Using T3: TableTop Toolkit

Philip Tuddenham
<Firstname>.<Lastname>@cl.cam.ac.uk
For T3 version 1.0.4
16 October 2008

Licence

Introduction and Getting Started

T3 (TableTop Toolkit) is a research-quality Java toolkit that enables CSCW researchers to rapidly prototype high-resolution tabletop applications for remote and co-located collaborators. T3 was developed by Philip Tuddenham, under the supervision of Peter Robinson at the University of Cambridge Computer Lab.

This guide outlines the licence, requirements, and architecture of T3, along with tips for writing programs.

The T3 documentation is rather sketchy. The best way to get started is to try the example programs (see below) and look at their code. You'll need a Windows XP computer (haven't tested on Vista, it may work), a graphics card supporting OpenGL, and an installation of Java 1.6. Run run-example-1.bat. It should output a couple of lines of text to a console and remain running. Now run run-client-sample.bat. It should show a full-screen window containing a rectangle that you can drag around and click on with the mouse buttons. Press F12 to exit the client, and then click on the cross of the console window to close the run-example-1.bat. Try the other examples in the same way.

The T3 API is largely documented using javadoc, but to work in T3 you'll also need an understanding of its architecture. I suggest reading Chapter 4 of my PhD dissertation, shortly available as a technical report from http://www.cl.cam.ac.uk/t3, or alternatively:

> Philip Tuddenham and Peter Robinson. T3: Rapid Prototyping of High-Resolution and Mixed-Presence Tabletop Applications. *Proc. 2nd IEEE*

International Workshop on Horizontal Interactive Human-Computer Systems (TABLETOP'07), Newport, RI, October 2007.

Requirements

T3 runs a multi-projector display using a single desktop PC and multiple graphics cards (though a single projector display using a single graphics card is also fine). T3 requires Windows XP (perhaps Vista also works) and Java 1.6 or later.

The graphics cards must be capable of using OpenGL hardware acceleration on each graphics card head (monitor output) simultaneously. In our system at Cambridge we use 3 dual-head cards with Nvidia chipsets, and Nvidia drivers.

In order to calibrate a multi-projector display automatically, T3 requires a high precision pointing input device whose coordinate system is defined absolutely in terms of the display surface itself. A large graphics tablet is suitable but a conventional mouse is not. T3 currently supports wintab-compatible graphics tablets, and Maxell Anoto streaming styluses, and it is possible to extend the system to use other input devices such as multi-touch surfaces. At a push, you could calibrate by measuring using a large sheet of squared paper. There is not currently a facility to calibrate the system using cameras.

Architecture

T3 uses a client-server architecture inspired by Mark Ashdown's Escritoire work. The client handles rendering, user input and multi-projector issues such as warping. The server provides a programmer-friendly API and handles multiple clients. The client and the server are two separate processes and communicate via TCP. They are started by running separate scripts. If you exit the client but keep the server running then you can reconnect to the same session to resume as you left it. If you exit the client you won't be able to start another server until you close the old one.

This architecture may seem counter-intuitive for a system that processes a great deal of image data, but it aids T3's design goals. Ordinarily, the client and the server run on the same computer.

The Client

The client handles rendering, user input and multi-projector issues such as warping, using JOGL. The client packages are t3.hrd.* and t3.remotehrd.client.

Run the client using the run-client.bat script, specifying as an argument the name of a java properties file that contains the client configuration, for example:

run-client.bat myconfig.txt

The distribution contains a sample client configuration file, and the script run-client-sample.bat uses this sample file. It should run on any system that meets the minimum requirements.

The configuration file contains:

- The id of the client
  - clientId=0

- The name and TCP port of the server to connect to. Eg:
  - serverName=localhost
  - serverPort=2000

- Whether to share textures between the OpenGL windows. Sharing can lead to a performance increase on some systems:
  - shareTexturesBetweenContexts=false

- For each point input device, the name of the class that handles the device and any configuration details for that device. Begin at zero and work upwards. If no devices are specified then the system will use the mouse. Currently only a wintab class (for large graphics tablets) is available. To implement your own class, you need to subclass the PointInputDevice class. Eg:
  - hrd.pointInputDevices.0.className=hrd.input.Wintab
  - hrd.pointInputDevices.0.personId=0
  - hrd.pointInputDevices.0.xDivJump=10
  - hrd.pointInputDevices.0.yDivJump=10
  - hrd.pointInputDevices.0.xScale=0.025400
  - hrd.pointInputDevices.0.yScale=0.025400
  - hrd.pointInputDevices.0.buttonResolving=false

- The indexes and options for each graphics card head to be used by the system. The indexes tend to start at 0 and can be identified using the run-diag.bat program (below). The order of the indexes is irrelevant. The fullScreenExclusive option should be set to false. Eg:
  - hrd.projectorConfig.graphicsDeviceIndexes=0,1,2,3,4,5
  - hrd.projectorConfig.0.fullScreenExclusive=false
  - hrd.projectorConfig.1.fullScreenExclusive=false
  - hrd.projectorConfig.2.fullScreenExclusive=false
  - hrd.projectorConfig.3.fullScreenExclusive=false
  - hrd.projectorConfig.4.fullScreenExclusive=false
  - hrd.projectorConfig.5.fullScreenExclusive=false

- The corner points for each graphics card head's projector, as 4 (x,y) pairs. On a tabletop system, set each pair initially to 0 and then set it using the calibration utility (see below). If you are debugging and just using a conventional monitor then see the advice in the "Debugging" section below. Eg:
  - hrd.projectorConfig.0.DESKcorner0x=23.519432344516268

    - hrd.projectorConfig.0.DESKcorner0y=290.6627784474181
    - hrd.projectorConfig.0.DESKcorner1x=35.71870744075227
    - hrd.projectorConfig.0.DESKcorner1y=597.549768037677
    - hrd.projectorConfig.0.DESKcorner2x=431.1046906074772
    - hrd.projectorConfig.0.DESKcorner2y=608.9637469932343
    - hrd.projectorConfig.0.DESKcorner3x=458.5923427556402
    - hrd.projectorConfig.0.DESKcorner3y=299.1475395911261

- The rectangle representing the visible area of the workspace. Graphics are clipped so that only this rectangle is visible.
    - hrd.projectorConfig.DESKvisibleRectMask.0x=0.0
    - hrd.projectorConfig.DESKvisibleRectMask.0y=0.0
    - hrd.projectorConfig.DESKvisibleRectMask.1x=820.0
    - hrd.projectorConfig.DESKvisibleRectMask.1y=700.0

See the sample configuration file for more details.

The run-diag.bat program tests the OpenGL capabilities of each graphics card head on the system and outputs the corresponding index numbers. It does not take any arguments.

The run-calib.bat utility takes a configuration file with dummy calibration values and allows you to calibrate using point input device 0 defined in the file. (If you have not defined any point input devices in the file then you cannot use run-calib.bat). It will project calibration points and ask you to use the device to click/press/touch on each point. When you are done you can test the display and save the data back to the file. It also allows you to test point input devices. The utility takes the name of the file as its first argument.

All the batch file scripts look a bit like this:

```
java -ea -Dsun.java2d.noddraw=true
-Djava.util.logging.config.file=logging.properties
-classpath lib/Jama-1.0.2.jar;lib/jogl.jar;t3-build;. <Java class to run>
```

Java.exe must be on your operating system path. If you have multiple JVMs installed (and you may well do if your Windows XP installation came with a pre-installed Microsoft JVM) then this must be the correct java.exe. You can test it by opening a command prompt and typing “java -version”.

The other attributes are as follows: -ea enables assertions; the next statement is necessary for OpenGL to operate; the next statement instructs the Java logging system to use a configuration specified in “logging.properties”; and the classpath contains two jar files and the t3 class files. If the client frequently runs out of heap space on a particularly demanding application then you may need to edit the scripts to specify additional options to allow more heap space.

If you wish to run these programs through an IDE then you must configure the JVM arguments accordingly using the IDE’s interface.

The Server and Example Programs

Server Overview

Tabletop applications use the Portfolios API provided by the server, contained in the t3.portfolios package. This provides a brief overview; for more details, please see the example programs and the Portfolio and PortfolioServer javadoc comments.

A portfolio is essentially a user interface component on the display surface. You create a portfolio by instantiating the Portfolio class or one of its subclasses. You may wish to create your own subclass, which allows you to specify your own methods for painting and event handling, rather like creating a new subclass of JComponent. Before you create any portfolios, you must create an instance of PortfolioServer.

Portfolios are arranged in a tree, with each having a parent and children. The root portfolio is created automatically when you instantiate PortfolioServer. Portfolios are positioned, oriented and scaled with respect to their parent portfolio. Keyboard events and point input device events are passed firstly to the portfolio on which the event occurred and then up the tree. PortfolioLinks represent a line between specified rectangles in two portfolios.

A portfolio can also be configured to identify text that appears in its tile image and automatically identify areas that would benefit from being “unwarped rectangles” to improve legibility of text on the display.

Creating a new application

You create a new application by creating a new Java file and writing a “public static void main” method that creates an instance of PortfolioServer and then creates instances of Portfolio.

To build your application (using javac, ANT, or an IDE like Eclipse), you may need to set your class path to include the jar files in the lib directory of the distribution and the class files in the t3-build directory.

To run your application, you use something like:

```
java  -ea -Djavax.swing.adjustPopupLocationToFit=false
-Djava.util.logging.config.file=logging.properties
-classpath lib/Jama-1.0.2.jar;lib/jogl.jar;t3-
build;<YOURBUILDDIRECTORY>;. <NAMEOFYOURJAVACLASS>
```

The JVM options do the following: -ea enables assertions; the next option allows the Swing portfolios to function; the next statement instructs the Java logging system to use a configuration specified in “logging.properties”; and the classpath contains two jar files, the t3 class files and your own class files. If the application frequently runs out of heap space then you may need to edit the scripts to specify additional options to

allow more heap space. Obviously if you wish to run your application using an IDE like Eclipse (and you probably should, in order to use a step through debugger and other useful development tools) then you will need to configure the run environment carefully to match these options.

Once the application is running you then run the client, as described previously, to view the interface.

For more details, please see the example programs and the Portfolio javadoc comments.

Using Swing Components in Portfolios

By using or subclassing SwingFramePortfolio, you can create portfolios whose tiles appear as frame, into which you can add Java Swing components that respond to keyboard and point input device events. Some limitations apply.

For more details, please see the example programs and the t3.portfolios.swing.SwingFramePortfolio javadoc comments.

Debugging

T3 programs should exit with a stack trace to the standard error stream whenever an unhandled exception (or other Throwable) arises, even if generated by the AWT event processing thread. If this should occur, look carefully down the stack trace to find the cause of the exception. Using an IDE or log files makes the trace more readable.

For convenience, programs can be tested using a conventional desktop computer with a single monitor before trying them on the tabletop. The points in the client configuration file correspond to points at the corners of the framebuffer. Thus the configuration file should look something like this (multiply the numbers by a scalar constant depending on the scale at which you wish to view):

- serverName=localhost
- serverPort=2000
- clientId=0
- shareTexturesBetweenContexts=false
- hrd.projectorConfig.graphicsDeviceIndexes=0
- hrd.projectorConfig.0.fullScreenExclusive=false
- hrd.projectorConfig.0.DESKcorner0x=0
- hrd.projectorConfig.0.DESKcorner0y=0
- hrd.projectorConfig.0.DESKcorner1x=0
- hrd.projectorConfig.0.DESKcorner1y=600
- hrd.projectorConfig.0.DESKcorner2x=600
- hrd.projectorConfig.0.DESKcorner2y=600
- hrd.projectorConfig.0.DESKcorner3x=600
- hrd.projectorConfig.0.DESKcorner3y=0.0

- hrd.projectorConfig.DESKvisibleRectMask.0x=0.0
- hrd.projectorConfig.DESKvisibleRectMask.0y=0.0
- hrd.projectorConfig.DESKvisibleRectMask.1x=1000.0
- hrd.projectorConfig.DESKvisibleRectMask.1y=1000.0

The sample configuration file supplied looks like the above.

In the absence of any point input devices, the system will use the mouse as point input device 0 (or 1 if the ctrl key is held down), with personid 0 (or 1 if ALT is held down) with mouse buttons corresponding to pid buttons 0 (left), 1 (right) and 2 (middle, i.e. clicking the scroll wheel). Keyboard input appears to come from person 0 (or 1 if ALT is held down).

T3 uses the Java logging API, which can be configured by altering the supplied logging.properties file. Each class sends log entries to a Logger with the same name as its package. The properties file determines what happens to these entries. The supplied properties file instructs the logging system to ignore any entries less important than INFO (i.e. to ignore FINE, FINER and FINEST) and to output everything else both to the console and to a log file in the logs directory. You can alter the properties file to obtain information that might be useful for debugging. In particular, to see all traffic between the client and the server, look at the FINE log entries from the t3.remotehrd.client (in the client's log) or t3.remotehrd.server (in the application's log).

To output your own entries to a log, add the following to the top of your classes:

```
private static final Logger logger=Logger.getLogger("MYPACKAGENAME");
```

You can then call methods on logger to output log entries. If you use the supplied properties file then any entry of INFO or more importance will appear in the log file and on the console.

InputSources

InputSources are like clients that provide point input device data to the server but do not have a display themselves. You might use an InputSource to provide an extra keyboard for a client.

Create an InputSource by running run-inputsource.bat supplying the name of a properties file containing:

- The id of the client with whom the input is associated
  - clientId=0
- The name and port on the server to connect to
  - serverName=localhost
  - serverInputSourcePort=2001
- PointInputDevice configuration as with the client's configuration file.

History

For version 1.0.2:

- Client
  - Bug fixes
  - Texture sharing option to share textures between the different opengl contexts.
    - Need to add shareTexturesBetweenContexts=false to client config file
  - Client now specifies textures as RGBA but doesn't specify colour depth. Colour depth will probably match the Windows display mode so for faster performance change your display mode to a lower colour depth.
  - Warns when textures become non-resident.
  - Fixed button interpretation for wintab
  - Higher update rate for point input devices
  - ALT modifies personid (keyboard and mouse) whereas CTRL modifies pentype (mouse)
- Server swing
  - Bug fix
  - Multiple keyboard focuses so different ppl can enter text to different portfolios and swing components.
- Server RotateNTranslate and others
  - Better and also a new RotateNTranslate with translate only region in centre.
  - Can specify multiple pidtypeandbutton's
- Architecture changes
  - Can copy pixel data from tile to tile
  - Added inputsources
    - Extra port required when instantiating PortfolioServer
  - Added clientid:
    - Need to add clientId=0 to client config file
  - Personid for pids now in config file

For version 1.0.3:

- Needs Java 1.6
- Client
  - Eights not sixths for calibration points
  - Corners not calibration points in config file
  - Doesn't use blend texture any more
  - Displays now all perform back-buffer-flip at the same time
  - Calibration utility bug fixes
- Server
  - Bug fix for links
- Architecture
  - Unwarped Rectangles
    - Now checks for rotation and scale before deciding whether to use them
    - Now have background colour

    - Added texture residency options for tiles
    - serialVersionUID added to protocols.
- New limitations apply to server swing – see javadoc for details.

For version 1.0.4:

- Lots of changes!

Known Issues

- Tiles where parts of the tile are massively off screen (eg 9m off screen) do not get their textures loaded correctly onto the graphics card as the massive coordinates mess up nvidia's opengl drivers.

Third Party Components Used by T3

T3 makes use of:

- The JAMA package (developed by MathWorks and NIST and released by them into the public domain).
- JOGL (available from https://jogl.dev.java.net/ under the BSD licence).
- Jwintab (written by Jun Rekimoto, available from http://www.csl.sony.co.jp/person/rekimoto/java/jwintab/ under the LGPL licence).
- Parts of the R3 toolkit (Copyright Stanford University, written by Ron Yeh and available from http://hci.stanford.edu/paper/ under the BSD licence).