change "`127.0.0.1`" to the ip of the actual machine running the **Render Server** at line 20 of the `src/cammvj-ui.py` file:

[1]You can find information about calibration on `reacTIVision` Home Page: `http://reactivision.sourceforge.org`

[2]On Windows, you should specify the %PYTHONPATH% environnment variable using some graphical tool... and slashes become backslashes.

```
HOST, PORT = "127.0.0.1", 9999
```

What we are saying is:

1. add "`libs/pymt`" directory to the search path of Python;
2. launch the python program named `cammvj-ui.py` inside the "`src/`" directory;
3. use a TUIO input provider called "`react`" listening on port `3333` of any network interface.

The **Users Interface** window will appear. To launch it in fullscreen mode, add the "`-a`" switch, i.e.:

```
export PYTHONPATH="libs/pymt"
python src/cammvj-ui.py -p react:tuio,0.0.0.0:3333 -a
```

To exit the program, simply press the `escape` button.

4.2 Colors tracker

4.2.1 Setup

Compiling `OpenCV-2.0.0` :

1. Unpack the `3rdparty/OpenCV-2.0.0.tar.bz2` archive into the `libs/` directory.
2. Go inside the `libs/OpenCV-2.0.0/` directory and follow the instruction[3] specific to you Operating System to build the library.

Preparing Movid :

1. Unpack the `3rdparty/Movid-latest.tar.bz2` archive into the `apps/` directory;
2. Go inside the `apps/Movid/` directory and follow the instrction[4] specific to your Operating System to build the application;
3. Copy the configuration file from `doc/left_right-colortrack.txt` to `apps/Movid/presets/` directory.

[3]`http://opencv.willowgarage.com/wiki/InstallGuide`

[4]In the `README` file

4.2.2 Usage

1. Launch the `Movid` application with the provided configuration[5], i.e.:

```
./movid -n -l presets/left_right-colortrack.txt
```

2. Take two little objects[6], one **yellow** and the other **green**. Hold the **yellow** with your **left** hand, and the **green** with your **right** hand.

In the window you should see some white zones corresponding to your objects, and two circles tracking those zones. When you see the circles, it means that the tracking is working.

Controls :
Your **left** hand is controlling the following parameters in the **RenderEngine**:

- **x** coordinate: rotation speed of the camera aroung the z axis;
- **y** coordinate: zoom.

Your **right** hand is controlling those other parameters:

- **x** coordinate: rotation speed of the camera around the y axis;
- **y** coordinate: rotation speed of the camera around the x axis.

4.3 Render Server

4.3.1 Setup

If not already done, unpack the `3rdparty/pymt-latest.tar.bz2` archive into the `libs/` directory.

4.3.2 Usage

Launch CammVJ Render Server :

```
export PYTHONPATH="libs/pymt"
python src/cammvj-render.py -p left:tuio,0.0.0.0:3334 -p
    right:tuio,0.0.0.0:3335 -a
```

As for the **Users Interface**, to exit the application simply press the `escape` button.

[5]If you are running the `Render Server` on another machine, change the configuration file accordingly, i.e. change `127.0.0.1` part of "`pipeline set leftTuio ip`" and "`pipeline set rightTuio ip`" lines to the ip of the actual machine.

[6]I used two felt-tip colored caps.

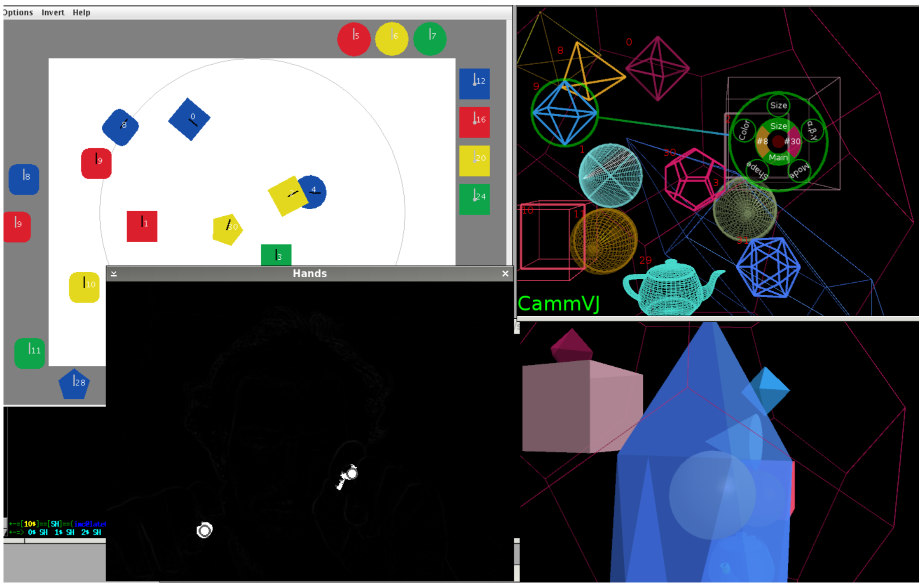

Figure 7: All the software running at the same time. Top-left: `TUIO_Simulator`, top-right: `cammvj`'s **Users Interface**, bottom-left: `Movid`'s **Hands Tracker**, bottom-right: `cammvj`'s **Render Server**.

5 Conclusion

Developing in `Python` was really exciting, using the `pymt` library was quite hard at the beginning, but after a while it was very natural and intuitive. And powerful. I would like to thank the main `pymt` developers that took the time to clarify my doubts, helped me understanding the library and fixed some bugs in very short time. To all the good people in the `#pymt IRC` channel on `irc.freenode.org`: "Thank you for the great work and the invaluable help you gave me!". I would also like to thank the developers of the great `Movid` framework, but it happens that they are... well, the very same bright persons[7], so thank you again!

Possible improvements

- **Users Interface** multicasting to several **Render Servers**, each (potentially) using a different **Hands Tracker** client, to project the same scene with different cameras and/or on different projection spots;
- make the `Users Interface` aware of the `Hands Trackers` clients, and make it possible to attach those inputs to some properties of the objects (i.e., changing colors, size, etc..);
- implement some kind of music input for `pymt`, capable of transforming music (frequencies, peaks, beats) into useable data, i.e. x, y coordinates or actions;

[7]Their blogs are on `http://pymt.txzone.net/planet/`, check them out!

- pixel shaders effects, controlled either via the **Users Interface**, `Movid`'s image recognition or the eventual music input.

References

[1] Movid. `http://movid.org/`. Modular Open Vision Interaction Daemon, modular tracker based on `OpenCV`[2].

[2] Opencv. `http://opencv.willowgarage.com/`. Open Computer Vision software library.

[3] Opengl. `http://www.opengl.org/`. Open standard environnment for developing portable 2D/3D applications.

[4] Osc. `http://opensoundcontrol.org/`. Open Sound Control protocol.

[5] Pickle. `http://docs.python.org/library/pickle.html`. A python object serialization module.

[6] Pymt. `http://pymt.txzone.net/`. Python MultiTouch library, handles multiple inputs and renders with pygame/pyopengl.

[7] Python. `http://www.python.org/`. A powerful, dynamic and object oriented programming language.

[8] reactivision. `http://reactivision.sourceforge.org/`. Fiducial markers tracking software.

[9] Tuio. `http://www.tuio.org/`. TUIO - a protocol for table based Tangible User Interfaces, based on `osc`[4].