Remote visualization – VirtualGL

Paul Melis

SARA Visualization Group
(paul.melis@sara.nl)
Overview

- Remote visualization
  - Why? What? How?

- VirtualGL
  - Popular package for remote visualization
  - Background, usage, features, ...

- TurboVNC
  - Good way to provide access to remote visualization applications

- SARA Remote Visualization Service
Large-scale visualization

- Large-scale parallel computations usually produce large-scale output data
  - Gigabytes to terabytes (or more) data
  - → In general also need parallel processing and rendering in order to visualize
  - → Fast access to data storage necessary

- Approach A: visualize on HPC system where data was produced
  - Problem #1 - HPC systems usually batch-oriented
    - Creating visualizations has important interactive component
    - Queue: need to wait for job to start, unpredictable when it runs
  - Problem #2 - GPUs often not available in HPC systems
    - Software-only 2D/3D rendering possible, but on the order of 5x - 50x slower than using a GPU
Large-scale visualization (2)

- Approach B: visualize locally
  - I.e. on user's own PC
  - Problem #1 - Need to transfer data and store it locally
    - Network bandwidth? Storage?
  - Problem #2 - Enough CPU/GPU resources available?
Large-scale visualization (3)

- **Approach C: remote visualization**
  - Provide dedicated visualization resources in data center
  - Visualization resource has direct fast connection to central storage
  - Avoid large data transfers over external network by running visualization application remotely
  - Data stays in data center, only pixels/images sent to external user
Interactive visualization applications
- ParaView already remote (client-server)
- What about other applications?

Remote rendering using Linux/Unix application and servers

2D/3D rendering with OpenGL
- Low-level API for drawing 2D and 3D content
- All self-respecting graphics cards support it
- De-facto standard for rendering on Linux/Unix

/* OpenGL snippet: */
/* clear window */
glClear(GL_COLOR_BUFFER_BIT);

/* draw unit square polygon */
glBegin(GL_POLYGON);
glVertex3f(-0.5, -0.5, 2);
glVertex3f(-0.5, 0.5, 2);
glVertex3f(0.5, 0.5, 1);
glVertex3f(0.5, -0.5, 1);
glEnd();

/* Swap GL buffers */
glSwapBuffer();
Remote Visualization - Concept

■ Dedicated visualization resource
  □ Usually cluster of render nodes
    ▷ One or more GPUs per node
  □ (Fast) interconnect between nodes
  □ Fast connection to central storage
  □ Reasonable connection to outside world
  □ User connects to resource and only receives visualization output

■ User advantage: facility maintained by HPC center

■ Open questions
  □ How to work with applications remotely?
  □ What software is supported/what isn't?
  □ Parallel rendering?
Remote visualization – X Windows?

**X Windows (a.k.a. X11)**
- Used in all modern Linux desktop installations
  - X.org server
- Client-server model

**X Server**
- Client applications connect to server
- Receives input from user (keyboard, mouse, tablet, etc)
- Forwards input to correct application
- Applications tell X server what to draw on the screen, etc.
- X server manages screen contents
- Communication between clients and server using the X Protocol

**Powerful feature:**
- Run X11 applications remotely
- I.e. X Protocol over network connection
  - “X forwarding”
- **Local X11 application:**
  - `local$ my-vis-app`

- **Remote X11 application:**
  - `local$ ssh -X <remote machine>`
  - `remote$ my-vis-app`
  - Visualization application runs on remote system
  - Application GUI displayed on local system
  - Keyboard/mouse input from local system forwarded to application on remote system
“OpenGL eXtension for the X Windows System”

Application that wants to draw with OpenGL:
- App uses GLX to get OpenGL context
- App draws 2D/3D primitives with OpenGL commands
- X Server takes care that commands get to OpenGL library
- App → GLX → X Server → OpenGL library → GPU → Screen

Direct Rendering Interface (DRI):
- App → OpenGL library → GPU → Screen
- Only for applications running on the same system as the X server (app needs direct access to GPU)

Remote applications can only use GLX over network ("indirect rendering")
- Lots of 2D/3D geometry or textures might be transferred from server to client with GLX
- Possibly new data each frame
  - E.g. animated isosurface, 2D textured slice being moved through a volume, etc.
Other GLX disadvantage

- Subset of OpenGL features available when using indirect rendering
  - Certain OpenGL extensions need direct GPU access
- Small test on my workstation:
  - `local$ glxinfo > info.local`
  - `local$ ssh -X <remote render node>`
  - `# Note: no X server running on remote node!`
  - `# glxinfo connects to X server on local workstation`
  - `remote$ glxinfo > info.remote`

Local:
OpenGL version string: 4.2.0 NVIDIA 295.40
OpenGL shading language version string: 4.20 NVIDIA via Cg compiler
...

Remote:
OpenGL version string: 2.1.2 NVIDIA 295.40
OpenGL shading language version string: (null)
<lots of extensions missing...>
General remote X11 disadvantages

- X11 protocol is quite verbose. On high-latency and/or low-bandwidth connections it doesn't really work nicely
  - E.g. home ↔ office connection
  - On gigabit LANs usually quite usable
- Need X11 server on client-side
  - Windows: either Exceed (commercial) or Cygwin/X (open-source)
Better remote rendering idea?

- We want to run applications remotely, as if they're running on our desktop
- But want to use a server-side GPU

- How?

- This is where VirtualGL comes in...
VirtualGL

- **History**
  - Started by Sun as part of the Sun Visualization System
  - VirtualGL open-sourced when Sun stopped supporting SVS
- **Open-source** (http://www.virtualgl.org)
- **Supported platforms**
  - Server: Unix (Linux, Solaris)
  - Client: Unix, Windows, MacOS
- **Note**: server only available on Unix-like systems, but in general those are the systems used for servers anyway
- **So how does it work?**
VirtualGL – Split Rendering

- 3D rendering commands intercepted at remote side and forwarded to remote GPU
  - 3D OpenGL rendering commands
  - Need X server (“3D X server”) and GPU on remote side now as well
- 2D commands go to local X server (“2D X server”) untouched
  - Menus, buttons, etc.
  - Basically unaltered remote X11
- Resulting image from 3D rendering read back
  - Extra image stream needed to transfer 3D image result to user's side
VirtualGL – Behind the scenes

- VirtualGL preloads a shared library that intercepts some GLX, X11 and OpenGL calls
  - New window created by application → VirtualGL creates corresponding offscreen rendering buffer (PBuffer) on remote GPU
    - Window resize → offscreen buffer adjusted
  - OpenGL draw commands executed → Drawing is done on Pbuffer
  - Application finishes rendering frame → Pbuffer contents read back and 3D image sent to client
  - All other calls left untouched

- All OpenGL features available, because Pbuffer on local GPU is used
  - No missing extensions, etc.
  - If application runs locally on a server, it will run remotely on that server under VirtualGL
VirtualGL – “VGL Transport”

- Extra network connection used from remote to local side, to send separate 3D rendered image
- Rendered 3D images are inserted in application in the correct subwindow
- All done behind the scenes, when using VirtualGL:
  - local$ /opt/VirtualGL/bin/vglconnect <user>@<remote>
  - remote$ /opt/VirtualGL/bin/vglrun <app> ...
VGL Transport

Advantages

- Application uses remote 3D rendering, but acts as a local application
- 3D rendered parts are sent to user's side using on-the-fly compression, and only sending those parts of the image that change
- Supports stereographic rendering
  - Quad-buffered rendering on server, displayed on client (both need stereo graphics)
  - Or, combining into anaglyph images on the server (only server needs stereo)

Disadvantages

- Based on remote X11 protocol, application can be slow to update on low-bandwidth or high-latency networks
  - Interestingly, 3D parts (processed separately by VirtualGL!) update much faster than the 2D menus, buttons and such
- Need X server on local machine
  - On Windows somewhat cumbersome
- Application state not preserved, when disconnected session is lost as the application terminates
Image delivery method #2: VNC (X Proxy)

- **X Proxy**
  - "Virtual X server"
  - Acts like an X server, but provides only minimal functionality
    - E.g. no keyboard/mouse input
    - No access to GPU
  - Most useful proxy is VNC server
    - "Virtual Network Computing"
      - Similar to Remote Desktop Protocol (RDP) on Windows
    - VNC is a widely used remote desktop technology
      - User connects VNC viewer to VNC server
      - Server sends back updated desktop image, shown in VNC client
      - User's keyboard and mouse input are transferred to application on VNC server
    - Efficient transfer of changes in the desktop image
      - Desktop image divided in tiles
      - Only tiles that have changed since last frame sent to client
      - Different compression method per tile possible
VGL Transport (left)  X proxy (right)

User's workstation

Keyboard --> Mouse --> Screen

2D X Server

X client
my-vis-app (local)

X client
my-vis-app (remote)

GPU

VGL

X11 protocol

VGL Transport (left)

Remote machine

GPU --> 3D X Server

OpenGL

VirtualGL

GPU

3D X Server

VirtualGL

X client
my-vis-app (remote)

X client
my-vis-app (local)

User's workstation

Running a VNC viewer

VNC protocol

TurboVNC

(2D X proxy)

Remote machine

GPU --> 3D X Server

OpenGL

VirtualGL

GPU

3D X Server

VirtualGL

X client
my-vis-app (remote)
TurboVNC

- Preferred X proxy for VirtualGL
  - Good integration with VirtualGL
  - VirtualGL and TurboVNC maintained by same software developers
  - TurboVNC was specifically created to handle interactive 3D and video workloads
    - Uses optimized JPEG compression routines
- Stable, good documentation
- Supported platforms
  - Server: Unix (Linux, Solaris)
  - Client: Unix, Windows, MacOS
- See http://www.virtualgl.org
TurboVNC - Usage

■ **Server node**
  ■ Set password (one-time action)
    □ $ /opt/TurboVNC/bin/vncpasswd
  ■ Start VNC server
    □ $ /opt/TurboVNC/bin/vncserver :1 [-geometry <width>x<height>]

■ **Client node**
  □ Connect with VNC viewer from client machine
    □ $ vncviewer <server>:1

■ Stopping
  □ $ /opt/TurboVNC/bin/vncserver -kill :1

■ Listing active VNC servers
  □ $ /opt/TurboVNC/bin/vncserver -list

■ Server log files in $HOME/.vnc
  □ Contains info on connections, compression settings, etc.
  □ Useful for debugging connection problems

■ Can put extra environment options in startup script $HOME/.vnc/xstartup.turbovnc
TurboVNC + VirtualGL

- Visualization applications running inside a VNC session need access to GPU
  - The VNC server itself has no access to GPU, as it is only a virtual X server, not a real one
  - VirtualGL will take care of 3D interception similar to VGL Transport method

- Usage in practice, again very simple:
  - Login in on server machine, start TurboVNC server
  - Connect with VNC client to server
  - Use `vglrun` command when starting application in VNC session, e.g.
    - `$ vglrun VolView`

- Small demo
TurboVNC - Options

- In viewer: F8 key brings up menu
  - Toggle fullscreen on/off
  - Send ctrl-alt-del, send F8
  - Set compression and JPEG quality settings
  - Request lossless refresh (Ctrl-Shift-Alt-L)

- Multi-threaded compression on server possible
  - Screen divided in N horizontal strips, encoding and compression done by separate thread per strip
  - remote$ TVNC_MT=1 vncserver ...
  - By default same # of threads as cores, use TVNC_THREADS=<n> to override
(Turbo)VNC - Security?

- Basically none provided in the RFB protocol!
  - Although passwords aren't sent in cleartext when authenticating...
  - ...*keystrokes and desktop images in a VNC session are unprotected!*
  - E.g. passwords you enter *within* a VNC session to login to other machine are going over the network in the clear

- How paranoid should you be about this?

- Use SSH tunneling
  - Will cost a bit of performance, but still very usable
  - $ ssh -L 590<d>:localhost:590<d> <remotevis-server>
  - $ vncviewer *localhost*:<d>
  - On Linux with TurboVNC:
    - $ vncviewer -via <remotevis-server> localhost:<d>
  - Need PuTTY's plink.exe on Windows
Simple way to do rudimentary collaboration

- Multiple users can connect their VNC viewers to the same VNC server
- They will all see the same desktop image
- They can all use mouse and keyboard
  - Can lead to interesting interactions :)

**vncviewer**
- -shared (default) or -noshared options

**vncpasswd**
- Set view-only password

---

**VNC server**
TurboVNC - Compatibility?

- Compatibility between clients and server versions
  - Underlying RFB protocol is platform-agnostic
  - Different “brands” of VNC (RealVNC, TightVNC, TigerVNC, TurboVNC, Chicken of the VNC, ...) should work with TurboVNC

- Successful test:
  - TurboVNC 1.0 64-bit server on Linux
  - TurboVNC 0.6 32-bit client on Linux
  - TigerVNC 1.0.1 32-bit client on Windows
  - TightVNC 1.3.9 64-bit client on Linux
  - E.g. three clients connected at the same time :)

TurboVNC – Annoyances (from experience)

- Sometimes after switching between vncviewer window and others with ALT-TAB the keys in the VNC session become confused
  - Try to press and release the ALT key in the VNC viewer

- Garbage characters entered in the VNC session?
  - Try adding
  - `export XKL_XMODMAP_DISABLE=1`
  - to `$HOME/.vnc/xstartup.turbovnc` on the system running the TurboVNC server

- Pressing ALT+F4 in the VNC session to close a window closes the VNC viewer window
(Turbo)VNC - Summary

- Advantages
  - Better interactivity compared to remote X11, efficient network usage
    - High-speed JPEG image compression
    - Only updated parts of desktop image get transferred
  - Collaboration is easy, simply have several people connect to same VNC server
  - If connection from client to server is lost, remote desktop will stay alive as long as VNC server keeps running. Simply reconnect with client.

- Disadvantages
  - Need to open up extra TCP port(s) on the remote machine for incoming VNC client connections
    - VNC :<d> will listen to TCP port 590<d>
    - Use SSH tunneling
  - Need to use SSH tunneling when security matters
  - VNC client window shows remote desktop in a window on your normal desktop, can be confusing with keys, focus, mouse, etc
TigerVNC?

- Much nicer GUI in client application for setting VNC options
- Server not directly focused on 3D and video applications, use TurboVNC instead

- More detailed comparison of TurboVNC and TigerVNC:
  - http://www.virtualgl.org/About/TigerVNC
VirtualGL - Capabilities

- In principle any OpenGL-based application supported
  - Without modification to the application

- Multiple remote users can simultaneously work on the same GPU
  - E.g. for more efficient use of render nodes when load per user is light
  - Most visualization applications do not spend all time per frame drawing using the GPU

- But... no abstraction of multiple GPUs and/or render nodes into single GPU resource
  - No magic parallel rendering for applications
  - Application needs to provide support for that
    - E.g. ParaView client-server
VirtualGL/TurboVNC - Concluding

- In almost all cases the provided scripts make it just work
  - Where it doesn't work the manual and/or mailing list can help
- Provides a general way to use remote rendering for visualization applications
- TurboVNC mode
  - Very easy to use, simple method of collaboration, session stays alive when disconnecting, usable over all types of networks
- VGL Transport mode
  - Works best on high-speed, low-latency networks
  - In case you want the actual application under your hands, not just a remote desktop showing that application
VirtualGL/TurboVNC - Experiences

- Very stable product
  - Works perfectly on SARA's visualization cluster

- Performance on local (SARA) network is excellent
  - When using remote cluster from home not to shabby either

- Very good documentation at www.virtualgl.org
  - Installation, configuration, etc.
  - Optimizing settings for different network environments
  - Advanced topics such as VirtualBox integration

- Open issue in (remote) visualization:
  - Real virtualization of GPU resources
Remote visualization setup

at

SARA
Dutch national supercomputing center
- Located on Science Park Amsterdam
- Around 60 employees
- Lots of users from Dutch universities and other institutes
- Very diverse scientific fields
- Active in national and international projects (PRACE-[123]IP, HPC-Europa, ...)

Facilities
- Huygens supercomputer
  (3,456 cores, 65 TFLOP/s peak, 15.75 Tbyte RAM, 700 Tbyte disk)
- Lisa compute cluster
  (4,480 cores, 12 Tbyte RAM, 20 TFLOP/s)
- HPC Cloud cluster “Caligo”
  (608 cores, 4.75 Tbyte RAM)
- Remote visualization service

Other areas of expertise
- E-Science
- High-performance networking
- Mass storage
- Visualization
  - Yours truly...
SARA Remote Visualization Service

- Pilot since Q1 of 2010, in production since Q1 of 2011
- 16 visualization nodes + 1 login node
  - Quad-core Xeon (2.4 Ghz), 12 Gbyte RAM, 800 Gbyte scratch/node
  - Nvidia GeForce GTX 460 GPU
  - Debian Linux 6.0 (x86_64)
- Direct connection with Huygens' GPFS file system
  - Through NFS exports
- Node interconnect is Gigabit ethernet
  - Sufficient so far
- 10 Gbit/s lightpath to rest of the world for network streaming
- Software
  - ParaView, VisIt, VTK
  - VMD, FFMpeg, ImageMagick, Blender, ...
  - VirtualGL
SARA RVS (2)

- Access using SSH
  - Only from white-listed IP addresses
- VNC ports opened on render nodes
  - SSH tunneling not needed if unwanted
  - But still possible
- ParaView server running on nodes directly accessible
- Resource reservation system
  - Torque/Maui
  - Limited usage period (max. 8 hours currently)
Initially had extensive documentation detailing lots of manual steps
  - Users complained (their good right)

We provide some scripts to make reservation and starting/stopping VNC easier for users
  - rvs_vnc <time-needed>
  - rvs_paraview <#nodes>
  - ...
SARA RVS (4)

- **Usage**
  - Mainly interactive visualization
  - Also some batch rendering of scientific visualization movies

- **Areas of science**
  - Simulations of geological activity (O(10^6) cells, ParaView)
  - Spreading of infectious diseases within social networks
  - MicroCT-based coral growth modeling (2048^3 cells)
  - Cosmological simulations
  - Ocean current simulations
SARA RVS - Areas of improvement

- User-friendliness
  - Having a simple web portal for making reservations would be nice
  - More scripts for common tasks

- Allow access PRACE users / integrate with PRACE infrastructure

- Keeping track and testing new technologies

- More users welcome

- Integration with the new Collaboratorium
SARA Collaboratorium

- Room for presentations, collaborations, visualizations
  - Opened end of March, 2012
- Easy to use for daily activities, advanced use possible with somewhat more effort
- Video-wall:
  - 4x2 Full-HD thin-bezel screens
  - Roughly 8,000 x 2,000 pixels
- Video matrix switch
  - Show (multiple) connected laptop signals on video wall
  - Scale to 2x2 displays
- Multi-touch overlay
- 3D stereo projector
- 2x 10 Gbit/s link to SARA data center
- Would be cool to call up remotely rendered high-resolution visualizations with a few touch gestures
  - Not realized yet :(
Thanks for the attention!

Questions?
VirtualGL + TurboVNC vs VGL Transport

- When using the VGL Transport, non-3D portions of the application’s GUI are sent over the network using remote X11, which will create performance problems on high-latency networks (such as broadband or long-haul fibre.) Non-3D portions of the application’s GUI will load and render much faster (perhaps even orders of magnitude faster) with TurboVNC than with the VGL Transport on such connections.

- For 3D applications whose rendered images do not contain very many unique colors (for instance, CAD applications in wireframe mode), the hybrid encoding methods used by TurboVNC will generally use less network bandwidth than the pure JPEG encoding method used by the VGL Transport.

- TurboVNC provides two lossless compression modes, one of which is designed to reduce server CPU usage on gigabit networks and the other of which is designed to provide reasonable performance on wide-area networks (at the expense of higher server CPU usage.) The VGL Transport’s only lossless option is uncompressed RGB.

- TurboVNC includes a lossless refresh feature that will, on demand, send a mathematically lossless image of the current VNC desktop to the client. A user connecting over a low-bandwidth connection can use low-quality JPEG to achieve the best performance when manipulating the 3D model, then they can request a lossless refresh when they are ready to study the model in detail.

- The TurboVNC Server can be configured to send a mathematically lossless copy of certain regions of the screen during periods of inactivity (Automatic Lossless Refresh.)

- TurboVNC provides rudimentary collaboration capabilities. Multiple clients can simultaneously view the same TurboVNC session and pass around control of the keyboard and mouse.

- The TurboVNC Viewer is stateless. If the network hiccups or the viewer is otherwise disconnected, the session continues to run on the server and can be rejoined from any machine on the network.

- No X server is required on the client machine. This reduces the deployment complexity for Windows clients.

- Any web browser or PDA can be used as a TurboVNC client (with reduced performance.)
No seamless windows. All application windows are constrained to a “virtual desktop”, which displays in a single window on the client machine.

TurboVNC will generally require about 20% more server CPU cycles to maintain the same frame rate as the VGL Transport, both because it has to compress more pixels in each frame (an entire desktop rather than a single window) and because it has to perform 2D (X11) rendering as well as 3D rendering.

TurboVNC does not support quad-buffered stereo or transparent overlays.