MA Petite Application

Carlos Grandón <cobra@inf.utfsm.cl>

July 8, 2004

1 Introduction

MAPA is an application for showing boxes in 2D or 3D. The main object is to support visualization results from domain filtering techniques. The main features are:

- Boxes visualization by criteria.
- Changing color/transparency by criteria.
- Taking objects pictures.
- Making video from screen shots sequences.
- Changing visualization window size.
- Running external commands or files in Python language.
- To show/hide axis.
- Reset original camera position.

The last version is available in:
`http://www-sop.inria.fr/coprin/personnel/Carlos.Grandon`

2 Philosophy and execution

MAPA is written in **Python**[7, 2] language and uses the **VTK**[5, 3] library for displaying objects.The GUI is based in **Tkinter**[4, 6], a **TK**[1] library for Python.

MAPA is an executable script that gets an input file as argument. The input file describes a set of boxes (each one in a different line) in the following format:

$$\begin{array}{ccccc} < box_1 > & < prop_1 > & < prop_2 > & ... & < prop_n > \\ < box_2 > & < prop_1 > & < prop_2 > & ... & < prop_n > \\ \vdots & \vdots & \vdots & & \vdots \\ < box_m > & < prop_1 > & < prop_2 > & ... & < prop_n > \end{array}$$

where $< box* >$ represents an object in 2D or 3D, and $< prop* >$ is an integer that encodes a box property[1].

Table 1 shows an example of an input file. There are three boxes in a 3D space. Each box has two properties.

$([0,1];$	$[0.5,10];$	$[0,1])$	1	1
$([1,3];$	$[0,1];$	$[0,2])$	2	1
$([0,1];$	$[1,2];$	$[0,1])$	3	2

Table 1: Example of input file

The boxes are enumerated in the same order that these on the input file followed by their properties. For example, in the first box (line 1), the first and second properties are equal to 1; In the third box (line 3) the second property is equal to 2.

In the application, for each box an actor is defined. An actor will represent the box in scene. The properties allow to select one or more actors based on criteria (see section 3.1).

3 Menu options

The application has nine option buttons. The first six show/hide sub-menus, and the others perform specific tasks (show/hide the axis, reset camera and exit program).

3.1 The Actor Menu

The actor menu allows to show/hide actors from scene. It is necessary to use the *criteria* for selecting objects. The properties of boxes are available for doing it. Some criteria examples are:

$1 == 7$	first box property is 7
$2 > 5$	second box property is a more than 5 value
$box([0,10];[0,5];[-0.5,1])$	all boxes in the $[0,10] \times [0,5] \times [-0.5,1]$ space.

Combinations of criteria are also available by using a ";" symbol separating the different criteria. The "all" criteria allows to select all boxes in the input file. Examples:

[1] It is not possible to merge 2D and 3D boxes. Each box must have at least one property, and all boxes must have the same number of properties.

$1 >= 4;\ 1! = 6$	first property greater or equal to 4 and not equal to 6
$box([0,8];[0,4]);\ 2 > 5$	all boxes in $[0,8] \times [0,4]$ which 2^{nd} property greater than 5.
all	Select all actors (boxes from input file).

It is possible to show actors one by one. The *step* option allows to give a time delay between two consecutive actors.

3.2 The Color Menu

The color menu allows to change the actors' color/transparency in scene. It is necessary to give a criterion (see section 3.1) and a color in RGB format. A transparency rate is also available with the *trans* option. For example:

Criteria: $2 == 4$ **color:** $0.5, 0, 0$ **Trans:** 0.5

Change the color for all boxes with their second property equal to 4 to dark red and 50% of transparency. This menu has an additional criterion: *background* (or *bg*), which represents the back color of the scene.

3.3 The Photo Menu

The photo menu allows to take snapshots from scene. To do this, a file name and format must be specified. If two snapshots have the same name, an *images sequence* option is suggested. A sequence is a set of images with the same name and a sequence number. For example: image001.bmp, image002.bmp, image003.bmp...

The name of each sequence is saved in memory allowing to make a video (see section 3.4 for more information).

3.4 The Video Menu

The video menu allows to make animations from images sequences. There are two format available: GIF and MPG. The images sequence must be selected from the list. Two options are available to customize the animation: *time delay* and $factor$. Time delay is the time between two consecutive images. Factor resizes the animation frame size with respect to initial image size. For example:

delay: 0.5 **Factor:** 0.75

Create an animation with 0.5s between images and 75% of the initial size.

3.5 The Size Menu

The size menu allows to change the visualization window size. The values are measured in pixels. The minimum width is 550. For example:

Image size: 800×600

Fixes the visualization window size to 800×600 pixels.

3.6 The Exec Menu

The exec menu allows to run external commands and files in Python language from the application. This permits the application capabilities can be easily extended with a source code (like a *plugin*). Many specific tasks can be performed automatically, and its can include internal application objects and functions.

Some internal application objects and functions are:

- **self.ren** The VTK Renderer of scene (see VTK en [5]).
- **self.renWin** The VTK Renderer Window.
- **self.actors**[] List of actors representing boxes. Each list item is an actor[2].
- **self.sequences**[] List of images' sequences in memory. Each list item is a two values list. The first one is the name of sequence (the same that the image name), and the second one is the amount of images took at the moment.
- **self.setCamera()** Resets the camera to the original values (This is equivalent to press the button *Reset* in the application option menu).
- **self.showAxis()** Shows/Hides axis from scene.
- **self.makeCriterion(text)** Sets the set of criteria from the *text* argument. "Text" is a chain of characters that describes a list of criteria separated by a ";" symbol (this is equivalent to *criteria* option in the *Actor* and *Color* menu[3]. This function returns 1 if the text was understood, other returns 0.
- **self.selectObjects()** Selects all boxes supporting the criterion (made in $self.makeCriterion(text)$). This selection is saved in the $self.searchResult[]$ list like integer values[4]. It returns 1 if selection gives at least an actor, and 0 in the other case.
- **self.writephoto(image,filename)** Writes an image on disk. The first argument is a vtkImageWriter[3] object, and the second one is a string with the image file name.

When a command or file is performed, and result message is included in the *result* list with the $-->OK!$ or $-->Error$(description) string.

[2]For example, $actors[0]$ is the first box in the input file, and $actors[1]$ is the second one.

[3]For example, a *text* argument can be "$1 == 7; box([0,10];[0,10][0,10]); 2 <= 4$", for setting all boxes with first property equal to 7 in the space $[0,10] \times [0,10] \times [0,10]$, with second property least than or equal to 4.

[4]For example, $self.searchResult[]$ can be a list with 1, 3, 5, indicating the actors[1], actors[3] and actors[5] support the criterions.

4 Examples

4.1 Input file

```
([2,4];[2,4];[2,4])          1 1
([2.5,3.5];[2.5,3.5];[1,2])  2 2
([4,5];[2.5,3.5];[2.5,3.5])  3 2
([2.5,3.5];[2.5,3.5];[4,5])  4 2
([1,2];[2.5,3.5];[2.5,3.5])  5 2
([2.5,3.5];[1,2];[2.5,3.5])  6 2
([2.5,3.5];[4,5];[2.5,3.5])  7 2
([0,6];[0,6];[0,6])          8 3
```

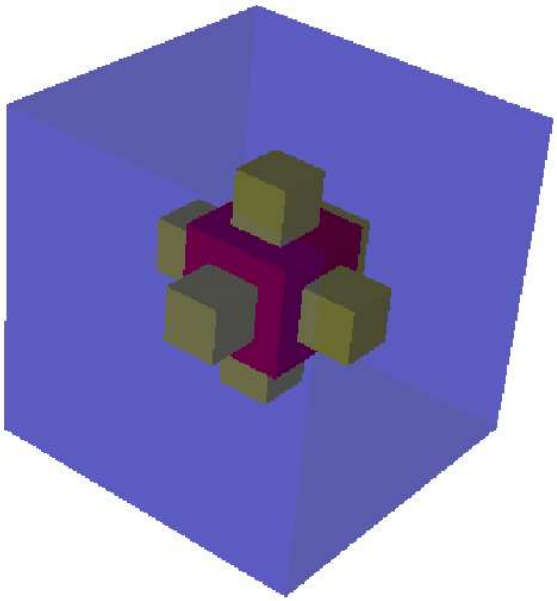

Figure 1: Visualization of MAPA input file

Figure 1 shows the visualization of MAPA input file after the following steps:

- Showing all actors (criterion: all) from *Actor* menu.
- Changing box color (0,0,1) and transparency (0.3) for the boxes with first property equal 8 (criterion: $1 == 8$) from *color* menu.
- Changing box color (1,1,0) for boxes with second property equal 2 (criterion: $2 == 2$) and after, changing box color (1,0,0) for all boxes with second property equal 1 (criterion: $2 == 1$).

4.2 Plugin (extension)

The follow python source code makes a sequence of images to rotate in the “x”, “y” or “z” axis. In the first part (The parameters), program looks for global values. It is possible to give arguments to the program in this way.

```
# Parameters ------------------------------------
try:
    AXIS  = self.AXIS         # Axis for movement
except:
    AXIS  = "y"
try:
    DEGREE= self.DEGREE       # degree by step
except:
    DEGREE= 4
try:
    TIMES = self.TIMES        # number of steps
except:
    TIMES = 91
try:
    FILENAME = self.FILENAME  # name for the sequence
except:
    FILENAME = "plugin-seq"
#-------------------------------------------------
os.mkdir('_'+FILENAME+'_')    # Making the directory
counter = 0                   # Setting the counter

# Getting the original camera information
cam = self.ren.GetActiveCamera()
Old = [cam.GetClippingRange(), cam.GetFocalPoint(), cam.GetPosition(), cam.GetViewUp()]

# Moving and taking photographs
for i in range(TIMES):
    image=vtkJPEGWriter()
    photoname =  '_'+FILENAME+"_/"+FILENAME+zfill(str(counter),3)+'.jpg'
    counter = counter + 1
    self.writephoto(image,photoname)
    if AXIS == "x":
        self.ren.GetActiveCamera().Elevation(DEGREE)
    elif AXIS == "y":
        self.ren.GetActiveCamera().Azimuth(DEGREE)
    elif AXIS == "z":
        self.ren.GetActiveCamera().Roll(DEGREE)
    self.renWin.Render()

# Putting the sequence in the list of sequences
self.sequences.append([FILENAME, counter])
```

```
# Setting the original values for the camera
self.ren.GetActiveCamera().SetClippingRange(Old[0])
self.ren.GetActiveCamera().SetFocalPoint(Old[1])
self.ren.GetActiveCamera().SetPosition(Old[2])
self.ren.GetActiveCamera().SetViewUp(Old[3])
```

5 Conclusion

This document describes MAPA, an application for displaying box objects in 2D or 3D. This program allows to interpret result values from many techniques of domain filtering, when the result values are boxes parallel to axis.

On the other hand, this application allows to take snapshots and make animations for including it in documents like papers or presentations. In this way, explanations of domain filtering techniques are more easy.

Running external code allows to extend the application capabilities to more complex objects (like sphere or special form). The interaction between this complex objects and the basic application objects and functions are also available. Very specific forms can be easily displayed and saved like images or movies.

Additionally, many complex and common tasks can be performed in an automatic way.

References

[1] http://tmml.sourceforge.net/doc/tk/. Tk reference manual, 2004.

[2] http://www.python.org/doc/. Python documentations index, 2004.

[3] http://www.vtk.org/doc/. Vtk documentation, 2004.

[4] Fredrik Lundh. An introduction to tkinter, 1999.

[5] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit, 3rd Edition*. Kitware, 2002.

[6] John W. Shipman. Tkinter reference: a gui for python, 2004.

[7] Guido van Rossum and Fred L. Drake. Python tutorial (in many languages), release 2.0, 2000.