Getting started with vrg3d

CSCI 2370, Fall 2014
1. Get sample code
2. Make sure you can build it
3. Make sure you can run it
4. Understanding the sample application
5. Getting your app to the Cave
Get sample code
Get sample code

- Our sample application that runs in the Cave and uses a library called `vrg3d`
Get sample code

- **vrg3d** is a library that will let you build a C++ OpenGL application that runs on different types of display

- Lets you develop a Cave application outside the Cave
Get sample code

- Copy tar into your home directory or sandbox

$ cp /research/graphics/tools/linux/import/vrg3d/vdemo.tgz ~/

$ tar -xzvf vdemo.tgz
Make sure you can build it
Make sure you can build it

• You now have C++ files for a Cave application

• To build the application, you’ll need to run a Makefile target that grabs the vrg3d libraries from the filesystem

  • The Makefile expects these to live in a repository at the path $G
Make sure you can build it

- **G** is an environment variable you need to define in your shell

- Try the command `echo $G` to make sure this isn’t already defined (if it is, talk to us)

  - If it outputs nothing, you’re good
Make sure you can build it

- Define G as the repository that's been checked out onto the CS filesystem

```
$ export G=/research/graphics/tools/linux
```
Make sure you can build it

- Run the makefile in the new vdemo directory

  $ cd vdemo

  $ make
Make sure you can run it
Make sure you can run it

• Run the newly built demo applications

$  ./chess desktop

$  ./demo desktop
Make sure you can run it

- Run the newly built demo applications

  $ ./chess desktop

  $ ./demo desktop
$ ./chess
No argument specified for a VR setup to load.
Continuing with a default desktop setup.

Installed VR setups are listed below:

multitouch-client Basic config for the multitouch projector
cavefront-winservmono Brown's CAVE - front wall using windows vrg3d-server (mono rendering)
msi-rightwall Brown's CAVE - front wall using windows vrg3d-server
fishtank304nodevicesmono The fishtank VR setup in the Brown user study room (CIT 304)
brown_tiledwall_tile5 Brown CCV tiledwall tile 9 with event server on audio-cave
desktopOmnimap A standard non-VR desktop window
multitouch Basic config for the multitouch projector
jvc-kinect A standard non-VR desktop window
fishtankmono A fishtank VR setup with no devices (mono display)
3DDLP A standard non-VR desktop window
samsung-with-devices A standard non-VR desktop window
msi-frontwallL10 Brown's CAVE - front wall using windows vrg3d-server
samsung-devices-prop Unknown description
msi-frontwallL11 Brown's CAVE - front wall using windows vrg3d-server
caveright Brown's CAVE - right wall
msi-frontwallL12 Brown's CAVE - front wall using windows vrg3d-server
jvc-with-forceserver A standard non-VR desktop window
cavelwall Brown's CAVE - run on the front wall only
msi-frontwallL15 Brown's CAVE - front wall using windows vrg3d-server
cavefloor Brown's CAVE - floor wall
msi-frontwallL16 Brown's CAVE - front wall using windows vrg3d-server
pt A standard non-VR desktop window
msi-frontwallL19 Brown's CAVE - front wall using windows vrg3d-server
samsung-with-devices-optitrack-pt A standard non-VR desktop window
radimenzo A standard non-VR desktop window
cavelfloor Brown's CAVE - left wall
multitouch-zspace-client Basic config for the multitouch projector
multitouch-stereo Basic stereo config for the multitouch projector
umnncave University of Minnesota Interactive Visualization Lab CAV
Understanding the sample application

• Makefile tells us that the chess executable is built from code in `chess.cpp` and `chess_demo.cpp`
Understanding the sample application

- Makefile tells us that the chess executable is built from code in `chess.cpp` and `chess_demo.cpp`

  - Defines data structures, drawing code specific to the application
  - the application code containing the main(); event handling; deals with vrg3d
Understanding the sample application

- chess.cpp
- init_pieces(), draw_pieces()
Understanding the sample application

- `chess_demo.cpp`
  - Constructor calls `init_pieces()` from `chess.cpp`
  - `drawGraphics()` calls `draw_pieces()` from `chess.cpp`
  - `doUserInput()` handles wand/keyboard/mouse
Understanding the sample application

- **chess_demo.cpp**
  - probably don’t need to mess with other code in constructor, other code in `doGraphics()`, or any of `main()`
Getting your app to the Cave
Getting your app to the Cave

• First, get a user account for the Cave

  • Apply for a CCV account online at ccv.brown.edu/start/account

  • Then send a note to john_huffman@brown.edu explaining that you want a matching account on the Cave machines for CS237
Getting your app to the Cave

- Now, about getting your code to the facility...
Getting your app to the Cave

• The quick and dirty way
• The cleaner way
Getting your app to the Cave

- The quick and dirty way - copy it over ssh (scp)
- The cleaner way - check it out from source control
Getting your app to the Cave

• The quick and dirty way - copy it over ssh (scp)
Getting your app to the Cave

• The quick and dirty way - copy it over ssh (scp)

• From your development machine, ssh into CCV, then the Cave server:

  $ ssh user@ssh.ccv.brown.edu
  <blah blah welcome message>

  [user@login002 ~]$ ssh user@10.2.128.224
  <blah blah welcome to cave-server>

  [user@cave-server ~]$ scp user@ssh.cs.brown.edu:~/myfiles.tgz .
Getting your app to the Cave

• The quick and dirty way - copy it over ssh (scp)

• From your development machine, ssh into CCV, then the Cave server:

```
$ ssh user@ssh.ccv.brown.edu
<blah blah welcome message>
[login002 ~]$ ssh user@10.2.128.224
<blah blah welcome to cave-server>
[cave-server ~]$ scp user@ssh.cs.brown.edu:~/myfiles.tgz .
```

• From inside the Cave server, scp from the CS filesystem:

```
[user@cave-server ~]$ scp user@ssh.cs.brown.edu:~/myfiles.tgz .
```
Getting your app to the Cave

• The cleaner way - check it out from source control
Getting your app to the Cave

• The cleaner way - check it out from source control

• Use the graphics group CVS repository
Getting your app to the Cave

• The cleaner way - **check it out from source control**

• Use the graphics group CVS repository

• See **README.txt in vdemo/** for detailed instructions toward the end of the document — these slides contain only the broad strokes
Getting your app to the Cave

• Step 1: ‘Checkout’ existing repository from the CS filesystem

   $ cvs -d /research/graphics/cvsroot checkout project/class/cs237/2014

• Step 2: Read the README in this new directory

   • Follow the rules for the code repository, which is a shared space for code
Getting your app to the Cave

• Step 3: Make a directory for your project code inside project/class/cs237/2014/

• `cvs add` your code to your new directory

• `cvs commit` often and do it immediately before you head to the Cave

```bash
$ cd project/class/cs237/2014
$ mkdir myproject
$ cvs add myproject
$ cvs commit
$ cd myproject
```
Getting your app to the Cave

• Step 3.5: Develop and test your application
Getting your app to the Cave

• Step 4: From the Cave server, check out your code from CVS
Getting your app to the Cave

From the README in vdemo/,

If you copy the ~/.ssh/id_rsa.pub file contents from the cave machines into your ~/.ssh/authorized_keys file at CS, you can check things into and out of the viz CVS tree ($G) directly. See the CVS instructions below.
Getting your app to the Cave

• Step 4: From the Cave server, checkout your code from CVS

$ cvs -d :ext:yourLogin@ssh.cs.brown.edu:/research/graphics/cvsroot checkout project/class/cs237/2014/myproject
Getting your app to the Cave

- Step 4: From the Cave server, checkout your code from CVS

```
$ cvs -d :ext:yourLogin@ssh.cs.brown.edu:/research/graphics/cvsroot
cHECKOUT project/class/cs237/2014/myproject
```

* Only need to do this checkout once! *
Getting your app to the Cave

• Step 5: Each time you visit the Cave, you can pull any updates that you’ve committed

$ cd project/class/cs237/2014/myproject
$ cvs update -d
Getting your app to the Cave

• Step 6: Your code is in the Cave now. Follow the Cave run instructions in your README (which might include Cave directions from vdemo/README.txt)