Virtual Reality Technology and Programming

TNM053

Matt Cooper

Lectures

1. Introduction and Unix environment.
2. C programming.
3. OpenGL programming.
4. Human Factors.
5. Scene Graphs.
8. Auralization and Audio Control and Feedback.
10. Interaction Techniques (continued).
12. Summary, Revision and Discussion.

Lab exercises

Linux Lab:
1. Unix programming (unassessed)
2. C programming
3. OpenGL programming
4. Scene Graphs
VR Lab:
5. Vortex Workbench
6. Haptics Workstation

Lab assistants

- Matt Cooper
- Anders Henrysson
- Samir Rugovaj
- Karljohan Lundin (haptics lab)
- Others filling in.

Labs: Points to note.

- Mailing list: tnm053@itn.liu.se
  - For questions about lectures and labs.
- Final date for lab demonstrations:
  - Thursday 18th December
  - Will be time during later exam periods.

Labs: More points

- There are not enough computers in the Linux labs for one each.
  - Two groups must be defined (A and B).
  - Must work in 2’s (or 3’s).
- Fill in names of each group on the sheets at the front.
Labs, yet more points

- The VR lab has:
  - 3 haptic stations
  - 1 VR workbench driven by Linux PC
- This makes life difficult with these labs
- Don’t have to do the labs in order!
  - (Except 3 and 5?)

VR Lab

- VR-lab equipment must be booked
  - http://nvis.itn.liu.se/booking
  - Username: tnm053
  - Password: _______
- Do not book more than 4 contiguous hours!
- Do not make bookings and not turn up
  - Cancel booking if you can’t make it!
- Develop on other Linux machines as much as possible.

VR lab access

- You must complete and return the ‘intyg’ to me.
  - Must have read and understood the rules.
  - Gets access to lab with your key-card.

Virtual Reality technology and Programming

Lecture 1: Unix

Matt Cooper

Lecture plan

1. General background on Unix
2. Available documentation
3. Operating System
4. How do we use Unix?
5. The shell – your working environment in Unix
6. Stdin, stdout and errors
7. Networking, Remote accesses
8. Emacs
9. The X windowing system
10. Processes and communications

Unix: General background

- Developed in late 1960’s
- First used in 1969
- Unix ‘epoch’ begins January 1st, 1970
- Usual features:
  - Manage resources (CPU, Disk, memory etc).
  - Control security and protect the user.
  - Provide secure use of shared resources.
### Unix: General background

- Multi-tasking:
  - Many programs can be executed concurrently.
- Multi-user:
  - Many users can log in and use the computer concurrently.
- Real-time:
  - Fast, light, responsive O/S.
  - Predictable maximum response times.
- Standard Unix is not a real-time O/S
  - Does have many features of one.
  - Can be (and has been) made into one.

### Standards

- Unix is well defined
- Covered by a number of standards
  - Mostly 'open' standards
- POSIX is the most important.
  - Portable Operating System Interface for Unix
  - ‘POSIX compliance’ is much sought after by Unix vendors and developers.

### Unix: Windowing

- Does not have a windowing interface
  - Not like MS-Windows.
- Several systems have been made available
  - X is, now, by far the most common

### Online documentation

- Provided through the command ‘man’
- Provides:
  - Formatted documents
  - On per command basis
  - Simple search function

### Using ‘man’

- Type ‘man command’
- Get back:
  - Name of command.
  - list of options (typically lots!).
  - Descriptions of:
    - Base functionality.
    - Effect of each option.
  - Perhaps example(s) of use.

### Sample: ‘man ls’

- What is
- How you write it
- What it does
- Options
Using man to search

- ‘man man’ will tell you about ‘-k’
  - keyword search
- grep the keyword list for matches
- Can locate command you want.
- Will also produce lots you don’t!

man sections

- man pages are organized into sections
  - base commands
  - specific languages
  - scripting languages
  - system functions, etc. etc.
- e.g. man 3 printf
  - Gives man page for the printf function
  - from section 3 - system functions

The Operating System - What do you see?

- A bunch of programs with silly names
  - ls, cd, cp, rm, mkdir, mv, find, grep, sed, awk...
- Full access to the resources of the O/S
  - fork, exec (processes)
  - brk/sbrk (memory)
  - open, read, write, close (files)
  - Linux has about 190 basic system calls:
    - Linux: Torvalds started Linux out of a fascination with the idea that Unix can be implemented using just four!
    - open, read, write, fork

Working with Unix

- Often used as server system:
  - e-mail, files, databases, WWW, DNS...
- Because it’s:
  - Reliable
  - Secure
  - Efficient

Working with Unix

- Also makes a good development platform for programming.
- Good stable compilers
  - most of the time!
- Produces efficient code
  - and an efficient execution environment

The Shell

- The basic user environment
  - what you see when you log into a Unix system
- Like an MS-Windows ‘cmd’
  - ‘DOS prompt’
- Provided with a bunch of commands
  - Either built-in
  - Or external
Unix shell commands

- `ls` - list files and directories
- `rm` - remove file and/or directory
- `mv` - move (or rename) file/directory
- `cp` - Copy file or directory
- `cd` - Change current directory
- `pwd` - Show current directory

Shell environment variables:

- `HOME` - Filespec of your home directory
- `PATH` - Places to look for a program to execute
  - eg. `/usr/local/bin:/usr/bin:/bin`
- `TERM` - Type of terminal you are using.
- `LD_LIBRARY_PATH` - Places to look for shared libraries.

Shell variables

- **Set with:**
  - `TERM=“xterm”`
- **Exported into the environment with:**
  - `Export TERM`
- **Value extracted with:**
  - `echo $TERM`
  - `or... echo ${TERM}`

Shell variables - `${}`

- **Need to use `${}`:**
  - when appending
  - When prepending
  - When using more complex forms
- `Echo $TERMfred` fails
- `Echo ${TERM}fred` works

Dot-files (hidden files)

- In your home directory.
- Filename prepended by `.`
  - e.g. `.login`, `.profile`, `cshrc`, `.xsession`
  - Not visible to normal commands like `ls`
  - Use `ls -a` to see them.
- **Used to:**
  - define environment.
  - setup aliases for common commands, e.g.
    - alias li="ll -i"
    - alias back="cd -`
  - Define initialization for programs (`.emacs`, `.netscape`)

Shell script

- Shell ‘script’ is an interpreted programming language with:
  - Functions
  - Loops
  - Conditionals
  - Arguments
  - String processing
- You won’t need to know that.
stdin, stdout and stderr

- Every Unix process has three standard file handles available on execution:
  - stdin – Standard input (filespec 1)
  - stdout – Standard output (filespec 1)
  - stderr – Standard error (filespec 2)
- Usually connected to the terminal.
  - Stdin reads from keyboard
  - Stdout and stderr both write to screen

Stdin, stdout: To and from files

- Address this with special characters
  - `<`
  - `>`
- Cat `<.login` creates a file and concatenates the file to the screen
- Cat `>tmp/keyboardcapture.txt` concatenates keyboard input to the file
- Using `>>` causes the output to be appended to the file

Capturing stderr to a file

- If you want to capture the errors from a command to a file then, e.g.:
  - `cc -o program program.c > tmp/errors.txt`
- Redirects filespec 2 (stderr) to the file.
- Can then view the file using:
  - `more /tmp/errors.txt`

‘Piping’ stdout to stdin: using commands as filters

- Can connect commands using `|`
- Stdout of first process goes to stdin on the following command. E.g.
  - `ls | more`
- More complex example:
  - `cat /etc/passwd | grep “:mat” | sort `-t` `-k` `-k` `-t` `-k` `-t`

Networking and Remote access

- Explicit File transfers: FTP
  - File Transfer Protocol
- Connects with a remote computer
  - E.g.: `ftp ftp.sunet.se`
- User name/password: anonymous, matco@itn.liu.se
- Upload and download files. e.g.:
  - `get povlinux.tgz`
  - `put article.txt`
Networking and remote access

- Implicit file transfers:
  - NFS: Network File System
  - Samba (used in Linux lab)
- Transparent file structure. E.g.:
  - /home/student/putte123
  - Generally much slower than local disk access
- Not seen by user. Handled by the admin

Remote logins

- rsh/login
  - rsh: runs a program on a remote machine
  - rlogin: runs an interactive shell on a remote machine.
  - Don't use if secure shell available!
  - Probably unusable within ITN.

Remote logins - Secure

- ssh: Secure Shell
  - Like rsh except with encryption and secure authentication – Much safer!
  - Runs a shell/command on a remote machine:
    - ssh -l matco matrix.itn.liu.se
  - scp: Secure Copy
    - Copy files to/from a remote machine
    - Connection is authenticated
    - Transfer is encrypted.
    - e.g.: scp -l matco matrix.itn.liu.se:myfiles.tar .

Emacs

- A very powerful editor:
  - Provides a powerful system for editing code
  - Not a WYSIWYG wordprocessor
  - Implemented in LISP
  - Key commands connected with lisp functions
  - Includes special functionality for editing:
    - text, tex, html, C, C++, java etc.
    - Also includes facilities for:
      - www, ftp, executing compilation, debugging, playing games, reading email, executing a command shell, etc.

X windowing system

- The X windowing system is strongly coupled with Unix
  - Also available for MS-Windows
  - X is a client-server system
  - The client may also be local or remote
  - The X Server handles the windows
    - Handles frames, moving, adjusting the size, menus etc.
  - Clients can be running anywhere:
    - Programs (clients) can be running in Boston, Tokyo or Linköping and displayed on the local computer.
    - Clients interface with the local computer’s X server
X windows system

- X is a server
  - Clients connect to server (grafikterminalen)
  - Use the X protocol (over TCP/IP or direct sockets)

### Diagram

```
           X Server
            ↓
 CLIENT (QuaEll)
 CLIENT (Xterm)
 CLIENT (Emacs)
 CLIENT (VMWare)
 CLIENT (PhotoPaint)
```

- Powerful graphics functions such as OpenGL are implemented through X
  - GLX
  - WireGL - designed specifically for remote GL

- GL-based clients can execute locally or remotely
- GL functions are passed to the X server and executed to produce final rendering

X windows clients

- Common clients
  - xterm (eterm, kterm, ...)
  - emacs (or xemacs)
  - web browser (netscape, mozilla, konqueror etc)
  - acrobat or xpdf

- Window managers
  - kwm, enlightenment, gwm, twm etc etc.

Processes & communications

- A process is an instances of a program
- Each process runs one program
  - It consists of code (program), data and stack
  - Also includes state information
- Managing processes:
  - ps - lists processes (ps -u matco)
  - kill - sends signals to a process
    - kill -9 3445 - cause process 3445 to exit
  - top - Produces a sorted list of executing processes

Processes and communication

- Each process has an address space
  - All modern systems provide virtual memory
  - Segment:
    - Code/text
    - Data
    - Stack
  - Preallocated data
  - Heap (global data)
  - Dynamically allocated variables
  - Local data

```

### Diagram

```
           DATA
            ↓
           CODE
            \-
           STACK
```

- Programs can consist of more than one process
- In order to work effectively they must communicate.
- They can do this several ways:
  - Sockets (local or TCP/IP)
  - Shared memory
  - Semaphores
  - named pipes (look like files but actually pipes)
### Unix File Structure

- **Standard programs**: `/bin, /usr/bin, /usr/local/bin`
- **Home directories**: `/home, /usr/users, /usr/home`
- **Program substructures**: `/usr, /usr/local`
  - E.g.: `/usr/X11R6/{bin|lib|include}`
  - Or: `/usr/local/X11R6/{bin|lib|include}`
- **Program libraries**: `/usr/lib`
- **Program header files**: `/usr/include`
- **System configuration files**: `/etc`
- **Temporary files**: `/tmp, /usr/tmp`
- **Devices**: `/dev`
  - Examples: `/dev/hda, /dev/tty`
- **Log files, print queues, e-mail queues etc.**: `/var`

### Course home page

- [Course home page](http://www.itn.liu.se/~matco/TNM053/TNM053.html)
  - Lecture notes
  - Papers
  - Lab exercises
  - News and other information about the course