3D visualization for the humanities

Alex Razoumov
alex.razoumov@westdri.ca

Digital Research Alliance of Canada

SFU SIMON FRASER UNIVERSITY

(1) copy of these slides and other files at `https://bit.ly/dhslides`

- will download dh.zip (~ 12 MB)
- uncompress it into your Downloads folder
- find the slides `dhsi.pdf` inside

(2) software installation described in slide 11

Bit of background

- Background in computational astrophysics
  - numerical simulations in galaxy formation, core-collapse supernovae, accretion disks, stellar hydrodynamics
  - designing numerical methods in computational fluid dynamics and radiative transfer
  - lots of parallel programming
- Day job: scientific visualization and teaching research computing across four Western provinces
  - 3D sci-vis of large computational models `https://ccvis.netlify.app`
  - basic and parallel programming: bash, Python, Julia, Chapel, C/C++, Fortran
- So ... teaching a 3D visualization workshop at DHSI
- My approach: apply scientific visualization tools to DH data
  - think of it as extension of interactive 2D plotting into the 3rd dimension using general-purpose, open-source, highly scalable tools
  - research scenario: already have a 3D dataset and want to visualize it
  - would love to hear back about humanities and social sciences problems for which we can apply today's tools

What we are not covering today

- Virtual tours, museums, reconstructions
  - game engines such as Unreal Engine, Unity, Godot (all open-source now)
  - 3D animation with Maya or Houdini (although these can be used for sci-vis)
  - special virtual environments such as Vsim (3D learning env. for DH), OpenSimulator (multi-user online env.)
  - don't confuse these with viewing visualizations with VR/AR headsets, which can be used for looking at pretty much everything we'll do today
- Architectural renderings
- 3D printing, modeling tools for design and prototyping – covered in another DHSI course?
- Photogrammetric processing of images – covered in DHSI course #52
  1. building 3D models from a set of images taken from various directions
     - many commercial offerings, rich area for VR and AR
     - for open-source `http://www.regard3d.org` ... good topic for a PhD thesis?
  2. building polynomial texture maps from a set of images taken with varying lighting direction
- Visualization of point cloud data – but we can do this with today's tools
- Artistic text visualizations

What we are covering today

- 3D multi-attribute scatter plots
  - semantic text analysis with multidimensional scaling to reduce distances to a 3D map
  - country ratings data from the Legatum 2015 Prosperity Index
- 3D graphs
  - NetworkX built-in graphs and layouts
  - custom layouts: encoding attibute(s) in the third dimension
  - scripting selections
  - graph statistics
- Continuous distributions
  - 2D function f(x,y) extended into the third dimension
  - 3D function f(x,y,z)
  - using 3D filters to analyze data
- Creating animations
- Putting 3D visualizations on the web
  - briefly on the Smithsonian collection
  - vtk.js library on top of WebGL for client-side visualization
  - ParaView Glance
  - `http://3dhop.net`, an open-source software package for presenting interactive high-resolution 3D models online, aimed at the Cultural Heritage field

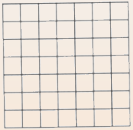
General-purpose 3D visualization tools

What is VTK?

- 3D **Visualization Toolkit** software system for 3D computer graphics, image processing, and visualization
- Open-source and cross-platform (Windows, Mac, Linux, other Unix variants)
- Suppors OpenGL hardware acceleration
- Originally a C++ class library, now with interpreted interface layers for Python, Java, Tcl/Tk, JavaScript
- Supports wide variety of visualization and processing algorithms for polygon rendering, ray tracing, mesh smoothing, cutting, contouring, Delaunay triangulation, etc.
- Supports many data types: scalar, vector, tensor, texture, arrays of arrays
- Supports many 2D/3D spatial discretizations: structured and unstructured meshes, particles, polygons, etc. – see next slide
- Includes a suite of 3D interaction widgets, integrates nicely with several popular cross-platform GUI toolkits (Qt, Tk)
- Supports parallel processing and parallel I/O
- Base layer of many excellent 3D visualization packages (ParaView, VisIt, MayaVi, and several others)

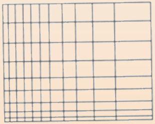
VTK 2D/3D data: 6 major discretizations (mesh types)

- **Image Data/Structured Points**: *.vti, points on a regular rectangular lattice, scalars or vectors at each point



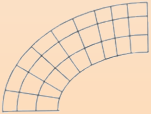
(a) Image Data

- **Rectilinear Grid**: *.vtr, same as Image Data, but spacing between points may vary, need to provide steps along the coordinate axes, not coordinates of each point



(b) Rectilinear Grid

- **Structured Grid**: *.vts, regular topology and irregular geometry, need to indicate coordinates of each point



(c) Structured Grid

VTK 2D/3D data: 6 major discretizations (mesh types)

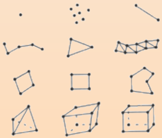
- **Particles/Unstructured Points**: *.particles

(d) Unstructured Points

- **Polygonal Data**: *.vtp, unstructured topology and geometry, point coordinates, 2D cells only (i.e. no polyhedra), suited for maps

(e) Polygonal Data

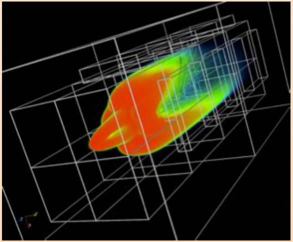
- **Unstructured Grid**: *.vtu, irregular in both topology and geometry, point coordinates, 2D/3D cells, suited for finite element analysis, structural design

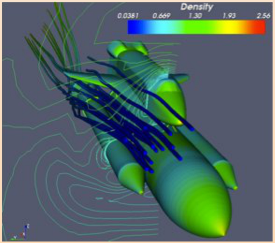


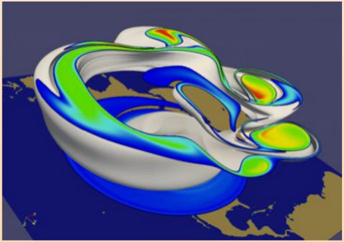
(f) Unstructured Grid

ParaView as GUI frontend to VTK classes

- 3D visualization tool for extremely large datasets
- From laptops to supercomputers with hundreds of thousands of cores
- Open source, pre-compiled downloads for Linux/Mac/Windows from `http://www.paraview.org`
- Interactive GUI and Python scripting
- Uses MPI for distributed-memory parallelism on HPC clusters
- Client-server architecture
- Developed by VTK authors, fully supports all VTK classes and data types
- Huge array of visualization features

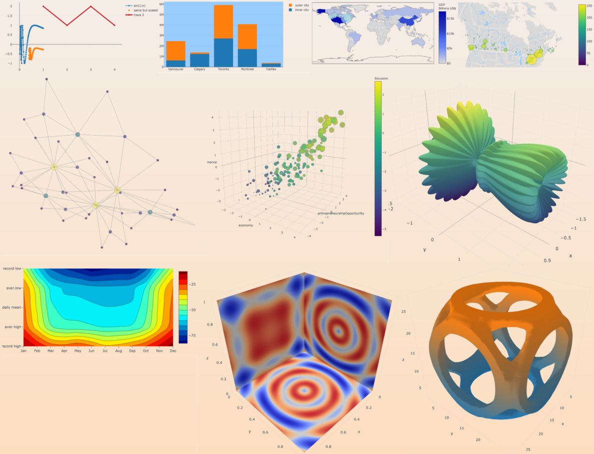






Alternative tool: Plotly Python library

- Open-source project from Plot.ly `https://plot.ly/python`
- Produces dynamic html5 visualizations for the web
- APIs for Python (with/without Jupyter), R, JavaScript, MATLAB



- Can work offline (free) or by sending your data to your account on plot.ly (public plotting is free, paid unlimited private plotting and extra tools)

Software installation

- Today's only software requirement: `https://www.paraview.org/download`
  - ParaView comes with its own Python shell and VTK, but it is somewhat tricky to install libraries there
- To run Python scripts today, we will use `https://jupyter.dhsi.c3.ca`
  - now I will distribute the usernames and passwords
  - option 1: run a Python 3 notebook (not today: would need to compile VTK from source for 3.11)
  - option 2: run Python 3.10 scripts from the command line, all libraries already installed there
- Optional local installation:
  1. for your OS install Python 3.10 or 3.11 ☜ there are many ways, e.g. via Miniconda `http://conda.pydata.org/miniconda.html`, or a binary Python installer, or (MacOS only) `https://brew.sh`

```
$ brew install python   # installs into /opt/homebrew/bin/python3
$ export PATH="$(brew␣--prefix)/opt/python@3/libexec/bin:$PATH"   # add this to your ~/.bashrc
```

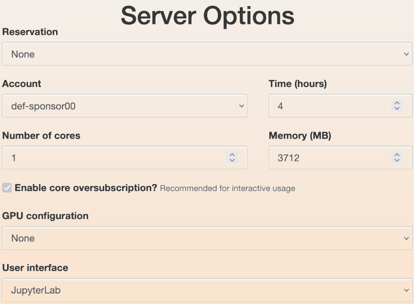
  2. start the command shell (Terminal in MacOS/Linux, Anaconda or some other Prompt in Windows) and then install the required Python packages:

```
$ pip install virtualenv
$ virtualenv --no-download dhsi-env  # install Python tools in the current directory
$ source dhsi-env/bin/activate       # (every time) load the environment
$ pip install --upgrade pip
$ pip install pip_search             # optional
$ pip_search vtk
$ pip install numpy networkx vtk gensim scikit-learn plotly pandas
...
$ deactivate                         # (every time) unload the environment
```

  3. start Python and test your Miniconda installation:

```
>>> import vtk, networkx as nx
```

Download data to the remote machine



1. Open `https://jupyter.dhsi.c3.ca` in your browser, log in with your unique username and password, leave the OTP field blank
2. Start a server using the settings on the right
3. Launch a terminal by clicking on a button, also can do this via File | New Launcher | Terminal
4. Download and unpack the scripts and data

```
mkdir ~/tmp
cd ~/tmp
wget --no-check-certificate http://bit.ly/dhslides -O dhfiles.zip
# cp /project/def-sponsor00/shared/dhfiles.zip .    # if wget does not work
unzip dhfiles.zip
rm dhfiles.zip
ls
```

5. Load our Python environment

```
source /project/def-sponsor00/shared/dhsi-env/bin/activate
```

Python function to write points and graphs as VTK

- Function `writeObjects()` in `writeNodesEdges.py`
- Stores points or graphs as vtkPolyData or vtkUnstructuredGrid

```
def writeObjects(nodeCoords,
                 edges = [],
                 scalar = [], name = '', power = 1,
                 scalar2 = [], name2 = '', power2 = 1,
                 nodeLabel = [],
                 method = 'vtkPolyData',
                 fileout = 'test'):
    """
    Store points and/or graphs as vtkPolyData or vtkUnstructuredGrid.
    Required argument:
    - nodeCoords is a list of node coordinates in the format [x,y,z]
    Optional arguments:
    - edges is a list of edges in the format [nodeID1,nodeID2]
    - scalar/scalar2 is the list of scalars for each node
    - name/name2 is the scalar's name
    - power/power2 = 1 for r~scalars, 0.333 for V~scalars
    - nodeLabel is a list of node labels
    - method = 'vtkPolyData' or 'vtkUnstructuredGrid'
    - fileout is the output file name (will be given .vtp or .vtu extension)
    """
```

Making 3D scatter plots

Semantic mapping

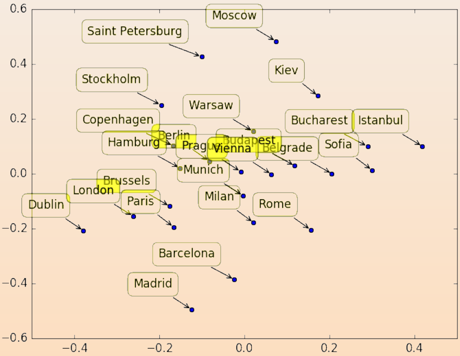
Idea inspired by this blog post from 2009

- Analyzed a corpus of 5,733,721 articles from 2,231 research journals (mostly science, technology and medical fields)
- Mapped the position of each journal in the 512-dimensional "semantic space" (more on this later)
- Calculated a 2231×2231 distance matrix in 512D
- Used multidimensional scaling to convert this matrix to 2D positions of 2231 points

- Coloured the points by 23 human-created journal categories
- Found excellent correspondence with human-created journal categories

Multidimensional scaling

Challenge: given a 24×24 table of pairwise distances between 24 cities, reconstruct their relative positions in 2D.



Semantic analysis of five public-domain texts

(1) THE TIME MACHINE, by Herbert Wells

(2) OLIVER TWIST, by Charles Dickens

(3) ADVENTURES OF HUCKLEBERRY FINN, by Mark Twain

(4) THE WAR OF THE WORLDS, by Herbert Wells

(5) GALILIEAN-INVARIANT COSMOLOGICAL HYDRODYNAMICAL SIMULATIONS ON A MOVING MESH, by Volker Springel

- We'll analyze dictionaries and relative word frequencies and visualize a distance-based map of these texts in 3D

Algorithm

(1) From each text pick up 30 longest paragraphs, $\times 5$ texts $\Rightarrow$ 150 paragraphs

(2) Convert line breaks and dashes to spaces, remove punctuation

(3) Remove common words (prepositions, articles, etc)

(4) Count words across all paragraphs and remove words that appear only once across all texts

(5) Build a global dictionary (one for all five texts) of words, with N_{words} words

(6) Vectorize each paragraph in the N_{words}-dimensional space, positioning it according to its word count; for details see `http://radimrehurek.com/gensim/tut1.html`

(7) Normalize each vector to the number of words in its paragraph, to count relative word frequencies

(8) Calculate pairwise distances between all paragraphs in the N_{words}-dimensional space $\Rightarrow$ 150×150 matrix of numbers

(9) Use multidimensional scaling to convert the distance matrix to paragraph positions in 3D, store them as VTK points

(10) Visualize these points in 3D with ParaView, colouring by the author and sizing by the text per author (two texts for Herbert Wells)

Algorithm

(1) From each text pick up 30 longest paragraphs, $\times 5$ texts $\Rightarrow$ 150 paragraphs

(2) Convert line breaks and dashes to spaces, remove punctuation

(3) Remove common words (prepositions, articles, etc)

(4) Count words across all paragraphs and remove words that appear only once across all texts

(5) Build a global dictionary (one for all five texts) of words, with N_{words} words

(6) Vectorize each paragraph in the N_{words}-dimensional space, positioning it according to its word count; for details see `http://radimrehurek.com/gensim/tut1.html`

(7) Normalize each vector to the number of words in its paragraph, to count relative word frequencies

(8) Calculate pairwise distances between all paragraphs in the N_{words}-dimensional space $\Rightarrow$ 150×150 matrix of numbers

(9) Use multidimensional scaling to convert the distance matrix to paragraph positions in 3D, store them as VTK points

(10) Visualize these points in 3D with ParaView, colouring by the author and sizing by the text per author (two texts for Herbert Wells)

Algorithm

(1) From each text pick up 30 longest paragraphs, $\times 5$ texts $\Rightarrow$ 150 paragraphs

(2) Convert line breaks and dashes to spaces, remove punctuation

(3) Remove common words (prepositions, articles, etc)

(4) Count words across all paragraphs and remove words that appear only once across all texts

(5) Build a global dictionary (one for all five texts) of words, with N_{words} words

(6) Vectorize each paragraph in the N_{words}-dimensional space, positioning it according to its word count; for details see `http://radimrehurek.com/gensim/tut1.html`

(7) Normalize each vector to the number of words in its paragraph, to count relative word frequencies

(8) Calculate pairwise distances between all paragraphs in the N_{words}-dimensional space $\Rightarrow$ 150×150 matrix of numbers

(9) Use multidimensional scaling to convert the distance matrix to paragraph positions in 3D, store them as VTK points

(10) Visualize these points in 3D with ParaView, colouring by the author and sizing by the text per author (two texts for Herbert Wells)

Algorithm

(1) From each text pick up 30 longest paragraphs, $\times 5$ texts $\Rightarrow$ 150 paragraphs

(2) Convert line breaks and dashes to spaces, remove punctuation

(3) Remove common words (prepositions, articles, etc)

(4) Count words across all paragraphs and remove words that appear only once across all texts

(5) Build a global dictionary (one for all five texts) of words, with N_{words} words

(6) Vectorize each paragraph in the N_{words}-dimensional space, positioning it according to its word count; for details see `http://radimrehurek.com/gensim/tut1.html`

(7) Normalize each vector to the number of words in its paragraph, to count relative word frequencies

(8) Calculate pairwise distances between all paragraphs in the N_{words}-dimensional space $\Rightarrow$ 150×150 matrix of numbers

(9) Use multidimensional scaling to convert the distance matrix to paragraph positions in 3D, store them as VTK points

(10) Visualize these points in 3D with ParaView, colouring by the author and sizing by the text per author (two texts for Herbert Wells)

Algorithm

(1) From each text pick up 30 longest paragraphs, $\times 5$ texts $\Rightarrow$ 150 paragraphs

(2) Convert line breaks and dashes to spaces, remove punctuation

(3) Remove common words (prepositions, articles, etc)

(4) Count words across all paragraphs and remove words that appear only once across all texts

(5) Build a global dictionary (one for all five texts) of words, with N_{words} words

(6) Vectorize each paragraph in the N_{words}-dimensional space, positioning it according to its word count; for details see `http://radimrehurek.com/gensim/tut1.html`

(7) Normalize each vector to the number of words in its paragraph, to count relative word frequencies

(8) Calculate pairwise distances between all paragraphs in the N_{words}-dimensional space $\Rightarrow$ 150×150 matrix of numbers

(9) Use multidimensional scaling to convert the distance matrix to paragraph positions in 3D, store them as VTK points

(10) Visualize these points in 3D with ParaView, colouring by the author and sizing by the text per author (two texts for Herbert Wells)

Algorithm

(1) From each text pick up 30 longest paragraphs, $\times 5$ texts $\Rightarrow$ 150 paragraphs

(2) Convert line breaks and dashes to spaces, remove punctuation

(3) Remove common words (prepositions, articles, etc)

(4) Count words across all paragraphs and remove words that appear only once across all texts

(5) Build a global dictionary (one for all five texts) of words, with N_{words} words

(6) Vectorize each paragraph in the N_{words}-dimensional space, positioning it according to its word count; for details see `http://radimrehurek.com/gensim/tut1.html`

(7) Normalize each vector to the number of words in its paragraph, to count relative word frequencies

(8) Calculate pairwise distances between all paragraphs in the N_{words}-dimensional space $\Rightarrow$ 150×150 matrix of numbers

(9) Use multidimensional scaling to convert the distance matrix to paragraph positions in 3D, store them as VTK points

(10) Visualize these points in 3D with ParaView, colouring by the author and sizing by the text per author (two texts for Herbert Wells)

Algorithm

(1) From each text pick up 30 longest paragraphs, $\times 5$ texts $\Rightarrow$ 150 paragraphs

(2) Convert line breaks and dashes to spaces, remove punctuation

(3) Remove common words (prepositions, articles, etc)

(4) Count words across all paragraphs and remove words that appear only once across all texts

(5) Build a global dictionary (one for all five texts) of words, with N_{words} words

(6) Vectorize each paragraph in the N_{words}-dimensional space, positioning it according to its word count; for details see `http://radimrehurek.com/gensim/tut1.html`

(7) Normalize each vector to the number of words in its paragraph, to count relative word frequencies

(8) Calculate pairwise distances between all paragraphs in the N_{words}-dimensional space $\Rightarrow$ 150×150 matrix of numbers

(9) Use multidimensional scaling to convert the distance matrix to paragraph positions in 3D, store them as VTK points

(10) Visualize these points in 3D with ParaView, colouring by the author and sizing by the text per author (two texts for Herbert Wells)

Algorithm

(1) From each text pick up 30 longest paragraphs, $\times 5$ texts $\Rightarrow$ 150 paragraphs

(2) Convert line breaks and dashes to spaces, remove punctuation

(3) Remove common words (prepositions, articles, etc)

(4) Count words across all paragraphs and remove words that appear only once across all texts

(5) Build a global dictionary (one for all five texts) of words, with N_{words} words

(6) Vectorize each paragraph in the N_{words}-dimensional space, positioning it according to its word count; for details see `http://radimrehurek.com/gensim/tut1.html`

(7) Normalize each vector to the number of words in its paragraph, to count relative word frequencies

(8) Calculate pairwise distances between all paragraphs in the N_{words}-dimensional space $\Rightarrow$ 150×150 matrix of numbers

(9) Use multidimensional scaling to convert the distance matrix to paragraph positions in 3D, store them as VTK points

(10) Visualize these points in 3D with ParaView, colouring by the author and sizing by the text per author (two texts for Herbert Wells)

Hands-on: running the script

1. The entire algorithm is implemented in `semanticMapping.py`
   - ☞ let's take a look at it
   - if working inside a Jupyter notebook, load the code into the current cell with:

```
%load semanticMapping.py   # fills the current cell with code from the script
```

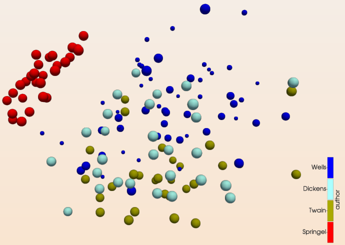
     and then run it
   - if working in the terminal, use the commands:

```
pwd      # should be in /home/userXX/tmp
cat semanticMapping.py
python semanticMapping.py
```

2. Locate and download `texts.vtu` to your computer

Hands-on: viewing results in ParaView

- Load `texts.vtu` into ParaView and follow my instructions
  - colour glyphs by "author"
  - switch from continuous to categorical colours and annotate them, e.g.
    - blue, author=1, Herbert Wells
    - pale blue, author=2, Charles Dickens
    - beige, author=3, Mark Twain
    - red, author=4, Volker Springel
  - size glyphs by "novel per author" (large: The Time Machine, small: The War of the Worlds)



- Save the state to file `fourAuthors.pvsm`
- On Unix-like systems can reload from the GUI or from the command line with something like

```
/path/to/paraview --state=fourAuthors.pvsm
```

- Alternatively could map to 2D, using the third dimension to visualize another attribute, e.g. the year of publication, or the text size, or the number of protagonists, etc.

Hands-on: viewing results with plotly

```
$ diff semanticMapping.py directMapping.py
< print(coords)
98,99c98,116
< writeObjects(coords, scalar=author, name='author', fileout='texts',
<              scalar2=novelPerAuthor, name2='novel␣per␣author', method = 'vtkUnstructuredGrid')
---
> x = [point[0] for point in coords]
> y = [point[1] for point in coords]
> z = [point[2] for point in coords]
>
> import plotly.offline as py
> py.init_notebook_mode(connected=True)
> import plotly.graph_objs as go
> spheres = go.Scatter3d(x=x, y=y, z=z, mode='markers',
>                        marker=dict(
>                            sizemode = 'diameter', sizeref = 0.2, size = novelPerAuthor,
>                            color = author, colorscale = 'Viridis',
>                            colorbar = dict(title = 'author'),
>                            line = dict(color='rgb(140,␣140,␣170)')))    # sphere edge
> layout = go.Layout(title='Each␣sphere␣is␣a␣paragraph␣coloured␣by␣author'+
>                    '␣and␣sized␣by␣novelPerAuthor')
> fig = go.Figure(data=[spheres], layout=layout)
> py.iplot(fig)
```

- There are two code versions: (1) for running/rendering inside a Jupyter notebook, and (2) for running in a terminal and saving into an HTML file
  - ☞ make sure to run the correct one

Speculative semantic analysis of the four gospels ▷

- `gospels.py` is a copy of `semanticMapping.py` doing the same analysis on the four gospels (Matthew, Mark, Luke and John) in Greek
- Run it inside Jupyter
- Download `testament.vtu` and load it into ParaView
- Continuous colouring with the default colour map shows **Matthew in blue**, **Mark in pale blue**, **Luke in beige**, and **John in red**
- Switch to categorical colouring, assign similar colours and annotate them

Speculative semantic analysis of the four gospels ▷▷

- John (red) is the most independent
- Luke (beige) ← Matthew (blue) + Mark (pale blue)
  - Luke has a lot of overlap with Matthew and Mark, so likely a composition from both Matthew and Mark
  - not the other way around (Matthew or Mark being a composition from Luke), as Matthew and Mark are sufficiently different
- Drop Luke (author=3) from the analysis: apply a Threshold filter 2.5 – 3.5 acting on the output of the Glyph filter, and invert

Speculative semantic analysis of the four gospels ▷▷▷

- Matthew ← John + Mark
  - Matthew may have pulled a bit from John and Mark (sitting in the middle between the two)
  - this leaves us with John and Mark as primaries
- Drop Matthew (author=1) from the analysis: add a second Threshold filter 1.5 – 4

Speculative semantic analysis of the four gospels ▷▷▷▷

- Mark and John have good separation, but there is an open region between them left by Matthew and Luke
- Supports (does not rule out?) the idea that there could have been another (now lost) primary that would have filled this region that both Matthew and Luke pulled from
- This leaves us with John, Mark, and a lost source as our primaries
- Couple of extreme outliers: written by entirely different authors, possibly in a different time period?

Exercises: pick the one you like

(1) Apply `semanticMapping.py` to several other texts, visualize the results, and do your own analysis. Do the results make sense?

- need to be in plain text (any language), not in a proprietary binary format
- could be your own texts

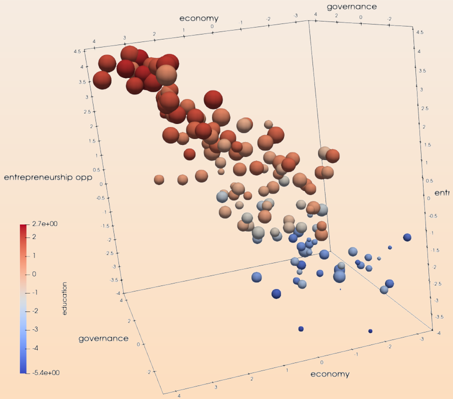
(2) Combine the five texts in English and the four gospels in Greek into a single analysis

(3) More challenging: use multidimensional scaling to reduce distances to 2D, and then plot the paragraphs in 3D using the third dimension to visualize some interesting attribute

(4) Repeat the "four gospels" visualization in Plotly

Prosperity index: 3D scatter plot, 5 attributes

- Data from the Legatum 2015 Prosperity Index `http://www.prosperity.com/#!/ranking` (click on Scores, best to copy/paste from Firefox)
- Take a look at the data in `legatum2015.csv`: 8 rankings for each country
- Run the code `countries.py` (writes five attributes into `countries.vtp`)
- Apply Glyph filter with Glyph Mode = All Points
- 3D position by (economy, entrepreneurshipOpportunity, governance)
- Colour by education
- Size by safetySecurity
- Save the state to `countries.pvsm`
- Optionally turn on labels for countries – see the next slide



Labeling nodes

(1) Press V to bring up Find Data dialogue

(2) Find=Points from=countries.vtp with ID ≥ 0 (will labels all points) and press Find Data

(3) Make sure countries.vtp is visible in the pipeline browser

(4) Check Point Labels -> tag to display the label (and not another variable)

(5) Click on the gear icon (Edit Label Properties) and set opacity=0 and adjust the Point Label Font size

(6) Now try labeling a single country (unfortunately, "`tag is countryName`" syntax does not work), but can look up the country in `legatum2015.csv`, check the line number, subtract 2, and use that as ID

(7) Now label all those with poor security: safetySecurity ≤ 3

(8) Now label all those with good education: education ≥ 1.5

Labeling nodes

(1) Press V to bring up Find Data dialogue

(2) Find=Points from=countries.vtp with ID ≥ 0 (will labels all points) and press Find Data

(3) Make sure countries.vtp is visible in the pipeline browser

(4) Check Point Labels -> tag to display the label (and not another variable)

(5) Click on the gear icon (Edit Label Properties) and set opacity=0 and adjust the Point Label Font size

(6) Now try labeling a single country (unfortunately, "`tag is countryName`" syntax does not work), but can look up the country in `legatum2015.csv`, check the line number, subtract 2, and use that as ID

(7) Now label all those with poor security: safetySecurity ≤ 3

(8) Now label all those with good education: education ≥ 1.5

Labeling nodes

(1) Press V to bring up Find Data dialogue

(2) Find=Points from=countries.vtp with ID ≥ 0 (will labels all points) and press Find Data

(3) Make sure countries.vtp is visible in the pipeline browser

(4) Check Point Labels -> tag to display the label (and not another variable)

(5) Click on the gear icon (Edit Label Properties) and set opacity=0 and adjust the Point Label Font size

(6) Now try labeling a single country (unfortunately, "`tag is countryName`" syntax does not work), but can look up the country in `legatum2015.csv`, check the line number, subtract 2, and use that as ID

(7) Now label all those with poor security: safetySecurity ≤ 3

(8) Now label all those with good education: education ≥ 1.5

Labeling nodes

(1) Press V to bring up Find Data dialogue

(2) Find=Points from=countries.vtp with ID ≥ 0 (will labels all points) and press Find Data

(3) Make sure countries.vtp is visible in the pipeline browser

(4) Check Point Labels -> tag to display the label (and not another variable)

(5) Click on the gear icon (Edit Label Properties) and set opacity=0 and adjust the Point Label Font size

(6) Now try labeling a single country (unfortunately, "`tag is countryName`" syntax does not work), but can look up the country in `legatum2015.csv`, check the line number, subtract 2, and use that as ID

(7) Now label all those with poor security: safetySecurity ≤ 3

(8) Now label all those with good education: education ≥ 1.5

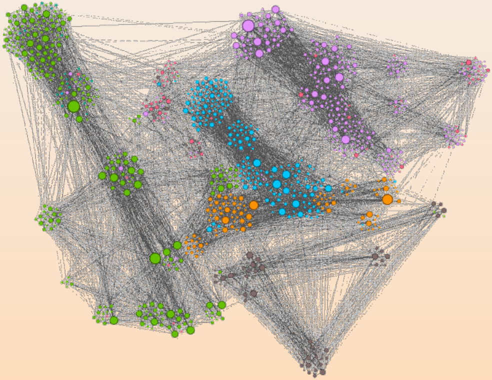
Prosperity index in Plotly (prosperity.py)

```
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import pandas as pd
df = pd.read_csv('legatum2015.csv')
spheres = go.Scatter3d(x=df.economy,
                       y=df.entrepreneurshipOpportunity,
                       z=df.governance,
                       text=df.country,
                       mode='markers',
                       marker=dict(
                           sizemode = 'diameter',
                           sizeref = 0.3,
                           size = df.safetySecurity+5.5,
                           color = df.education,
                           colorscale = 'Viridis',
                           colorbar = dict(title = 'Education'),
                           line = dict(color='rgb(140, 140, 170)'))) # sphere edge
layout = go.Layout(title='Each sphere is a country sized by safetySecurity',
                   scene = dict(xaxis=dict(title='economy'),
                                yaxis=dict(title='entrepreneurshipOpportunity'),
                                zaxis=dict(title='governance')))
fig = go.Figure(data=[spheres], layout=layout)
py.iplot(fig)
```

Visualizing 3D graphs

Dedicated 2D graph tools

- Many dedicated 2D tools, most popular ones are Gephi, Cytoscape (both open source)



- How can we extend this to 3D? And do we really want to?

Dedicated 3D graph tools (circa 2016)

- Force Atlas 3D plugin for Gephi `http://bit.ly/1QcLuLK` gives a 2D projection with nodes as spheres at (x,y,z) and the proper perspective and lighting, but can't interact with the graph in 3D
- Functional brain network visualization tools, e.g. Connectome Viewer `http://cmtk.org/viewer`
- GraphInsight was a fantastic tool, free academic license, embedded Python shell – went to the dark side in the fall 2013 (purchased by a bank, no longer exists, can still find demo versions and youtube videos) http://www.graphinsight.com is down ... https://twitter.com/GraphInsight
- Walrus `http://www.caida.org/tools/visualization/walrus` was a research project, latest update in 2005, old source still available but people seem to have trouble compiling and running it now
- Network3D from Microsoft seems to be a short-lived research project, Windows only
- BioLayout Express 3D `http://www.biolayout.org/download` is Ok, written in Java, development stopped in 2014 but still works, only the commercial tool maintained ($500)
- ORA NetScenes from Carnegie Mellon for "networked text visualization", not bad, Windows only, not open-source, licensing not clear (more of a demo license, they reserve the right to make it paid)
- Number of other research projects, in my view not targeting end users, e.g. `http://www.opengraphiti.com` (pain to compile: tends to pick /usr/bin/python, only Mac/Linux), or WebGL projects `https://youtu.be/qHkjSxbnzAU` that really require programming knowledge
  - `https://markwolff.shinyapps.io/QMtriplot17C/` is a nice WebGL example in R + Shiny
  - 3D Force-Directed Graph web component `https://github.com/vasturiano/3d-force-graph` implemented with ThreeJS and WebGL for 3D rendering and d3-force-3d for the layout (force) engine, not bad overall, but very CPU/GPU-heavy on the client

Is there any good, open-source, cross-platform, currently maintained, user-friendly dedicated 3D graph visualization tool?

... or we could use a general-purpose visualization tool

NetworkX + VTK + ParaView

- Our first solution: NetworkX + VTK + ParaView
  - advantages: (1) using general-purpose visualization tool; (2) everything is scriptable; (3) can scale directly to $10^{\sim 5.5}$ nodes, with a little extra care to $10^{\sim 9.5}$ nodes
  - disadvantages: graphs are static 3D objects, can't click on a node, highlight connections, move nodes, etc. (but we can script all these interactions!)
  - note: in the current implementation edges are displayed as straight lines; possible to use vtkArcSource or vtkPolyLine to create arcs and store them as vtkPolyData

(1) We'll use NetworkX + VTK to create a graph, position nodes, optionally compute graph statistics, and write everything to a VTK file; we'll do this in Python 3.8

(2) Load that file into ParaView

- On presenter's laptop see *mutOnCtOrbits.mp4* for a more complex graph (6×10^5 edges) created with this workflow

NetworkX + Plotly

- Our second solution: NetworkX + Plotly
  - no intermediate steps: graph created directly in Python, opens automatically in a web browser
  - everything is scriptable
  - limited scaling
  - similarly to ParaView, no proper “graph controls” in 3D

NetworkX graphs

- NetworkX is a Python package for the creation, manipulation, and analysis of complex networks
- Documentation at `http://networkx.github.io`

```
import networkx as nx

# return all names (attributes and methods) inside nx
dir(nx)

# generate a list (of 139) built-in graph types
# with Python's ''list comprehension''
[x for x in dir(nx) if "_graph" in x]
```

NetworkX layouts ▷

```
# generate a (much shorter) list of built-in graph layouts
[x for x in dir(nx) if "_layout" in x]
# will print ['arf_layout', 'bipartite_layout', 'circular_layout',
#  'fruchterman_reingold_layout', 'kamada_kawai_layout', 'multipartite_layout',
#  'planar_layout', 'random_layout', 'rescale_layout', 'rescale_layout_dict',
#  'shell_layout', 'spectral_layout', 'spiral_layout', 'spring_layout']

# can always look at the help pages
help(nx.circular_layout)
```

- `spring_` and `fruchterman_reingold_` are the same, so really 13 built-in layouts
- can use 3rd-party layouts
- `circular_`, `random_`, `shell_` are fixed layouts
- `spring_` and `spectral_` are force-directed layouts: linked nodes attract each other, non-linked nodes are pushed apart

NetworkX layouts ▷▷

- Layouts typically return a dictionary, with each element being a 2D/3D coordinate array indexed by the node's number (or name)

```
# generate a random graph with 10 nodes and 50 edges
H = nx.gnm_random_graph(10,50)

# the layout is a dictionary of 2D coordinates of all 10 nodes
nx.shell_layout(H,dim=2)      # in this layout only dim=2 supported

# each value of these is an (x,y,z) coordinate of a node
nx.circular_layout(H,dim=3)
nx.spring_layout(H,dim=3)
nx.random_layout(H,dim=3)
nx.spectral_layout(H,dim=3)
```

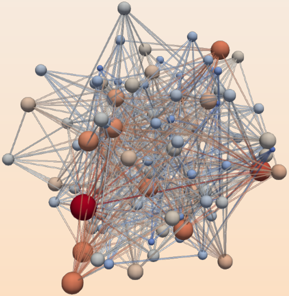
Our first graph (randomGraph.py)

```
import networkx as nx
from writeNodesEdges import writeObjects

numberNodes, numberEdges = 100, 500
H = nx.gnm_random_graph(numberNodes,numberEdges)
print('nodes:', H.nodes())
print('edges:', H.edges())

# return a dictionary of positions keyed by node
pos = nx.random_layout(H,dim=3)
# convert to list of positions (each is a list)
xyz = [list(pos[i]) for i in pos]

degree = [d for i,d in H.degree()]
writeObjects(xyz, edges=H.edges(), scalar=degree,
             name='degree', fileout='network')
```



Load this graph into ParaView

- After you run `randomGraph.py` from the command line, to reproduce the previous slide, you have three options:
  (1) load the file `network.vtp`, apply Glyph filter, apply Tube filter, edit their properties, or
  (2) in ParaView's menu navigate to File | Load State and select `drawGraph.pvsm`, or
    - important: adjust the data file location!

```
$ grep Users drawGraph.pvsm
<Element index="0" value="/Users/razoumov/teaching/humanities/network.vtp"/>
<Element index="0" value="/Users/razoumov/teaching/humanities/network.vtp"/>
```

  (3) on a Unix-based system start ParaView and load the state with one command:

```
/path/to/paraview --state=drawGraph.pvsm
```

- For subsequent plots, you can reload data without rebuilding the plot File | Reload Files

Labeling graph nodes

(1) Press V to bring up Find Data dialogue

(2) Find Points with ID>=0 (or other selection)

(3) Make points visible in the pipeline browser

(4) Check Point Labels -> ID (can also do this operation from View -> Selection Display Inspector)

(5) Adjust the label font size

(6) Set original data opacity to 0

Also we can label only few selected points, e.g. those with degree ≥ 10

Switch to spring layout

- Let's apply a force-directed layout

```
$ diff randomGraph.py randomGraph2.py
10c10,11
< pos = nx.random_layout(H,dim=3)
---
> pos = nx.spring_layout(H,dim=3,k=1)
```

- Run "python randomGraph2.py" from the command line
- Press Disconnect to clear everything from the pipeline browser
- Reload the state file `drawGraph.pvsm`

Few more graphs: complete bipartite graph

Composed of two partitions with N nodes in the first and M nodes in the second. Each node in the first set is connected to each node in the second.

```
$ diff randomGraph2.py completeBipartite.py
5,7c5,6
< H = nx.gnm_random_graph(numberNodes,numberEdges)
< print('nodes:', H.nodes())
< print('edges:', H.edges())
---
> H = nx.complete_bipartite_graph(10,5)
> print(nx.number_of_nodes(H), 'nodes␣and', nx.number_of_edges(H), 'edges')
15a15
> print('degree␣=', degree)
```

- Run “python completeBipartite.py” from the command line
- Press Disconnect to clear everything from the pipeline browser
- Reload the state file `drawGraph.pvsm`

Your own graphs

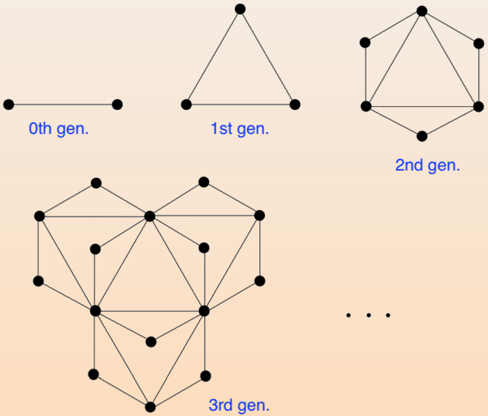
We are not limited to NetworkX's built-in graphs.
Can build our own graphs with:

```
H = nx.Graph()
H.add_node(1) # add a single node
H.add_nodes_from([2,3]) # add a list of nodes
H.add_edge(2,3) # add a single edge
H.add_edges_from([(1,2),(1,3)]) # add a list of edges
...
```

Dorogovtsev-Goltsev-Mendes graph

Dorogovtsev-Goltsev-Mendes graph is a fractal network from `http://arxiv.org/pdf/cond-mat/0112143.pdf`; in each subsequent generation:

1. every edge from the previous generation yields a new node, and
2. the new graph can be made by connecting together three previous-generation graphs



Dorogovtsev-Goltsev-Mendes graph (`dgm.py`)

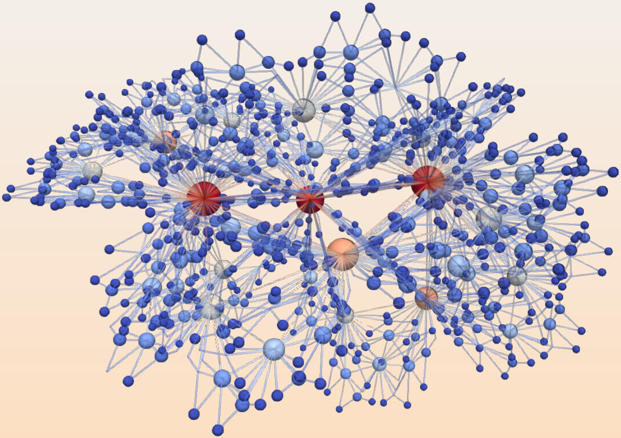
```
import networkx as nx
# from forceatlas import forceatlas2_layout
from writeNodesEdges import writeObjects
import sys
generation = int(sys.argv[1])
H = nx.dorogovtsev_goltsev_mendes_graph(generation)

# Force Atlas 2 from https://github.com/tpoisot/nxfa2.git
# pos = forceatlas2_layout(H, iterations=100, kr=0.001, dim=3)
pos = nx.spring_layout(H, dim=3)

# convert to list of positions (each is a list)
xyz = [list(pos[i]) for i in pos]

print(nx.number_of_nodes(H), 'nodes and', nx.number_of_edges(H), 'edges')
degree = [d for i,d in H.degree(H.nodes())]
writeObjects(xyz, edges=H.edges(), scalar=degree,
        name='degree', power=0.333,
        fileout='network')
```

Dorogovtsev-Goltsev-Mendes graph (7th generation)



From the command line run

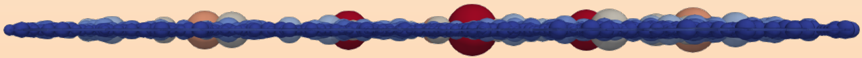
```
python dgm.py 1
...
python dgm.py 7    # takes ~3s on my laptop
```

Custom layouts ▷

Let's first make a flat graph:

```
9,10c9,10
< pos = nx.spring_layout(H, dim=3)
---
> pos = nx.spring_layout(H, dim=2)
13c13
< xyz = [list(pos[i]) for i in pos]
---
> xyz = [[pos[i][0], pos[i][1], 0] for i in pos]
```

Run this with `python dgmFlat.py 7`, reload the state file `drawGraph.pvsm`, adjust glyph radii

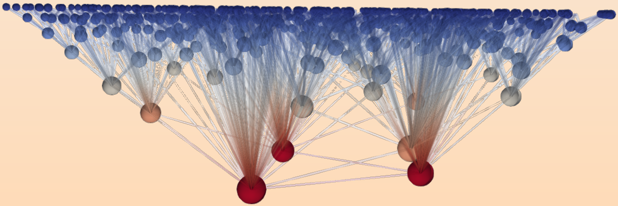


Custom layouts ▷▷

Now let's offset each node in the z-direction by a function of its degree:

```
$ diff dgmFlat.py dgmOffset.py
12,13d11
< xyz = [[pos[i][0], pos[i][1], 0] for i in pos]
15a14,15
> xyz = [[pos[i][0], pos[i][1], (degree[i])**0.5/15.] for i in pos]
```

Run this with "python dgmOffset.py 7" and colour edges by degree



Social network (`florentineFamilies.py`)

Let's visualize `nx.florentine_families_graph()`. It returns a list of edges with the nodes indexed by the family name. The function writeObjects() expects integer ID indices instead – hence the loop below: (when plotting, don't forget to turn on the labels!)

```
import networkx as nx
from writeNodesEdges import writeObjects
H = nx.florentine_families_graph()
nodes = list(H.nodes())

# index edges by their node IDs
edges = []
for edge in H.edges():
    edges.append([nodes.index(edge[0]),nodes.index(edge[1])])

pos = nx.spring_layout(H,dim=3,k=1) # return a dictionary of positions keyed by node
xyz = [list(pos[i]) for i in pos] # convert to list of positions (each is a list)

degree = [d for i,d in H.degree(H.nodes())]
writeObjects(xyz, edges=edges, scalar=degree, name='degree',
             fileout='network', nodeLabel=nodes, power=0.333)
```

Highlighting individual nodes (and edges)

Let's highlight nodes 'Strozzi', 'Tornabuoni', 'Albizzi' with colour.

```
$ diff florentineFamilies.py florentineFamilies2.py
14c14,17
< degree = [d for i,d in H.degree(H.nodes())]
---
> degree = [1]*len(nodes)
> selection = ['Strozzi', 'Tornabuoni', 'Albizzi']
> for i in selection:
>     degree[nodes.index(i)] = 3
```

How about highlighting the selection and their edges?
That's very easy: simply colour the edges by node degree.

Eigenvector centrality (`dgmCentrality.py`)

Let's compute and visualize eigenvector centrality in the 5th-deneration Dorogovtsev-Goltsev-Mendes graph with our custom 3D layout.

```
import networkx as nx
# from forceatlas import forceatlas2_layout
from writeNodesEdges import writeObjects
H = nx.dorogovtsev_goltsev_mendes_graph(5)
# pos = forceatlas2_layout(H, iterations=100, kr=0.001, dim=2)
pos = nx.spring_layout(H, dim=3)
print(nx.number_of_nodes(H), 'nodes and', nx.number_of_edges(H), 'edges')
degree = [d for i,d in H.degree(H.nodes())]
xyz = [[pos[i][0], pos[i][1], (degree[i])**0.5/5.7] for i in pos]

# compute and print eigenvector centrality
ec = nx.eigenvector_centrality(H) # dictionary of nodes with EC as the value
ecList = [ec[i] for i in ec]
print('degree =', degree)
print('eigenvector centrality =', ecList)
print('min/max =', min(ecList), max(ecList))

writeObjects(xyz, edges=H.edges(), scalar=degree, name='degree', power=0.333,
        scalar2=ecList, name2='eigenvector centrality', power2=0.333, fileout='network')
```

- Run `python dgmCentrality.py` and load into ParaView by hand
- Colour by degree, size by eigenvector centrality

Other statistics in NetworkX

- Various centrality measures: degree, closeness, betweenness, current-flow closeness, current-flow betweenness, eigenvector, communicability, load, dispersion `https://networkx.github.io/documentation/stable/reference/algorithms/centrality.html`
- Several hundred built-in algorithms for various calculations `https://networkx.github.io/documentation/stable/reference/algorithms`

Graphs in Plotly (`dgmDirect.py`)

Last tested in 2019. Since then many functions have been redefined or moved around ...

```
import plotly.offline as py, plotly.graph_objs as go, networkx as nx, sys
py.init_notebook_mode(connected=True)
gen = int(sys.argv[1])
H = nx.dorogovtsev_goltsev_mendes_graph(gen)
print(H.number_of_nodes(), 'nodes and', H.number_of_edges(), 'edges')
pos = nx.spring_layout(H, dim=3)
Xn = [pos[i][0] for i in pos]; Yn = [pos[i][1] for i in pos]   # node coordinates
Zn = [pos[i][2] for i in pos]; Xe, Ye, Ze = [], [], []
for edge in H.edges():
    Xe += [pos[edge[0]][0], pos[edge[1]][0], None]   # edge ends' coordinates
    Ye += [pos[edge[0]][1], pos[edge[1]][1], None]
    Ze += [pos[edge[0]][2], pos[edge[1]][2], None]
degree = [deg[1] for deg in H.degree()]   # list of degrees of all nodes
labels = [str(i) for i in range(H.number_of_nodes())]
edges = go.Scatter3d(x=Xe, y=Ye, z=Ze, mode='lines',
             line=go.Line(color='rgb(160,160,160)', width=2), hoverinfo='none')
nodes = go.Scatter3d(x=Xn, y=Yn, z=Zn, mode='markers',
            marker=go.Marker(sizemode = 'area', sizeref = 0.01, size=degree,
                             color=degree, colorscale='Viridis',
                             line=go.Line(color='rgb(50,50,50)', width=0.5)),
                     text=labels, hoverinfo='text')
axis = dict(showbackground=False, showline=False, zeroline=False, showgrid=False,
            showticklabels=False, title='')
layout = go.Layout(title = str(gen) + "-gen Dorogovtsev-Goltsev-Mendes graph",
    showlegend=False, scene=go.Scene(xaxis=go.XAxis(axis), yaxis=go.YAxis(axis),
                                     zaxis=go.ZAxis(axis)), margin=go.Margin(t=100))
fig = go.Figure(data=[edges,nodes], layout=layout)
py.iplot(fig)
```

Visualizing continuous distributions in 3D

Mockup 2D continuous function

2D function defined inside a unit square ($x, y \in [0, 1]$)

$$f(x, y) = (1 - y)\sin(\pi x) + y \sin^2(2\pi x)$$

discretized on a 30^2 Cartesian grid and stored in `2d000.vtk`

- Load the data into ParaView
- Display $f(x, y)$ in 2D
- Apply the WarpByScalar filter to display it in 3D

Mockup 3D continuous function

3D "sine envelope wave" function defined inside a unit cube ($x_i \in [0,1]$)

$$f(x_1,x_2,x_3) = \sum_{i=1}^{2}\left[\frac{\sin^2\left(\sqrt{\xi_i^2+\xi_{i+1}^2}\right)-0.5}{\left[0.001(\xi_i^2+\xi_{i+1}^2)+1\right]^2}+0.5\right], \text{ where } \xi_i \equiv 30(x_i-0.5)$$

discretized on a 100^3 Cartesian grid and stored in `sineEnvelope.nc`

1. Load the data into ParaView as "NetCDF generic"
2. Surface view
3. Clip filter
4. Slice filter
5. Contour filter at $f(x,y,z) = 0.3$ and 0.115 (we'll use the former in the last section)
6. Volume view
7. On presenter's laptop see *growth.mp4*

Creating animations in ParaView

Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "Forward"
2. Use ParaView's ability to recognize a sequence of similar files
   - time animation only, very convenient
   - try loading `data/2d*.vtk` sequence and animating it (visualize one frame and then press "Forward")
3. Script your animation in Python (not covered in this workshop)
   - steep learning curve, very powerful, can do anything you can do in the GUI
   - typical usage scenario: generate one frame per input file
   - a simpler exercise without input files: see next slide

Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "Forward"
2. Use ParaView's ability to recognize a sequence of similar files
   - time animation only, very convenient
   - try loading `data/2d*.vtk` sequence and animating it (visualize one frame and then press "Forward")
3. Script your animation in Python (not covered in this workshop)
   - steep learning curve, very powerful, can do anything you can do in the GUI
   - typical usage scenario: generate one frame per input file
   - a simpler exercise without input files: see next slide

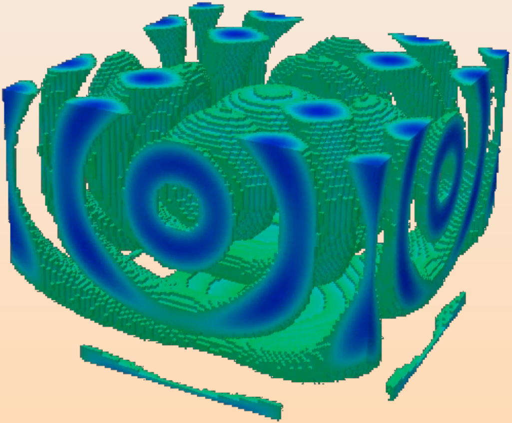
Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "Forward"
2. Use ParaView's ability to recognize a sequence of similar files
   - time animation only, very convenient
   - try loading `data/2d*.vtk` sequence and animating it (visualize one frame and then press "Forward")
3. Script your animation in Python (not covered in this workshop)
   - steep learning curve, very powerful, can do anything you can do in the GUI
   - typical usage scenario: generate one frame per input file
   - a simpler exercise without input files: see next slide

Exercise: animating function growth ▷

- 3D sine envelope wave function defined inside a unit cube ($x_i \in [0,1]$)

$$f(x_1,x_2,x_3)=\sum_{i=1}^{2}\left[\frac{\sin^2\left(\sqrt{\xi_{i+1}^2+\xi_i^2}\right)-0.5}{\left[0.001(\xi_{i+1}^2+\xi_i^2)+1\right]^2}+0.5\right], \text{ where } \xi_i\equiv 15(x_i-0.5)$$



- Reproduce the movie on the screen
  `https://vimeo.com/248501176`
  or *growth.mp4* on presenter's laptop

Exercise: animating function growth ▷▷

To visualize a single frame of the movie:

1. load `data/sineEnvelope.nc` (discretized on a 100^3 grid)
2. apply Threshold keeping only data from 1.2 to 2
3. apply Clip: origin $O = (49.5, 15, 49.5)$, normal $N = (0, -1, 0)$
4. colour by the right quantity

Two possible solutions:

1. bring up **Animation View** to animate Clip's O_2 from 0 to 99, for best results save animation as a sequence of PNG files
2. not covered in this workshop) Start/Stop Trace to record the workflow, save the corresponding **Python script**, enclose parts of it into a loop changing O_2 from 0 to 99 and writing a series of PNG screenshots, run it inside ParaView to produce 100 frames

in either case, merge PNGs using a 3rd-party tool, e.g.

```
ffmpeg -r 30 -i frame%04d.png -c:v libx264 -pix_fmt yuv420p \
       -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" movie.mp4
```

Camera animation in the GUI

Good introductory resource `https://www.paraview.org/Wiki/Advanced_Animations`

1. Start with any static visualization
2. Click on 'Adjust Camera' icon (one of the left-side icons on top of the visualization window)
   - adjust / write down Camera Focal Point
3. Bring up Animation View (or erase all previous timelines)

(3a) In Animation View:

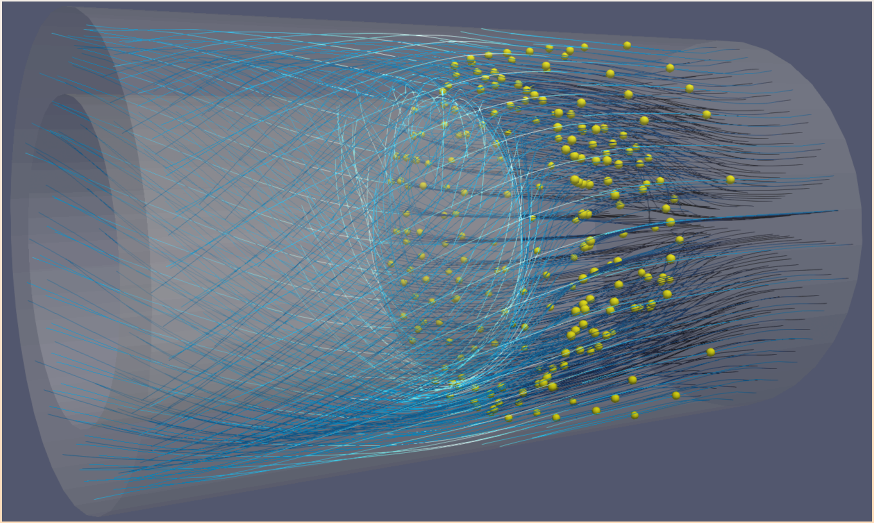
- select Camera - Orbit
- click "+" to create a new timeline
- set Center = Camera Focal Point, for the rest accept default settings
- adjust the number of frames

(3b) In Animation View:

- select Camera - Follow Path
- click "+" to create a new timeline
- double-click on the white timeline
- double-click on Path... in the right column
- click on Camera Position
  - a yellow path with spheres will appear
  - drag the spheres around
- also can change Camera Focus and Up Direction

4. Click "Forward"

Animating a stationary flow: time contours ▹



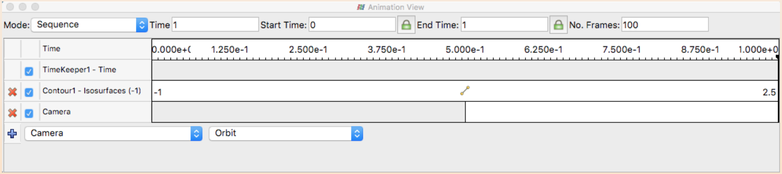
`https://vimeo.com/248509153` or *timeContours.mp4* on presenter's laptop

Animating a stationary flow: time contours ▷▷

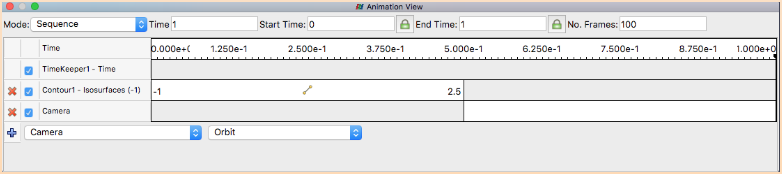
- Start with the streamtracer lines, however drawn
- Apply a Countour filter to the output of Streamtracer
  - contour by Integration Time
  - probe the range of values that works best
- Apply Glyph filter to the output of Countour
- Animation View: animate Contour - Isosurfaces
- This video was recorded with 2000 frames at 60 fps
  - such high resolution only for the final production video
  - debugging animation with 100 frames is perfectly Ok

Combining many timelines in one animation ▷

- Start with the previous integration-time-contour animation
- Add the second timeline to the animation: Camera - Orbit from $t = 0.5$ to $t = 1$ (while the first animation is still playing for its second half)

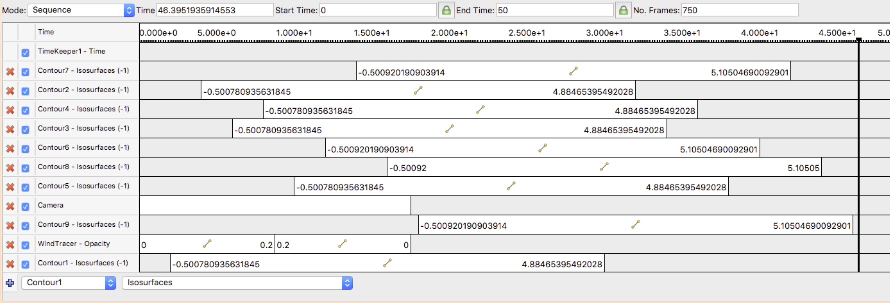


- Now complete integration-time-contour animation before rotation



Combining many timelines in one animation ▷▷

- In principle, can add as many timelines (with their individual time intervals and variables!) to the animation as you want
- Here is an example from WestGrid's 2017 *Visualize This* competition submission by Nadya Moisseeva (UBC) `https://scivis2021.netlify.app/2017`



- Here are couple more animation examples `https://ccvis.netlify.app/gallery/photorealistic-rendering`

Putting 3D visualizations on the web

- E.g. historical artifacts, digital prototypes, 3D buildings or terrains, point cloud (lidar) maps
- Would like a visitor to your page to be able to
  - rotate the object in 3D, zoom in/out
  - perhaps click on some predefined hotspots to launch additional actions

Example: Smithsonian 3D digitization

The Smithsonian museum has a collection of 3D textured models
`https://3d.si.edu`

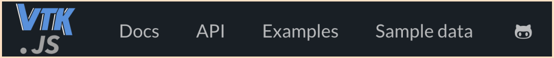
- Implemented their own Smithsonian X 3D Explorer viewer, a JavaScript/WebGL application talking to a proprietary server
- Navigate objects in 3D or 2.5D (six preset viewpoints) on low bandwidth
  - setup toolbox to compare different objects side-by-side in split mode
  - material toolbox to adjust colours, opacity, reflection, occlusion shadows, etc.
  - lighting toolbox to adjust the direction, colour and intensity of up to 3 light sources
  - environment toolbox to change background colours and the background grid
  - tools toolbox to measure and dissect models, plot various profiles along lines
- Also hosts Chandra X-ray Observatory 3D models
  `https://3d.si.edu/collections/Chandra`
- Some models accompanied by an interactive guided tour
- Some models available for downloading
  - OBJ, STL (Stereo Lithography) - understood natively by ParaView
  - GLB = requires a plugin in ParaView
  - USDZ = open-source Universal Scene Description (with ZIP) from Apple/Pixar, can be read via ParaView Connector plugin

Recall VTK = Visualization Toolkit

- Software for 3D computer graphics, image processing, volume rendering, and scientific visualization
- In development since the early 1990s
- Open-source, multi-platform: Linux, Windows, Mac, the Web and mobile devices
- Core functionality written in C++, wrapped into other language bindings: Tcl, Python, Java
- Sits on top of a graphics library (typically OpenGL)
- Distributed-memory parallel processing via MPI
- Many-core and GPU architecture support via VTK-m (separate code base)

VTK.js

- Open-source JavaScript library for sci-vis on the web
  - not all VTK classes implemented
  - more comlex applications: vtk.js ES6 code can be integrated into a web application in Node.js environment, typically requires a web server for local testing and for deployment
  - simpler usage: can be directly imported as a script tag inside live HTML pages from a global CDN (content delivery network) such as `https://unpkg.com`
- Uses WebGL (check your browser compatibility `https://get.webgl.org`)
  - WebGL2 for best performance `https://get.webgl.org/webgl2` (Chrome, Firefox)
- Variety of visualization algorithms
- Main resource `https://kitware.github.io/vtk-js`



  - docs and tutorials assume JavaScript knoweldge and familiarity with browser devtools
  - check code examples under both API and Examples $\Rightarrow$ can run simpler examples inside live HTML pages

Basic example: render a cone (`glyphs.html`)

Drop this file into your browser

```
<html>
  <body>
    <script type="text/javascript" src="https://unpkg.com/vtk.js"></script>
    <script type="text/javascript">
      // create a basic cone object
      var cone = vtk.Filters.Sources.vtkConeSource.newInstance();
      cone.setRadius(0.3);
      cone.setResolution(50);
      var glyph = vtk.Filters.Sources.vtkSphereSource.newInstance();
      glyph.setRadius(0.015);
      glyph.setThetaResolution(30);
      glyph.setPhiResolution(30);
      // map polygonal data into renderable geometry
      var coneMapper = vtk.Rendering.Core.vtkMapper.newInstance();
      coneMapper.setInputConnection(cone.getOutputPort());
      var glyphMapper = vtk.Rendering.Core.vtkGlyph3DMapper.newInstance(); // special mapper with 2 connections
      glyphMapper.setInputConnection(cone.getOutputPort(), 0);             // cone output goes to input port 0
      glyphMapper.setInputConnection(glyph.getOutputPort(), 1);            // glyph output goes to input port 1
      // create an OpenGL object
      var coneActor = vtk.Rendering.Core.vtkActor.newInstance();
      coneActor.setMapper(coneMapper);
      coneActor.getProperty().setEdgeVisibility(true);
      var glyphActor = vtk.Rendering.Core.vtkActor.newInstance();
      glyphActor.setMapper(glyphMapper);
      // create a full-webpage renderer
      var fullScreenRenderer = vtk.Rendering.Misc.vtkFullScreenRenderWindow.newInstance();
      // from which you create a renderer itself
      var renderer = fullScreenRenderer.getRenderer();
      renderer.addActor(coneActor);   renderer.addActor(glyphActor);
      renderer.resetCamera();
      // and a render window
      var renderWindow = fullScreenRenderer.getRenderWindow();
      renderWindow.render();
    </script>
  </body>
</html>
```

ParaView Glance

https://kitware.github.io/paraview-glance

PV Glance is an open-source standalone web app for in-browser 3D sci-vis

- very easy to use, ideal for sharing pre-built 3D scenes via the web
- no server ⇒ up to medium-size data (server support planned in future versions)
- interactive manipulation of pre-computed polygons
  - volumetric images, molecular structures, geometric objects, point clouds
- written in JavaScript and vtk.js + can be further customized with vtk.js and ParaViewWeb for custom web and desktop apps
- source and installation instructions
  https://github.com/kitware/paraview-glance

---

1. Create a visualization with several layers, make **all layers visible in the pipeline**
2. Many options in File | Export Scene... ⇒ save as VTKJS to your laptop
3. Open https://kitware.github.io/paraview-glance/app
4. Drag the newly saved file to the dropzone on the website
5. Interact with individual layers in 3D: rotate and zoom, change visibility, representation, variable, colourmap, opacity

Automatically load a visualisation into Glance

`https://discourse.paraview.org/t/customise-pv-glance/2831`

- Use the query syntax `GLANCEAPPURL?name=FILENAME&url=FILEURL` to pass `name` and `url` to the web server
- E.g. using ParaView Glance website `https://kitware.github.io/paraview-glance/app?name=sineEnvelope.vtkjs&url=https://raw.githubusercontent.com/razoumov/publish/master/data/sineEnvelope.vtkjs`
  - shortened to `https://bit.ly/2KtPWNf`
- You can parse long strings with JavaScript (next slide)

Embed your vis into a website with an iframe

File `embed.html`

```
<html>
  <head>
    <title>Sine envelope function</title>
  </head>
  <body>
    <h1>3D sine envelope function</h1>
    <script>
      var app = "https://kitware.github.io/paraview-glance/app";
      var dir = "https://raw.githubusercontent.com/razoumov/publish/master/data/";
      var file = "sineEnvelope.vtkjs";
      document.write("<iframe src='" + app + "?name=" + file + "&url=" +
                     dir + file +
                     "' id='iframe' width='1100' height='900'></iframe>");
    </script>
    <p>More stuff in here</p>
  </body>
</html>
```

- JavaScript here only to parse long strings

Popular 3D model hubs

- `https://sketchfab.com` is probably the world's most popular commercial hub for 3D model hosting and sharing
  - free limited hosting
  - can upload polygonal data files
  - can publish directly to `https://sketchfab.com` from some applications and libraries, e.g. from `https://yt-project.org`
- USDZ sharing platform `https://usdzshare.com`

3DHOP = 3D Heritage Online Presenter

Caution: source code `https://github.com/cnr-isti-vclab/3DHOP` last updated in June 2020

- `http://3dhop.net` is an open-source package for presenting 3D high-resolution models online
  - ▶ from the Visual Computing Lab of the Istituto di Scienza e Tecnologie dell'Informazione, oriented toward the Cultural Heritage field
  - ▶ written in HTML and JavaScript
  - ▶ well-documented `http://3dhop.net/howto.php` + check their online demos!
- Can handle the following file formats:
  (1) single-resolution PLY (polygon file format)
    - ☞ ParaView can export scenes in PLY format
    - alternatively can use the 3D unstructured triangular mesh editor MeshLab `http://meshlab.sourceforge.net` to convert other formats 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
    - Nexus package `http://vcg.isti.cnr.it/nexus` (developed in the same lab) can convert to NXS
    - Nexus uses 3DHOP for visualization on the web

  (3) point clouds with $10^6 - 10^8$ points

Import 3D polygon file into 3DHOP ▹

(1) In your local ParaView build a "sine envelope" isosurface at $f(x, y, z) = 0.3$

(2) File | Save Data as PLY into `sineEnvelope.ply`, making sure to check Enable Coloring and Enable Alpha

- Serve it locally or remotely with 3DHOP:

```
git clone https://github.com/cnr-isti-vclab/3DHOP.git 3dhop-src
cd 3dhop-src/minimal
mkdir -p models/singleres/
cp /path/to/your/download/sineEnvelope.ply models/singleres/
cp 3DHOP_all_tools.html index.html    # page with all tools in a sidebar
sed -i "" -e "s|models/gargo.nxz|models/singleres/sineEnvelope.ply|" index.html
sudo python -m http.server 80         # Python 3
```

  - local demo: point your web browser at `http://localhost`

- You can also find `3dhop/index.html` inside the ZIP download

Import 3D polygon file into 3DHOP ▷▷

Creating interactive hotspots in a 3DHOP scene ▷

- I created a couple of hotspot meshes `ring.ply` and `top.ply`
  - loaded the original model into ParaView
  - twice applied Clip filter (result: Unstructured Grid), followed by Extract Edges (result: Polygonal Mesh)
  - for both File | Save Data as PLY into `top.ply` and `ring.ply`, making sure to check Enable Coloring and Enable Alpha
- Using `index.html` as template, I created a new file `hotSpots.html` in which we
  (1) defined mesh1, mesh2, mesh3,
  (2) set up ringSpot and topSpot objects,
  (3) defined "Hide Hotspots" and "Show Hotspots" buttons and added them to function actionsToolbar(),
  (4) defined actions in function onPickedSpot()
- You can see the changes with

```
diff index.html hotSpots.html
```

Creating interactive hotspots in a 3DHOP scene ▷▷

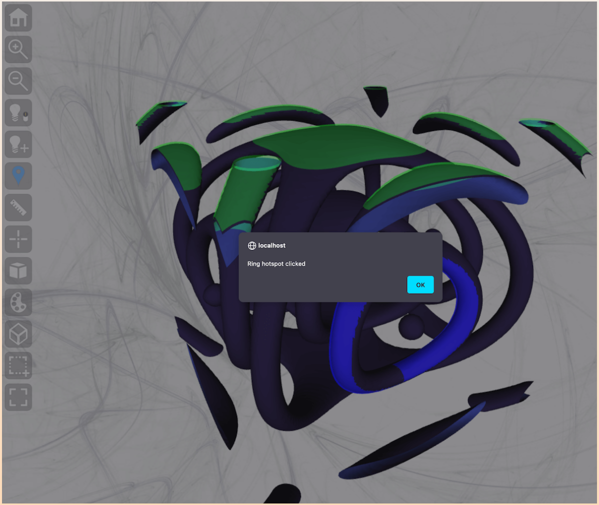
- Copy the hotspot meshes and a modified html into the corresponding directories and serve it locally or remotely with 3DHOP:

```
cd /path/to/3dhop-src/minimal
cp /path/to/your/download/{ring,top}.ply models/singleres
cp /path/to/your/download/3dhop/hotSpots.html .
sudo python -m http.server 80         # Python 3
```

  - local demo: point your web browser at `http://localhost/hotSpots.html`
  - now there is a button "Show/Hide Hotspots"
  - clicking on the top hotspot opens DHSI homepage in a new window
  - clicking on the ring hotspot opens an alert window

- You can find `3dhop/hotSpots.html` inside the ZIP download

Creating interactive hotspots in a 3DHOP scene ▷▷▷



- Today we concentrated heavily on VTK and general-purpose scientific visualization tools
  - 3D multi-attribute scatter plots
  - 3D graphs
  - continuous distributions
  - animations
  - putting 3D visualizations on the web

Questions?

- Email me at **alex.razoumov@westdri.ca**
- Submit a problem ticket at **support@tech.alliancecan.ca**
- Our visualization showcase and support `https://ccvis.netlify.app`
- Alliance | Western Canada visualization resources `http://bit.ly/vispages`