MA Petite Application

Carlos Grandón <cobra@inf.utfsm.cl>

8 de julio de 2004

1. Introducción

MAPA es un programa para desplegar cajas en 2D o 3D. El objetivo principal es facilitar la visualización de los resultados producidos por alguna técnica de filtraje sobre un espacio solución. Sus principales funciones son:

- Visualización de cajas segun criterio dado
- Cambiar color/transparencias segun criterio
- Tomar fotografías de los objetos desplegados
- Relizar video de animaciones con secuencias de fotos
- Cambiar el tamaño de la ventana de visualización
- Ejecutar comandos o archivos externos en Python
- Mostrar/esconder los ejes
- Recuperar la posicion original de la cámara

2. Filosofía y ejecución

MAPA esta escrito en **Python** [7, 2] y utiliza las bibliotecas de **VTK** [5, 3] para desplegar los objetos en la pantalla. La interfaz con el usuario esta basada en **Tkinter** [4, 6], una biblioteca de **TK** [1] para Python. Básicamente se trata de un script ejecutable que recibe un archivo como argumento. El archivo de entrada posee la descripcion de las cajas que se desea desplegar en el siguiente formato:

$$
\begin{array}{ccccc}
< caja_1 > & < prop_1 > & < prop_2 > & ... & < prop_n > \\
< caja_2 > & < prop_1 > & < prop_2 > & ... & < prop_n > \\
\vdots & \vdots & \vdots & & \vdots \\
< caja_m > & < prop_1 > & < prop_2 > & ... & < prop_n >
\end{array}
$$

Cuadro 1 Ejemplo de un archivo de entrada

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

donde $< caja* >$ representa un objeto en el espacio 2D o 3D, y $< prop* >$ es un entero que simboliza una propiedad (caracteriza a la caja[1]).

El cuadro 1 muestra un ejemplo de archivo de entrada compuesto por 3 cajas en el espacio. Cada caja posee dos propiedades.

Las propiedades se enumeran por orden de aparición, por ejemplo, la primera propiedad de la primera caja tiene valor 1. La segunda propiedad de la tercera caja tiene valor 2.

Para cada caja se define un *actor*, que será el encargado de representarla en la escena. Las propiedades permiten referirse a uno o mas de estos actores en los denominados *criterios* (ver sección 3.1)

3. Opciones del menú

La aplicación posee nueve botones con opciones. Los primeros 6 activan o desactivan sub-menus, mientras que los tres últimos ejecutam tareas específicas (mostrar/esconder los ejes, resetear la cámara y salir del programa).

3.1. Menú Actor

Permite mostrar/esconder actores de la escena. La selección se realiza a través de *criterios*. Para ello, se utilizan las propiedades definidas en el archivo de entrada. Ejemplos de criterios son:

$1 == 7$	primera propiedad de la caja posee el valor 7
$2 > 5$	segunda propiedad posee un valor mayor a 5
$box([0,10];[0,5];[-0.5,1])$	todas las cajas contenidas en box.

Esta opción también acepta combinación de criterios (separados por ";") y despliegue de todos los actores, ejemplos:

$1 >= 4;\ 1! = 6$	primera propiedad mayor o igual a 4 y distinta de 6
$box([0,8];[0,4]);\ 2 > 5$	todas las cajas mayor que 5 en el plano $[0,8]x[0,4]$.
all	Despliegue de todos los actores (cajas).

Adicionalmente es posible desplegar uno a uno los actores, con un retardo determinado. Para ello, la opción *step* permite ingresar el tiempo entre el

[1] No se pueden mezclar cajas en distintos espacios (2D ó 3D). Al menos debe existir una propiedad por caja. Además, es recomendable que todas las cajas tengan la misma cantidad de propiedades

despliegue de dos actores consecutivos.

3.2. Menú Color

Permite cambiar el color/transparencia de uno o mas actores en la escena. Para ello se especifica el criterio (ver 3.1) y el color en formato RGB normalizado. Adicionalmente es posible asignar un porcentaje de transparencia al color (también normalizado). Ejemplo:

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

(Cambia el color de todas las cajas cuya segunda propiedad sea igual a 4 al color rojo oscuro, con una transparencia de 50 %). En el caso del menú color, se reconoce un criterio adicional: *background* (o fondo de pantalla). También existe la especificación en versión reducida (bg).

3.3. Menú Photo

Permite generar imágenes de la escena. Para ello se especifíca el nombre con el cual se guardará la imágen y el formato. Cuando dos imágenes se almacenan con el mismo nombre, el programa consulta al usuario si desea generar una *secuencia*. Las secuencias son conjuntos de imágenes con igual nombre más un número. Por ejemplo: image001.bmp, image002.bmp, image003.bmp...

Cada secuencia tomada es almacenada en memoria, permitiendo posteriormente la generación de un video (ver 3.4).

3.4. Menú Video

Permite generar videos en formato GIF o MPG a partir de secuencia de imágenes. Para ello se debe seleccionar una secuencia de la lista, elejir un formato y presionar OK. Adicionalmente es posible entregar la cantidad de segundos de retardo entre dos imágenes consecutivas, y el factor de escala del video respecto al tamaño de las imágenes. Ejemplo:

delay: 0.5 **Factor:** 0.75

(Video con medio segundo entre imágenes y al 75 % del tamaño original)

3.5. Menú Size

Permite cambiar el tamaño de la ventana de visualización de la escena. Los valores se expresan en pixeles. El ancho mínimo es de 550 pixel. Ejemplo:

Image size: 800×600

(Ventana de visualización de 800 por 600 pixeles)

3.6. Menú Exec

Permite ejecutar comandos y archivos escritos en lenguaje Python directamente desde la aplicación. Este menú además permite extender las funcionalidades del programa, creando pequeños trozos de código al estilo *plugin* para realizar tareas específicas. Además, es posible utilizar las funciones y objetos internos del programa como parte del código.

Algunos objetos y funciones internas son:

- **self.ren** Corresponde al Renderer de la imágen (ver VTK en [5]).
- **self.renWin** Corresponde al Renderer Window.
- **self.actors**[] Lista de actores que representan las cajas leídas desde el archivo de entrada. Cada item de la lista corresponde a un actor[2].
- **self.sequences**[] Lista de las secuencias de imágenes existentes. Cada item de la lista esta compuesto por dos valores, el primero corresponde al nombre dado al archivo y el segundo a la cantidad de imágenes que componen la secuencia.
- **self.setCamera()** Resetea la cámara a su vista original (equivalente a presionar el botón *Reset* de la aplicación.
- **self.showAxis()** Muestra/oculta los ejes de coordenadas.
- **self.makeCriterion(text)** Establece el conjunto de criterios de selección de cajas dado un texto particular. La cadena de caracteres *text* posee las mismas caracteristicas de la entrada *criterion* en los menú *Actor* o *Color*[3]. Retorna 1 si el texto fue entendido y 0 en caso de error.
- **self.selectObjects()** Selecciona todas las cajas que corresponden con los criterios establecidos en $self.makeCriterion(text)$. Los valores se almacenan en la lista $self.searchResult[]$, como enteros que representan las cajas seleccionadas[4]. Retorna 1 si la selección dió como resultado, al menos un actor, y 0 en caso contrario.
- **self.writephoto(image,filename)** Escribe una imagen en el disco. El primer argumento corresponde a un objeto del tipo vtkImageWriter[3], y el segundo es una cadena de caracteres con el nombre de la imágen a escribir.

El resultado de cualquier comando ingresado (o archivo ejecutado) se presenta en *result*, con el valor $-->OK!$ en caso de ejecución exitosa o $-->Error$(descripción) en caso de error en la ejecución.

[2]Por ejemplo, $actors[0]$ corresponde a la primera caja del archivo, $actors[1]$ a la segunda, y así sucesivamente.

[3]Por ejemplo, *text* podría contener un valor "$1 == 7; box([0,10];[0,10][0,10]); 2 <= 4$", para indicar las cajas cuya primera propiedad es igual a 7, que se encuentran en el espacio $[0,10] \times [0,10] \times [0,10]$ y que su segunda propiedad es menor o igual a 4.

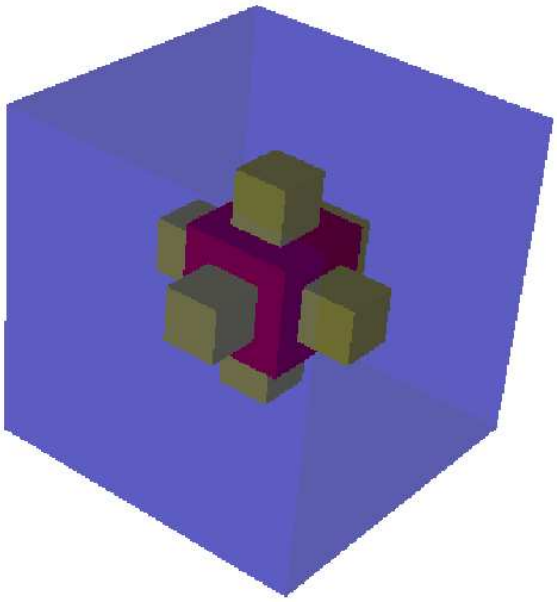
[4]Por ejemplo, $self.searchResult[]$ podría contener los valores 1, 3, 5, indicando que actors[1], actors[3] y actors[5] cumplen los criterios estabecidos.

4. Ejemplos

4.1. Archivo en entrada

```
([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
```

Figura 1 Visualización del archivo de entrada en MAPA



La figura 1 muestra el resultado de la ejecución de MAPA para el archivo de entrada de ejemplo. Los pasos realizados para su visualización son:

- Mostrar todos los actores (criterio: all) en el menu *Actor*
- Desde el menú *Color* asignar color azul (0,0,1) con trans de 30 % (0.3) a la caja con primera propiedad igual a 8 (criterio: 1 == 8).
- Asignar el color amarillo (1,1,0) a las cajas cuya segunda propiedad sea igual a 2 (criterio: 2 == 2) y, posteriormente, asignar el color rojo (1,0,0) a la caja cuya segunda propiedad sea igual a 1 (criterio: 2 == 1).

4.2. Plugin (extensión)

El siguiente trozo de código, ejecutado desde MAPA, costruye una secuencia de imágenes en torno a uno de los ejes. La primera parte examina si dentro del programa existen valores para los parámetros (eje, grados de rotación, número de veces que debe rotar y nombre del archivo de secuencia). De esta forma es posible entregarle valores, ejecutado comandos en el menú Exec.

```
# 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. Conclusión

En este documento se presentó una aplicación para visualizar cajas en 2D y 3D. Esto permite facilitar la interpretación de resultados obtenidos con técnicas de filtraje que generen espacios formados por figuras cuyos planos son paralelos a los ejes.

Por otro lado, la posibilidad de crear imágenes y videos, facilita la explicación de estas técnicas, al permitir la inclusión de ejemplos en documentos. La posibilidad de ejecutar código externos permite extender las potencialidades de MAPA, generalizando su uso a entornos más complejos. Además, las funciones y objetos internos (descritos en la sección 3.6) permiten automatizar tareas comunes, como desplegar conjuntos de actores, generar imágenes y manipular cámaras de la escena.

Por último, el acceso directo al *renderer* de la escena proporciona un camino para agregar diferentes objetos, de caracteristicas mucho más complejas, permitiendo su interacción con los objetos predeterminados.

Referencias

[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.