Graphical Simulator for Multi-robot Systems

Chitresh Bhushan and Sayandeep Purkayasth

April 1, 2007

1 Objective

Our objective was to code a OpenGL application that could read the positions and velocities of a given number of robots and render them onto a visual display unit.

2 Implementation

2.1 Libraries used

We have used the following libraries for graphics rendering and image manipulation.

- *OpenGL* - The Open Graphics Library
  This Library has been used for rendering the various scenes onto the display unit.
- *GLU* - The OpenGL Utility Library
  It has some routines that provide higher-level drawing routines from the more primitive routines that OpenGL provides.
- *GLUT* - The OpenGL Utility Toolkit
  It is a library of utilities which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, monitoring of keyboard and mouse input, and drawing some geometric primitives.
- *OpenCV* - The Open Computer Vision library
  This Library impements image manipulation routines and has been used to save snapshots of the simulation at runtime and also capture video segments of the simulation.

3 Overview

We have created a scene containing a grid plane, on which the ball and a number of bots are placed. On starting animation, data is read from the file (line by line, one at a time for each frame) in realtime and next directly rendered on to the screen output. A snapshot (JPG format) or a video (output file format is `avi`) may be saved. The simulation may be restarted, paused, speeded up/down as

required. Also the number of enabled lights may be increased from none (0) to four (4). The position and other light properties may also be set. The number of available lights may also be increased. The OpenGL camera may be relocated on top of any of the robots.

4 Compiling and building from source

We have built the necessary binaries using GCC 4.1.2 and Microsoft Visual C++ 6.0 compilers. However it is recommended that the same are rebuilt on the target computer.

4.1 Windows

The Microsoft Visual Studio 6.0 workspace we had created has been provided. OpenCV and OpenGL libraries must be installed before compilation. Please refer to their individual manuals for installation instructions.

- OpenCV
  Stable version 1.0 has been used. Source Code is available from its project page on SourceForge®.
  Please read `openCV instructions.pdf` for details on setting up a workspace with OpenCV libraries.

- OpenGL, GLU, GLUT
  These have been included along with source code.

4.2 Linux

OpenCV and OpenGL libraries must be installed before compilation. Details follow.

- OpenCV
  These are available from this SourceForge® link. These following packages must be installed.
  `libcv0.9.7-0, libhighgui0.9.7-0, libhighgui-dev, libcv-dev`
  Further development might require `libcvaux0.9.7-0` and `libcvaux-dev`

- OpenGL, GLU, GLUT
  These following packages must be installed.
  `glutg3, glutg3-dev`
  Hardware dependent drivers are also required. We used `libglu1-mesa, libglu1-mesa-dev` for MESA video hardware driver.

A makefile has been provided. Commands are as follows.

```
$ make Robot_Animation
```

5 Usage

A menu has been made available on right click within the animation window. This contains all runtime options. The keyboard equivalents are listed in *Table 1*.

Shortcut	Action
Animation control	
<escape>	Stop and Exit the simulation at any time
<space>	Toggle pause/play of simulation
r	Replay animation from start
View control	
0	Return view to isometric view
t	Change view to Top view
n	Toggle numbering of the displayed robots
s	Take a snapshot of the simulation
v	Toggle video recording of the simulation
<number>	Shift OpenGL camera on to top of robot numbered number
+	Zoom in
-	Zoom out
Accessories	
i	Toggle display of Information box

Table 1: Runtime keyboard shortcuts. Note: Uppercase forms of the specified characters will also work.

Right-click menu All the keyboard options have been included in the right-click menu. Over and above these, the following animation options are also available.

1. Light control
   Options for increasing number of enabled lights up to a maximum of 4 lights have been provided.

2. Animation speed control
   Options for increasing speed of animation from 1X up to a manximum of 25X have been provided.

5.1 Major Features

5.1.1 Save snapshot

This function reads pixels from the frame buffer and processes them into a JPG image using built-in OpenCV library functions. Images are saved in the working directory with the following file name format.

`capture_<time_int>_<theta>_<phi>.jpg`

Here `theta`, `phi` refer to θ and ϕ of the OpenGL camera position with respect to the spherical coordinate system.[1] `time_int` refers to the time (in integer casted form) when the snapshot is taken.

5.1.2 Save video

This function uses built-in OpenCV library functions to write frames to an AVI file. The codec to be used can be selected in Windows (A pop-up window shows the available codecs for video compression. One may be selected.) whereas it is restricted to `DIVX` codec in linux environment. Default *frames per second* (fps) is 25 fps. Videos are saved in the working directory with the following file name format.

`capture_<start_time_int>_<theta>_<phi>.avi`

Here `start_time_int` refers to the time (in integer casted form) from which video capture starts. `theta`, `phi` again refer to θ and ϕ are the respective values of the start frame.

Please note that enabling video recording reduces rendering speed and performance, in general due some inherent delay in the used OpenGL function to capture the screen.

5.1.3 Info Box

This is a sub window (within the animation window) that shows various runtime information. It shows the last command passed, the zoom level, θ, ϕ, etc.

Please note that enabling enabling Info box option also reduces rendering speed and performance, in general due some inherent delay in rapidly rendering changing text.

6 Technical details and Code editing

6.1 Working

The present code works in the following sequence.

1. The animation coordinate limits are found by reading in the whole `output_pos.log` file once. The animation floor is built using these X, Y limits. Then the file read pointer is repositioned to the file beginning.

2. Now the `output_pos.log`[2] file is read in line by line. For each line each frame is rendered, after taking into consideration the time delay between the previous and the present frame times.

[1] $x = \rho \sin\phi \cos\theta$, $y = \rho \sin\phi \cos\theta$, $z = \rho \cos\phi$. $\rho = 250$ units has been set constant for the simulation except while zooming in/out.

[2] The file to be read in can be changed directly from the code. Refer `Definitions.h` for declarations.

Our code is organised as follows. All function and variable declarations and include files are included in `Definitions.h`. `Robot_class.h` contains file input routines, and some variable are set here. `Robot_Animation.cpp` contains the OpenGL display and OpenCV screen capture routines.

Function	What it does
`Robot_Class.h`	
`display_data`	displays the data
`get_data`	read the data from input
`next_no`	finds the next number, reading from the string and updates the string pointer position
`Robot_Animation.h`	
`position_robot`	Displays the robots at their respective positions
`init`	Initializes OpenGL display
`display`	Displays the environment and calls
`play_control`	Controls the start, speed and end of animation
`mouse`	Mouse event Handler
`reshape`	Handler for window resize
`key_event_handler`	Handler for keyboard events
`menuSelect`	Menu Funtions
`motion`	Handles the Motion of mouse
`outputCharacter`	Draws the string at any point
`reshape`	Determines the Eye location
`kill_animation`	Stops the animation
`WindowDump`	Takes the screenshot
`video_dump`	Records a video
`drawString`	Draws the string in sub-window (2D text)
`subReshape`	reshape for sub-window(info-box)
`draw_floor`	Draws the floor on which robots are moving
`round_to_int`	Rounds off a float to an integer
`sleepMilli`	sleep function for unix using nanosleep()

Table 2: The functions used and what they do