3D visualization for the humanities

Alex Razoumov
alex.razoumov@westgrid.ca

(1) copy of these slides and other files at http://bit.ly/dhfileszip
   ▶ will download dh.zip
   ▶ uncompressed it into your Downloads folder

(2) software installation (ParaView, Miniconda, few Python packages)
   ▶ described in slide 11
Introducing myself

• 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 HPC programming

• Day job: visualization and training coordinator for WestGrid ⇒ helping researchers with 3D scientific visualizations of large computational models http://bit.ly/cctopviz, teaching parallel programming

• So ... reading a 3D visualization course at DHSI

• My approach: apply scientific visualization tools to DH research
  ▶ think of it as extension of interactive 2D plotting into the third dimension using general-purpose, open-source tools
  ▶ research scenario: already have a 3D dataset and want to visualize it
  ▶ very interested in learning DH problems in 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)
  - special virtual environments such as Vsim (3D learning env. for DH), OpenSimulator (multi-user online env.)
  - don’t confuse this 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 (another DHSI course?)
- Photogrammetric processing of images
  - building polynomial texture maps from a set of images taken with varying lighting direction
  - building 3D models from a set of images taken from various directions
    - number of commercial offerings
    - for open-source [http://www.regard3d.org](http://www.regard3d.org) ... good topic for a PhD thesis?
    - rich area for VR and AR
- Visualization of point cloud data (can do it 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 attribute(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

• **Putting 3D visualizations on the web – without demos**
  ▶ the Smithsonian collection
  ▶ WebGL files
  ▶ vtk.js library on top of WebGL for client-side visualization
  ▶ http://3dhop.net, an open-source software package for creating interactive online presentations of high-resolution 3D models
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)
- Supports OpenGL hardware acceleration
- **C++ class library, with interpreted interface layers for Python, Java, Tcl/Tk**
- 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 several really good 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

- **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

- **Structured Grid**: *.vts, regular topology and irregular geometry, need to indicate coordinates of each point
VTK 2D/3D data: 6 major discretizations (mesh types)

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

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

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

• For your OS install ParaView from http://www.paraview.org/download
• For your OS install Python 3.7 Miniconda distribution from http://conda.pydata.org/miniconda.html
• Start the command shell (Terminal in MacOS/Linux, Anaconda Prompt in Windows) and then install the required Python packages:

  conda install numpy scipy networkx vtk
  conda install gensim scikit-learn pandas
  conda install jupyter # optional (Jupyter Notebooks)
  conda install plotly # alternative plotting tool

• Start Python and test your Miniconda installation:

  import vtk
  import networkx as nx

• Note: ParaView comes with its own Python shell and VTK, but it is somewhat tricky to install NetworkX there
Python function to write points and graphs as VTK

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

```python
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 post*

- 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 2231x2231 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

Problem: given a $24 \times 24$ 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) *Galilean-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
Workflow

(1) From each text pick up 30 longest paragraphs, ×5 texts ⇒ 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 $N_{\text{words}}$ words

(6) Vectorize each paragraph in the $N_{\text{words}}$-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 $N_{\text{words}}$-dimensional space ⇒ 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)
Workflow

(1) From each text pick up 30 longest paragraphs, ×5 texts ⇒ 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 $N_{\text{words}}$ words
(6) Vectorize each paragraph in the $N_{\text{words}}$-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 $N_{\text{words}}$-dimensional space ⇒ 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)
Workflow

(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 $N_{\text{words}}$ words

(6) Vectorize each paragraph in the $N_{\text{words}}$-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 $N_{\text{words}}$-dimensional space $\Rightarrow$ 150 $\times$ 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)
Workflow

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 $N_{\text{words}}$ words
6. Vectorize each paragraph in the $N_{\text{words}}$-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 $N_{\text{words}}$-dimensional space \Rightarrow 150 \times 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)
Workflow

(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 $N_{\text{words}}$ words

(6) Vectorize each paragraph in the $N_{\text{words}}$-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 $N_{\text{words}}$-dimensional space $\Rightarrow 150 \times 150$ 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)
**Workflow**

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 $N_{\text{words}}$ words
6. Vectorize each paragraph in the $N_{\text{words}}$-dimensional space, positioning it according to its word count; for details see [http://radimrehurek.com/gensim/tut1.html](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 $N_{\text{words}}$-dimensional space $\Rightarrow$ 150 $\times$ 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)
Workflow

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 \(N_{\text{words}}\) words
6. Vectorize each paragraph in the \(N_{\text{words}}\)-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 \(N_{\text{words}}\)-dimensional space \(\Rightarrow\) 150 \(\times\) 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)
Workflow

(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 $N_{\text{words}}$ words

(6) Vectorize each paragraph in the $N_{\text{words}}$-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 $N_{\text{words}}$-dimensional space $\Rightarrow 150 \times 150$ 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)
Implementation: running the script

- The entire algorithm is implemented in `semanticMapping.py`
  - let’s take a look at it
- Run it however you like
  - in a shell:
    
    ```
    cd directoryWithAllUnzippedFiles
    python semanticMapping.py
    ```
  - inside Jupyter Notebook:
    
    ```
    [Bash Terminal or Anaconda Prompt] jupyter notebook
    ...
    %cd Downloads\dh  # (in Windows) if running Jupyter notebook from your home
    %cd C:\Users\username\Downloads\dh    # (in Windows) typical absolute path
    %load semanticMapping.py    # fills the current cell with code from the script
    ```
  - or you could copy/paste commands manually from the script into your favourite Python environment
Implementation: viewing results in ParaView

• Load `texts.vtu` into ParaView and follow my instructions
  ▶ colour glyphs by “author”, switching from continuous to categorical colours and annotating them
    • 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.
Implementation: viewing results with plotly

```python
$ diff semanticMapping.py directMapping.py
95d94
< print(coords)
99,100c98,114
< writeObjects(coords, scalar=author, name='author', fileout='texts',
<     scalar2=novelPerAuthor, name2='novel per author', method = 'vtk...Grid')
---
> 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
> 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)')))
> layout = go.Layout(title='Each sphere is a paragraph coloured by author '+
>     'and sized by novelPerAuthor')
> fig = go.Figure(data=[spheres], layout=layout)
> py.plot(fig, filename='fourAuthors.html')
```
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 with
  ```
  python gospels.py
  ```

- Load `testament.vtu` into ParaView

- Continuous colouring with the default colour map shows `Matthew in blue`, `Mark in pale blue`, `Luke in beige`, `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 ← Matthew + Mark
  - Luke (beige) has a lot of overlap with Matthew (blue) and Mark (pale blue), 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: introduce Threshold filters 1 – 2.5 and 3.5 – 4, both acting on the output of the Glyph filter
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: modify the first Threshold filter to 1.5 – 2.5
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](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 “python countries.py” (writes five attributes into `countries.vtp`)

- 3D position by economy + entrepreneurshipOpportunity + governance, colour by education, size by safetySecurity – easiest to load the state file `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 $\geq 0$ (will labels all points) and press Run Selection Query

(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 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 $\leq 3$

(8) Now label all those with good education: education $\geq 1.5$
Labeling nodes

(1) Press V to bring up Find Data dialogue

(2) Find=Points from=countries.vtp with ID $\geq 0$ (will labels all points) and press Run Selection Query

(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 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 $\leq 3$

(8) Now label all those with good education: education $\geq 1.5$
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 Run Selection Query
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 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 Run Selection Query
(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 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
import plotly.offline as py
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.plot(fig, filename='bubbles.html')
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

- Force Atlas 3D plugin for Gephi [http://bit.ly/1QcLuLk](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](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)

- Walrus [http://www.caida.org/tools/visualization/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](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](http://www.opengraphiti.com) (pain to compile: tends to pick /usr/bin/python, only Mac/Linux), or WebGL projects [https://youtu.be/qHkjSxbnzAU](https://youtu.be/qHkjSxbnzAU) that really require programming knowledge
  - [https://markwolff.shinyapps.io/QMtriplot17C/](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](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 $10^{5.5}$ nodes, with a little extra care to $10^{7.5}$ nodes, and with some thought to $10^{9.5}$ 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 3.7

2) Load that file into ParaView

• On presenter’s laptop see mutOnCtOrbits.mp4 for a more complex graph ($6 \times 10^5$ edges) created with this workflow
• Our second solution: NetworkX + Plotly
  ▶ no intermediate steps: graph created directly in Python, opens automatically in a web browser
  ▶ everything is scriptable
  ▶ limited scaling
  ▶ similar 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](http://networkx.github.io)

```python
import networkx as nx

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

# generate a list (of 128) built-in graph types
# with Python's 'list comprehension'
[x for x in dir(nx) if '_graph' in x]```
NetworkX layouts

```python
# generate a list built-in graph layouts
[x for x in dir(nx) if '_layout' in x]
# will print ['circular_layout', 'fruchterman_reingold_layout',
#             'kamada_kawai_layout', 'random_layout',
#             'rescale_layout', 'shell_layout',
#             'spectral_layout', 'spring_layout']

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

- `spring_` and `fruchterman_reingold_` are the same, so really 7 built-in layouts
- can use 3rd-party layouts (you’ll see at least one later in this presentation)
- `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)

```python
# generate a random graph
H = nx.gnm_random_graph(10, 50)

# the first element of the dictionary is a 2D array
nx.shell_layout(H, dim=2)[0]  # only dim=2 supported
nx.circular_layout(H, dim=2)[0]

# the first element of the dictionary is a 3D array
nx.spring_layout(H, dim=3)[0]
nx.random_layout(H, dim=3)[0]
nx.spectral_layout(H, dim=3)[0]
```
Our first graph (`randomGraph.py`)
Load this graph into ParaView

- After you run “python 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"/>
```

- 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.

```bash
$ 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:

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

- every edge from the previous generation yields a new node, and
- the new graph can be made by connecting together three previous-generation graphs
Dorogovtsev-Goltsev-Mendes graph (dgm.py)

```python
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)

# 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 ~12s on my laptop
```
Custom layouts

Let’s first make a flat graph:

```bash
$ diff dgm.py dgmFlat.py
9c9
< pos=forceatlas2_layout(H, iterations=100, kr=0.001, dim=3)
---
> pos=forceatlas2_layout(H, iterations=100, kr=0.001, dim=2)
12c12
< 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
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!)

```python
copy
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)
```

```bash
$ 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.
Let’s highlight neighbours of the selected nodes.

```python
$ diff florentineFamilies2.py florentineFamilies3.py
17c17,20
<     degree[ nodes.index(i) ] = 3
---
>     degree[ nodes.index(i) ] = 3
>     for j in list(nx.all_neighbors(H,i)):
>         degree[ nodes.index(j) ] = max(2.5, degree[ nodes.index(j) ])
```
Eigenvector centrality (dgmCentrality.py)

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

```python
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)
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
c = nx.eigenvector_centrality(H) # dictionary of nodes with EC as the value
cList = [c[i] for i in c]
print('degree=', degree)
print('eigenvector centrality=', cList)
print('min/max=', min(cList), max(cList))

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

- Several hundred built-in algorithms for various calculations
Graphs in Plotly (**dgmDirect.py**)

```python
import plotly.offline as py, plotly.graph_objs as go, networkx as nx, sys
from forceatlas import forceatlas2_layout

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 = forceatlas2_layout(H, iterations=100, kr=0.001, dim=3) # http://bit.ly/2Lop9QZ
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.plot(fig, filename='network.html')
```
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 \((x_i \in [0, 1])\)

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

discretized on a 100\(^3\) Cartesian grid and stored in *sineEnvelope.nc*

- Load the data into ParaView as “NetCDF generic”
- Surface view
- Clip filter
- Slice filter
- Contour filter at \(f(x, y, z) = 0.3\) and 0.115 (we’ll use the former in the last section)
- Volume view
- See *growth.mp4*
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
Nice example: Smithsonian X 3D Explorer

The Smithsonian museum has a small online collection of 3D textured models

- implemented their own Smithsonian X 3D Explorer, a JavaScript/WebGL application talking to a proprietary server http://3d.si.edu/article/getting-started
- 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

- comprehensive guided *interactive tours* [https://s.si.edu/2JxaC5f](https://s.si.edu/2JxaC5f) telling an interesting story about each object
- some models available for downloading after registration [https://3d.si.edu/download-browser](https://3d.si.edu/download-browser)
Smithsonian downloads

- Most Smithsonian artifacts stored as OBJ files, an open specification that encodes the 3D geometry, i.e., the position of each vertex and the faces (lists of vertices, normals, and texture)

- On presenter’s laptop we can take a look at either (both at two different resolutions)
  - The Cosmic Buddha, a headless figure from 6th-century China
  - The Palmyra Haliphat, a Greco-Roman funerary bust from 2nd-century (AD) Palmyra

  http://bit.ly/2JxbFCd

  1. load into ParaView, study the information tab
  2. apply the clip filter to ensure that we are dealing with a surface mesh
  3. switch to the wireframe view and zoom in

- Can view these in many different programs (even MacOS Preview!)
Export scene as WebGL
Simplest but not the best solution

(1) In your local ParaView build a “sine envelope” isosurface at
\[ f(x, y, z) = 0.3 \]

(2) File \(\rightarrow\) Export Scene as WebGL to sine.webgl \(\Rightarrow\) this will produce a
number of files with the same base name and different extensions
including sine.html and sine.webgl

- **Local demo**: load sine.html into your browser
- **Remote demo**: copy sine.html into a directory, cd there and serve files
  from the current directory on the default HTTP listening port 80

\[
\text{cd /path/to/sineDemo}
\]
\[
\text{sudo python -m http.server 80} \quad # \text{Python 3}
\]

- 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

- 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 http://bit.ly/StandaloneSceneLoader
  1. use their ParaView Python script export-scene-macro.py 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
  4. press C for a configuration panel, W to display all layers as wireframe, V as points, S as surfaces, R to display default zoom/position

- Live vtk.js + WebGL geometry rendering examples from Kitware:
  - machinery http://bit.ly/2nh1FSq
Probably, 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
3DHOP = 3D Heritage Online Presenter

- Open-source package for presenting 3D high-resolution models online [http://3dhop.net](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/howto.php)

- 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](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](http://vcg.isti.cnr.it/nexus) to convert to NXS
  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) In ParaView you can File→Save Data as PLY, but it does not write vertex colours and normals (both nice to have) – instead, we’ll do this:

- export it File→Export Scene to sineEnvelope.x3d
- open this scene in MeshLab File→ImportMesh and export it File→ExportMeshAs as sineEnvelope.ply making sure to save vertex normals

• Serve it locally or remotely with 3DHOP:

```bash
git clone https://github.com/cnr-isti-vclab/3DHOP.git
cd 3DHOP/minimal
mkdir -p models/singleres/
cp ~/teaching/humanities/sineEnvelope.ply models/singleres/
cp index_all_tools.html index.html
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 find 3dhop/index.html inside the ZIP download
Import 3D polygon file into 3DHOP (cont.)
Creating interactive hotspots in a 3DHOP scene

- **Described in** [http://3dhop.net/examples.php?id=7](http://3dhop.net/examples.php?id=7)

- I created a couple of *hotspot meshes* *ring.ply* and *top.ply*
  - loaded the original model into ParaView and 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. defined mesh1, mesh2, mesh3,
  2. set up ringSpot and topSpot,
  3. defined Hide/Show Hotspots buttons and add them to function actionsToolbar(),
  4. defined actions in function onPickedSpot()
Creating interactive hotspots in a 3DHOP scene (cont.)

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

```
cd /path/to/3DHOP/minimal
cp ~/teaching/humanities/{ring,top}.ply models/singleres
cp ~/teaching/humanities/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*
• Today we concentrated heavily on VTK and general-purpose scientific visualization tools
  ▶ 3D multi-attribute scatter plots
  ▶ 3D graphs
  ▶ continuous distributions
  ▶ putting 3D visualizations on the web

Questions?

• Email me at alex.razoumov@westgrid.ca
• Submit a problem ticket at support@computecanada.ca
• Compute Canada visualization showcase http://bit.ly/cctopviz
• WestGrid’s Visualize This! annual contest https://westgrid.github.io/visualizeThis