SAMBA Animation Designer's Package

John T. Stasko

Georgia Institute of Technology
stasko@cc.gatech.edu

This document describes the Samba program which provides an interpreted, interactive animation front-end to POLKA. Samba simply reads an ascii file, one command per line, in order to acquire its directions for creating an animation. This is beneficial because you can have the output of any program, be it Pascal, C, Modula-2, etc., drive an animation. I have used this tool in an undergraduate algorithms class. In addition to implementing an algorithm, students can develop an animation of it just by judiciously placing print statements into their program.

Animations developed with this system will be carried out in windows with a real-valued coordinate system that originally runs from 0.0 to 1.0 from left-to-right and from 0.0 to 1.0 from bottom-to-top. Note, however, that the coordinate system is infinite in all directions. You will create and place graphical objects within the coordinate system, and then move them, change their color, visibility, fill, etc., in order to depict the operations and actions of a computer algorithm.

The format for the individual commands is described later. You can run this program interactively by piping the output of your program to it, e.g.,

```
% yourprog | samba
```

If the textual output of your program was saved to a file (perhaps even on another computer system), you can simply have Samba read that file as input via a command line argument, e.g.,

```
% samba outfile
```

By using this method, you will be able to run your animation repeatedly without exiting Samba.

In order to build an algorithm animation, you need to augment your implementation of the algorithm under study by a set of output (e.g., printf, writeln, cout) statements. The statements should be placed in the program at the appropriate positions to provide a trace or depiction of what your program is doing.

Below we summarize the different commands that exist within the system. If there's a command you would like added, just ask. We are always looking to improve the system. To begin this section, we describe the commands in general.

Each command begins with a unique one word string. Make sure that you spell the strings correctly. Each graphical object that you create should be designated by a unique string id. You will need to use that id in subsequent commands that move, color, alter, etc., the object. In essence, the id is a handle onto the object. Most of the commands and their parameters should be self-explanatory. Arguments named `steps` and `centered` are of integer type. Arguments named `xpos, ypos, xsize, ysize, radius, lx, by, rx, ty` are real or floating point numbers. Make sure that they include a decimal point (typing 0 instead of 0.0 will cause an error). The argument `fillval` should be one of the following strings: `outline`, `half` or `solid`. The argument `widthval` should be one of the following strings: `thin`, `medthick`, or `thick`. The parameter `colorval` can be any color string name from the file `/usr/lib/X11/rgb.txt` (one word, no blanks allowed).

Note that any line with a per cent character (%) in column 1 is interpreted as an instructional comment—no command is carried out.

If you have used the Animator for the XTango system, you will find that the Samba system is very similar. In fact, Samba has been designed to be backward compatible with XTango Animator trace files. Samba simply adds new features to the base XTango Animator. What are these features? For one, all objects can be referred to by arbitrary string identifiers, instead of only integers. Samba also provides the option of multiple Views. In other words, users may have several animation windows open and running at one time, all with their own graphical objects and events. Samba also provides for explicit concurrency of animation events. Blocks of commands can be defined so that they all begin at the same point in time. The events need not end at the same time. Also, some additional objects have been added, which are detailed below.

Samba will print warning messages if you try to do erroneous actions like reuse an ID, try to delete or move an invalid ID, and so on. You can turn these warnings off by using the -O command line flag.

`comment A trailing string`

This command simply prints out any text following the “comment” identifier to the shell in which the animator was invoked.

`viewdef id <xsize> <ysize>`

This command defines a single view. It must be used to create multiple new views in an animation. The xsize and ysize parameters are optional specifications for the size of the window in pixels. View definitions must precede all other commands in a trace file (except for %). If no view definitions are found, then Samba will create default Samba view and perform all subsequent animation actions in it.

`view id`

This command sets the current view in which objects will be created or manipulated. That is, all trailing animation actions occur in this view until another `view` command is found. Note that objects are bound to the the view that was active at the time of their creation. Trying to manipulate objects from a particular view while another view is active will result in an error. The bottom line is: Change the view with this command before creating an object to put in that view, and change to that view to manipulate the object.

{

Begin a block of concurrent commands. All commands in the block will start at the same time index. Nesting of blocks are not allowed, but referencing different views within one block is allowed.

}

End a block of concurrent commands. Every { must have a matching }, and nesting is not allowed.

`bg colorval`

Change the background to the given color. The default starter is white.

`coords lx by rx ty`

Change the displayed coordinates (real valued) to the given values. You can use repeated applications of this command to achieve interesting panning or zooming effects in an animation view.

`delay steps`

Generate the given number of animation frames with no changes in them.

`line id xpos ypos xsize ysize colorval widthval`
Create a line with one endpoint at the given position and of the given size.

`pointline id xpos1 ypos1 xpos2 ypos2 colorval widthval`
This creates a line like the `line` command except that you provide the two endpoints of the line here rather than the line's size.

`rectangle id xpos ypos xsize ysize colorval fillval`
Create a rectangle with lower left corner at the given position and of the given size (size must be positive).

`circle id xpos ypos radius colorval fillval`
Create a circle centered at the given position.

`triangle id v1x v1y v2x v2y v3x v3y colorval fillval`
Create a triangle whose three vertices are located at the given three coordinates. Note that triangles are moved (for move, jump, and exchange commands) relative to the center of their bounding box.

`polygon id numsides colorval fillval xpos ypos vx1 vy1 ... vx7 vy7`
Create a polygon starting at (xpos, ypos) with up to seven other vertices specified by (vx1, vy1) ... (vx7, vy7). Specify any number of sides between three and eight with `numsides`.

`text id xpos ypos centered colorval string`
Create text with lower left corner at the given position if `centered` is 0. If `centered` is 1, the position arguments denote the place where the center of the text is put. The text string is allowed to have blank spaces included in it but you should make sure it includes at least one non-blank character.

`bigtext id xpos ypos centered colorval string`
This works just like the text command except that this text is in a much larger font.

`flextext id xpos ypos centered colorval fontname string`
This is the flexible text command, and it works just like the text command except that you can explicitly specify the font (string name) that you want to use. Any valid X11 font is allowable.

`set id num id1 id2 ...`
Create a set of already existing objects that can be manipulated as a group through a new ID. Specify the number of items in the set with the num parameter, and list the identifiers of the member objects in a space-delimited list. All manipulations of the set object affect the members of the set as well. All objects in the set must be in the same view. The set acts like a mathematical set, so duplicate identifiers in id1 on will be ignored.

`move id xpos ypos`
Smoothly move, via a sequence of intermediate steps, the object with the given id to the

specified position.

`moverelative id xdelta ydelta`
Smoothly move, via a sequence of intermediate steps, the object with the given id by the given relative distance.

`moveto id id`
Smoothly move, via a sequence of intermediate steps, the object with the first id to the current position of the object with the second id.

`jump id xpos ypos`
Move the object with the given id to the designated position in a one frame jump.

`jumprelative id xdelta ydelta`
Move the object with the given id by the provided relative distance in one jump.

`jumpto id id`
Move the object with the given id to the current position of the object with the second id in a one frame jump.

`color id colorval`
Change the color of the object with the given id to the specified color value.

`alter id newstring`
Change the string presented for the given text object to `newstring`.

`delete id`
Permanently remove the object with the given id from the display, and remove any association of this id number with the object.

`fill id fillval`
Change the object with the given id to the designated fill value. This has no effect on lines and text.

`width id widthval`
Change the width value of the line with the given id to the designated fill value. This will not work with any other graphic object.

`vis id`
Toggle the visibility of the object with the given id.

`lower id`
Push the object with the given id backward to the viewing plane farthest from the viewer.

`raise id`
Pop the object with the given id forward to the viewing plane closest to the viewer.

`exchangepos id id`
Make the two objects specified by the given ids smoothly exchange positions.

`switchpos id id`
Make the two objects specified by the given ids exchange positions in one instantaneous jump.

`swapid id id`
Exchange the ids used to designate the two given objects.

The next three pages illustrate three sample input files to Samba. The first shows basic features (from the old xtango animator), the second illustrates some new Samba commands (basic ones) and the third illustrates multiple views and explicit concurrency.

Acknowledgments

Irwin Coleman did a great job in helping to implement Samba. Without his assistance, the package would not be available.

```
% This example illustrates the more basic commands
% Note that it uses only integers as IDs, but in general,
% arbitrary character strings can be used
comment This is a sample animation script
circle 1 0.8 0.8 0.1 red half
line 2 0.1 0.1 0.2 0.2 green thin
rectangle 3 0.1 0.9 0.1 0.1 blue solid
text 4 0.0 0.0 0 black Hello
text 5 0.5 0.5 1 black RealLongStringandThenSomeAndEvenMore
circle 6 0.3 0.3 0.2 wheat solid
triangle 7 0.5 1.0 0.6 0.8 0.4 0.9 cyan solid
bigtext 8 0.2 0.2 0 black Some Big Text
moveto 1 6
moverelative 3 0.05 -0.4
jumprelative 4 0.4 0.4
raise 1
lower 1
color 6 blue
move 3 0.5 0.5
jump 3 0.9 0.9
jumpto 3 6
raise 3
fill 3 half
fill 3 outline
fill 3 half
vis 3
vis 3
vis 6
delete 8
line 200 0.9 0.2 0.0 0.6 black thick
bigtext 8  0.6 0.2 0   DeepPink    More Big Text
flextext 88  0.4 0.3 0   magenta  8x13bold    Flex Text 8x13bold
rectangle 12 0.7 0.7 0.1 0.1 green solid
exchangepos 12 3
exchangepos 12 3
exchangepos 12 3
exchangepos 12 3
switchpos 12 3
circle 99 0.8 0.8 0.15 black outline
exchangepos 1 99
bg pink
bg LemonChiffon2
coords -0.5 -0.5 1.5 1.5
```

```
% This example illustrates features not in xtango's animator

% Add a polygon
polygon poly1 4 purple solid 0.1 0.0 0.5 0.5 0.7 0.5 0.5 0.1 0.0 0.0

% Add a pointline
pointline liner 0.2 0.8 0.7 0.8 blue thin

% Add some text
text 12 0.6 0.6 1 black A text string

% Move the line down
move liner 0.08 0.3

% Change the polygon's color to cyan
color poly1 cyan

% Make the line thicker
width liner thick

% Change the text string
alter 12 Has just changed

% Make a set out of the polygon and line
set newset 2 poly1 liner

% Move them together
moverelative newset 0.3 0.3
```

```
% A simple animation for advanced feature demonstration.
% Initialize the views
viewdef MainView 600 600
viewdef SecondView 400 400
% Start animation
comment Switch to MainView.
view MainView
comment Draw a line there.
line thinblackline 0.2 0.2 0.5 0.0 black thin
comment Switch to the SecondView.
view SecondView
comment Draw a rectangle here.
rectangle redrectangle 0.5 0.5 0.3 0.2 red half
view MainView
comment Let's move the rectangle and the line at the same time.
{
view SecondView
move redrectangle 0.3 0.3
view MainView
move thinblackline 0.5 0.5
}
delay 15
comment Now, let's move them separately.
moverelative thinblackline -0.2 -0.2
view SecondView
moverelative redrectangle 0.2 0.2
comment Let's put in a polygon in SecondView.
polygon greenpoly 4 green solid 0.0 0.0 0.5 0.5 0.7 0.5 0.5 0.1 0.0 0.0
comment Let's alter the fill for the rectangle.
fill redrectangle solid
comment Let's get rid of that polygon.
delete greenpoly
view MainView
comment Let's make that thin line thick.
width thinblackline thick
view SecondView
text boxlabel 0.6 0.6 1 black Rectangle
comment Let's make the rectangle and its label a set.
set boxset 2 redrectangle boxlabel
comment Now we can move the set around like a single object.
moverelative boxset 0.5 0.5
```