Using ParaViewWeb for 3D visualization and data analysis in a web browser

+ other remote visualization and 3D online presentation techniques

Alex Razoumov
alex.razoumov@westgrid.ca

copy of these slides and other files in `http://bit.ly/pvwebfiles`
(will download a ZIP file)

ParaViewWeb

- Lightweight JavaScript API for writing **client-based HTML5 web applications** to display 3D interactive visualizations in a web browser
  - open-source project from Kitware, Inc.
- Most PVW applications use a remote ParaView server to process and render data
  - a handful of prebuilt applications available
  - the most complete app is Visualizer, providing most of ParaView Qt desktop application features within a web browser
  - in principle, can build your own apps
  - source in its own repository `https://github.com/Kitware/paraviewweb`
- Small 3D geometry can be rendered locally on the client using WebGL
  - in the past had some custom WebGL code in Visualizer, currently developing vtk.js-based visualization for handling 3D scenes with WebGL; should be done by August
- PVW's core and apps normally included with pre-compiled ParaView, but not in the source

ParaViewWeb

✔ Use it if you

- want to give anyone with a browser the ability to play with your 3D dataset(s), or
- want a much simpler/cleaner or more specialized interface than provided by standard desktop tools (ParaView, VisIt), or
- want a mobile, touch-friendly interface

✘ Might want to work with native client-server if you

- want to perform heavy 3D rendering on a remote high-spec server and display results interactively (as single user) locally on your laptop
  - ➩ ParaViewWeb without WebGL and without a remote GPU will be slow
  - ➩ native client-server: faster performance with our current setup (small geometry rendering done with OpenGL on your laptop's GPU), much more functionality
    (for threshold setting check ParaView settings → Render View tab → Remote/Parallel Rendering Options, default is 20MB)
  - ➩ I'll show ParaView and VisIt client-server demos running off a VM server in a minute

Launching a cloud virtual machine (VM)

- Cloud policies for **Rapid Access Service** `http://bit.ly/2mlXpER`
  - default **testing instances**: max 1 month, 15GB RAM, 2 vCPUs, 40GB disk
  - default **compute instances**: max 1 month, 307GB RAM, 80 vCPUs, 1000GB disk
  - default **persistent instances**: max 1 year, 45GB RAM, 10 vCPUs, 1000GB disk
  - for more resources, apply through our Resource Allocation Competitions (RAC)
  - terminate/delete VMs if you don't use them
  - you are in charge of your own security, backup, software installation
- Compute Canada Cloud Quick Start Guide `https://docs.computecanada.ca/wiki/Cloud_Quick_Start`
  - log in to CC's West/East Cloud portal using your CCDB account
  - set up SSH key pair
  - configure and launch a VM (CentOS, Ubuntu, etc)
  - assign a public IP
  - configure firewall (allow SSH)
- Initial VM set up
  - install dependencies for ParaView (and/or VisIt) `http://bit.ly/2kUTCNL`

Deco cube

We'll be visualizing a 30^3 discretization of

$$\begin{aligned} f(x,y,z) &= \left[(x^2+y^2-0.64)^2+(z^2-1)^2\right] \\ &\quad \times \left[(y^2+z^2-0.64)^2+(x^2-1)^2)\right] \\ &\quad \times \left[(z^2+x^2-0.64)^2+(y^2-1)^2)\right] \end{aligned}$$

where $x, y, z \in [-1.2, 1.2]$, rendering an isosurface at $f(x,y,z) = 0.03$

```python
# To write VTK to disk, we'll use https://bitbucket.org/pauloh/pyevtk
# In the repo see src/examples/{image,rectilinear,structured,points}.py
from evtk.hl import imageToVTK
from numpy import zeros
n = 30
data = zeros((n,n,n), dtype=float)
for i in range(n):
    x = ((i+0.5)/float(n)*2.-1.)*1.2
    for j in range(n):
        y = ((j+0.5)/float(n)*2.-1.)*1.2
        for k in range(n):
            z = ((k+0.5)/float(n)*2.-1.)*1.2
            data[i][j][k] = ((x*x+y*y-0.64)**2 + (z*z-1.)**2) * \
                            ((y*y+z*z-0.64)**2 + (x*x-1.)**2) * \
                            ((z*z+x*x-0.64)**2 + (y*y-1.)**2)
imageToVTK("decoCube", pointData={"scalar" : data})
```

Client-server ParaView demo

details at `http://bit.ly/2mm199f`

- I'll be using a VM on Arbutus at UVic (West Cloud)
- Compile ParaView on the VM with OSMesa (enabling OpenSWR software rendering) and Python (to enable ParaViewWeb)
- On the VM start ParaView server by hand

  ```
  ~/paraview/bin/pvserver --mesa-swr-avx2
  ```

  (should say "SWR detected AVX2" during rendering)
- Do something about port 11111 on the VM (not open in general), e.g., set up an SSH tunnel to it from your laptop

  ```
  ssh ubuntu@206.12.97.61 -L 11111:localhost:11111
  ```

Client-server VisIt demo

details at `http://bit.ly/2mNly39`

- Compile VisIt on the VM with OSMesa and Python
- On your laptop (VisIt client) select $\boxed{\text{Options}} \rightarrow \boxed{\text{Host profiles...}}$ to set nickname (`cloud west`), host name (VM's public IP address), path to remote VisIt installation (`/home/ubuntu/visit`), username (`ubuntu`), tunnel through SSH
- $\boxed{\text{Options}} \rightarrow \boxed{\text{Save Settings}}$
- $\boxed{\text{File}} \rightarrow \boxed{\text{Open file...}} \rightarrow \boxed{\text{Host=}}$ cloud west

ParaViewWeb applications

- **Visualizer** provides an experience inside the browser which is very like the ParaView Qt desktop application, example of what can be built with ParaViewWeb

  `https://github.com/kitware/visualizer`

  `https://kitware.github.io/visualizer/docs`

- **LightViz** provides simpler, more intuitive visualization

  `https://github.com/kitware/light-viz`

  `https://kitware.github.io/light-viz/docs`

- **ArcticViewer** is a standalone (no PV server needed) JavaScript viewer for Cinema- or Catalyst- pregenerated images
- **DataViewer** is a “big data” infovis tool, currently in development
- Few other in development at Kitware
- Theoretically anyone can write their own (JavaScript)

Single-user Visualizer on a laptop

- Great for testing, not so useful for production work
- Two ways to start, either one waits for incoming traffic on port 8080:
  (1) either a Python ParaViewWeb server application (serving Visualizer), or
  (2) a standalone JavaScript Visualizer app (in Node.js runtime environment)
- If a second user tries to connect, they'll just share the same session (everything will be mirrored)
- These Visualizer components are typically included in a precompiled ParaView binary, but not into the PV source (hosted in a separate repo):
  - Python ParaViewWeb server application `pvw-visualizer.py`
  - static HTML content directory `web/visualizer/www` with Visualizer code `Visualizer.js` inside

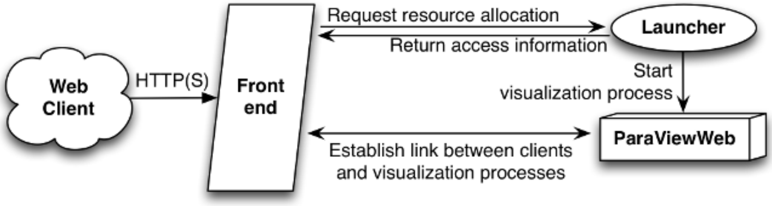
```
cd /Applications/ParaView-5.2.0.app/Contents
./bin/pvpython Resources/web/visualizer/server/pvw-visualizer.py --
    content Resources/web/visualizer/www --data ~/Documents/03-
    pvweb/data --port 8080
```

Single-user Visualizer in a cloud VM

- Install Visualizer in addition to ParaView (whose source does not include individual applications)
- Otherwise similar setup to a laptop, except that port 8080 is not open on the VM by default

not so useful as it starts only one ParaView process,
with all users sharing the same session ...

Multi-user Visualizer in a cloud VM


Request resource allocation
Return access information
Launcher
Web Client
HTTP(S)
Front end
Start visualization process
ParaViewWeb
Establish link between clients and visualization processes

(image from http://kitware.github.io)

In this demo we'll use the same machine for all components

- **Front end** serves static content and does network forwarding (single entry point for all clients), tells launcher when a new client wants to start a new visualization session
  ☞ *we'll use Apache web server (need v2.4.7 or higher)*
- **Launcher** starts a ParaViewWeb process for each user who requests one and communicates the session ID and an associated port number to the front end
  ☞ *we'll use a Python launcher*

Multi-user setup

Configuring a virtual host and a launcher requires some iterative work

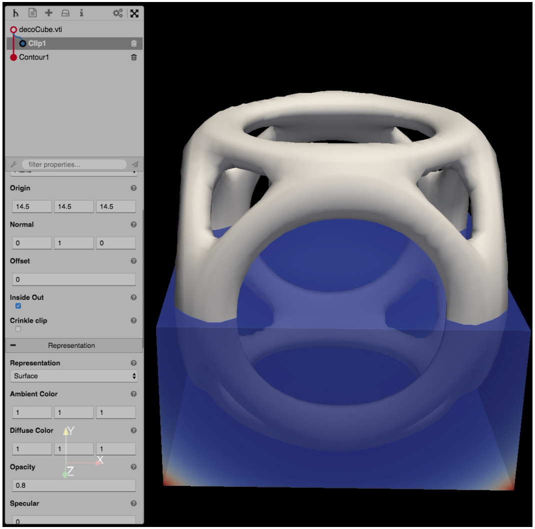
- See official documentation at `http://bit.ly/2nsViOB` for setting this in Ubuntu
- Need to configure `/etc/hosts`, `/data/pvw/conf/launcher.json`, `/etc/apache2/sites-available/001-pvw.conf`
    - ☞ you can find these in `config/pvw/` inside the ZIP download

1. put ParaView + Visualizer into `/data/pv` and data into `/data/pvw/data`
2. create a new "non-sudo" user *pvw-user* who'll be starting the launcher
3. give him ownership of `/data/pv` and `/data/pvw`
4. set up a proxy file that launcher and Apache will use to communicate with each other
5. if the launcher does not work, check the most recent log in `/data/pvw/logs` for errors

```
function startParaViewWeb() {
  sudo service apache2 restart
  sudo runuser -l pvw-user -c 'cd /data/pv/pv-current && ./bin/
      pvpython lib/paraview-5.2/site-packages/vtk/web/launcher.py /
      data/pvw/conf/launcher.json &'
}
function stopParaViewWeb() {
  sudo pkill pvpython
  sudo service apache2 stop
}
```

Visualizer GUI

- Let me start the machinery ...
- And now try opening `http://206.12.97.61/visualizer`, loading a dataset and playing with it
- Can hide the left panel entirely by clicking on the cyan Visualizer logo
- Main UI elements: toolbar at the top lets you show the pipeline browser, browse files, add elements (filters and objects), save screenshots and states, get dataset info
- To be able to load NetCDF, the backend PV server has to be compiled with NetCDF support, the PVW Visualizer server app needs to be launched with a proxy file `pvw-visualizer.py -proxies proxies.json` to define the reader based on the file extension
- VTK files load directly
- In this case noticeably slower than native client-server: no WebGL and no remote GPU


decoCube.vti
Clip1
Contour1

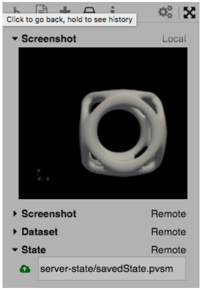
filter properties...

Origin: 14.5 14.5 14.5
Normal: 0 1 0
Offset: 0
Inside Out: ✓
Crinkle clip: ☐

Representation

Representation: Surface
Ambient Color: 1 1 1
Diffuse Color: 1 1 1
Opacity: 0.8
Specular: 0

Saving state


Click to go back, hold to see history
Screenshot Local
Screenshot Remote
Dataset Remote
State Remote
server-state/savedState.pvsm

(1) Can save current state from the save dialogue with the upload button

(2) Download the state file (along with the data file) to your laptop

```
scp ubuntu@206.12.97.61:/data/pvw/data/server-state/savedState.pvsm .
```

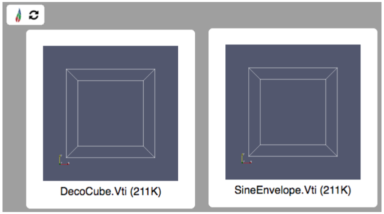
(3) Open local ParaView, $\boxed{\text{File}} \rightarrow \boxed{\text{Load State}}$, edit the location of the data file to recreate your visualization locally

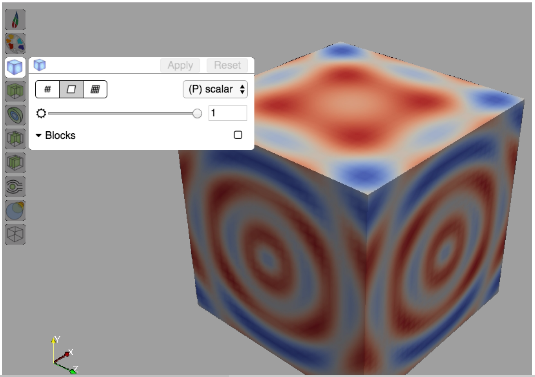
Few more Visualizer demos

- Threshold filter `decoCube.vti`
- Edit transfer function in volume rendering of a wavelet
- Clip `old/disk_out_ref.ex2`
- Animate `old/can.ex2`

Single- or multi-user LightViz on a cloud VM

- Install LightViz from its own repository (if compiling PV from source)
- Need to add datasets to LightViz before launching it $\Rightarrow$ it formats files in a specific way
- Otherwise similar to Visualizer setup
- Following modules currently available: dataset, clip, contour, slice, multi-slice, streamline, volume, threshold





Writing your own ParaViewWeb apps

- Is PVW right for you?
  - is your goal remote scientific visualization?
  - do you want to simply share 3D models online? (see the remainder of this webinar)

- It is JavaScript $\Rightarrow$ steep learning curve if you've never coded in it
- Can browse the API at `http://kitware.github.io/paraviewweb/api` and the PVW source code `https://github.com/Kitware/paraviewweb`
- Live code examples running in your browser at `http://kitware.github.io/paraviewweb/examples`
- Can play with Visualizer and LightViz source

1. Let me know the application/functionality you have in mind, or
2. Talk directly to Kitware `https://www.kitware.com`, they'll be happy to develop apps for you (and please keep me in the loop)

What if you don't need to run a complex renderer but simply want to put an existing 3D model or scene online for anyone to explore?

- Not so much scientific visualization but rather presenting 3D models online
  - e.g., historical artifacts, digital prototypes, 3D buildings or terrains, point cloud (lidar) maps
  - could be also used for presenting scientific visualizations (pre-rendered scenes)
- Would like a visitor to your page to be able to spin the object, zoom in/out, perhaps click on some predefined hotspots to define additional actions
- For an example, see `http://3d.si.edu/explorer?modelid=392` via the Smithsonian X 3D Explorer (a WebGL application talking to a proprietary server)

ParaViewWeb can do all of this but there are simpler solutions

Export scene as WebGL

Here is the simplest (but not the best) solution

(1) In your local ParaView build a "deco cube" isosurface at $f(x,y,z) = 0.03$

(2) $\boxed{\text{File}} \rightarrow \boxed{\text{Export Scene}}$ as WebGL to `decoCube.webgl` $\Rightarrow$ this will produce a number of files with the same base name and different extensions including `decoCube.html` and `decoCube.webgl`

(3) Copy these files to the VM into a separate directory, cd there and serve files from the current directory on the default HTTP listening port 80

```
cd decoCube
sudo python -m SimpleHTTPServer 80    # Python 2
sudo python -m http.server 80         # Python 3
```

(4) Point your web browser at `http://206.12.97.61/decoCube.html`

This will not give you zoom, hotspots, other advanced features

- no server $\Rightarrow$ will put quite a load on the viewer's WebGL client
- on the plus side can be cached by the browser

Use vtk.js to call WebGL inside your browser

Thanks to Sebastien Jourdain for pointing out this option!

- Vtk.js is a JavaScript library for client-side scientific visualization in a browser built on top of WebGL
  - open-source project from Kitware, Inc.
  - `https://kitware.github.io/vtk-js/docs`
  - can think of it as intermediate layer between WebGL and ParaViewWeb
- Leverages many of VTK functions and classes, entirely client-driven
- Example of a scene viewer that loads a local (to viewer) file and renders it with vtk.js at `https://kitware.github.io/vtk-js/examples/StandaloneSceneLoader.html`
  (1) Use their script `export-scene-macro.py` in ParaView to output any scene as a VTKJS data file
  (2) Load this file into the viewer using the "Choose File" button
  (3) Can easily incorporate the same html into your website or serve it on port 80

Other live vtk.js+WebGL geometry rendering examples from Kitware:
`https://kitware.github.io/vtk-js/examples/Execution.html`
`http://bit.ly/2nh1FSq` `http://bit.ly/2oez40I`

3DHOP = 3D Heritage Online Presenter

- Open-source package for presenting 3D high-resolution models online `http://3dhop.net`, written in HTML and JavaScript
  - open-source package from the Visual Computing Lab of the Istituto di Scienza e Tecnologie dell'Informazione
- Well-documented `http://3dhop.net/howto.php`
  - `http://3dhop.net/download/3DHOPsite_preparation.pdf`
  - `http://3dhop.net/download/3DHOPsite_deployment.pdf`
- Can handle the following file formats:
  1. single-resolution PLY (polygon file format) under 1MB
     - ☞ use 3D unstructured triangular mesh editor MeshLab `http://meshlab.sourceforge.net` to convert to PLY
     - per-vertex colour is supported
     - texture at the moment is not supported
     - vertex normals have to be included in the file
  2. NXS (batched multi-resolution mesh format) with $10^6 - 10^8$ triangles
     - ☞ first, use MeshLab to convert to PLY
     - ☞ then Nexus package `http://vcg.isti.cnr.it/nexus` to convert to NXS
  3. point clouds with $10^6 - 10^8$ points

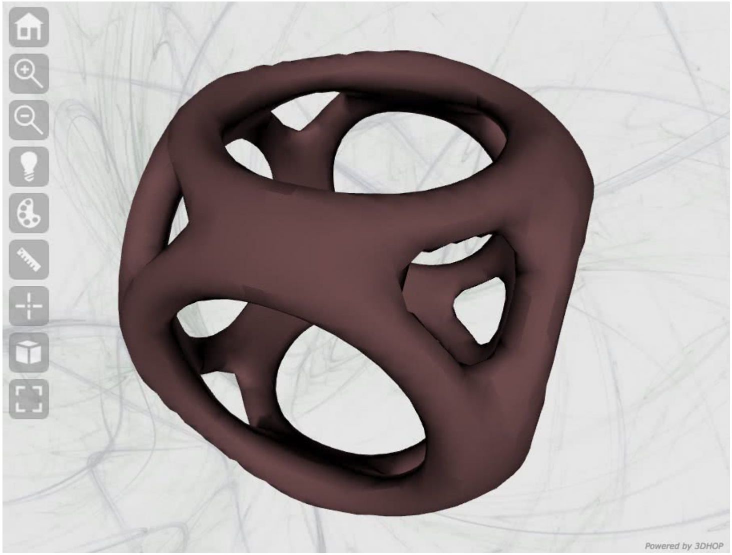
Import 3D continuous function into 3DHOP

- In your local ParaView build a "deco cube" isosurface at $f(x,y,z) = 0.03$
- In ParaView you can $\boxed{\text{File}} \rightarrow \boxed{\text{Save Data}}$ as PLY, but it does not write vertex colours (nice to have) or vertex normals (very nice to have) – instead, we'll do $\boxed{\text{File}} \rightarrow \boxed{\text{Export Scene}}$ to `decoCube.x3d`
- Open this scene in MeshLab $\boxed{\text{File}} \rightarrow \boxed{\text{ImportMesh}}$ and export it $\boxed{\text{File}} \rightarrow \boxed{\text{ExportMeshAs}}$ as `decoCube.ply` making sure to save vertex normals
- Upload `decoCube.ply` to the VM and then serve it with 3DHOP:

```
git clone https://github.com/cnr-isti-vclab/3DHOP.git
cd 3DHOP/minimal
mkdir -p models/singleres/
mv /path/to/decoCube.ply models/singleres/
cp index_all_tools.html index.html
sed -i -e 's|models/gargo.nxs|models/singleres/decoCube.ply|' index.html
sudo python -m SimpleHTTPServer 80   # Python 2
sudo python -m http.server 80        # Python 3
```

- Point your web browser at `http://206.12.97.61`
- You can find `index.html` in `config/3dhop/` inside the ZIP download

Import 3D continuous function into 3DHOP



Creating interactive hotspots in a 3DHOP scene

- Described in `http://3dhop.net/examples.php?id=7`
- I created a couple of *hotspot* meshes `ring.ply` and `edge.ply` (loaded the original model into ParaView, used the Clip filter, exported the scene as X3D file making sure the Clip's plane is not visible, converted this scene to PLY file with MeshLab)
- Using index.html as template, I created a new file hotSpots.html in which we
  1. define mesh1, mesh2, mesh3,
  2. set up ringSpot and edgeSpot,
  3. define Hide/Show Hotspots buttons and add them to actionsToolbar(),
  4. define actions in onPickedSpot()
- Copy the hotspot meshes and a *modifed html* into the local directory:

```
cp /path/to/{ring,edge}.ply ~/3DHOP/minimal/models/singleres
cp /path/to/indexHotSpots.html ~/3DHOP/minimal
```

- Point your web browser at `http://206.12.97.61/hotSpots.html`
  - now there is a button "Show/Hide Hotspots"
  - clicking on the edge hotspot opens WestGrid's homepage in a new window
  - clicking on the ring hotspot opens an alert window

Questions?

- Email me at alex.razoumov@westgrid.ca
- Submit a problem ticket, e.g., support@westgrid.ca
- `https://docs.computecanada.ca/wiki/Visualization`
- Compute Canada visualization support viz-support@computecanada.ca
- Compute Canada visualization gallery `http://bit.ly/cctopviz`