xyzviewer

data expiration: combines basic database processing with interactive 3d graphics and VR

availability

- github source at https://github.com/sjpt/xyzviewer
- demo version at https://sjpt.github.io/xyz/xyz
  - eg https://sjpt.github.io/xyz/xyz.html?startdata=data/small_test.csv
  - Star Carr archaeology data https://sjpt.github.io/xyz/xyz?arch
- FFL: I think a short report describing the viewer could be written and uploaded to arxiv: it would help spread the impact. Yes.

VR (webXR) appears to be working; including on phone/Google cardboard.

data storage and manipulation

Current input: one or more files. CSV, geojson, ply (3d models), XLSL, image?, TData (.yaml + .colbin)
CSV is the 'main' one. (can also be called .txt),

data source/view separation

- xyzviewer separates data storage (see TData below) from XYZ views
  - supports multiple views of same data by duplicate data load
- should also be easy to add viewport to view if needed
  - needs separate render for different objects, viewport belongs to renderer not three object

CSV

capability

- Currently XYZ columns are usually mapped to graphics xyx
- any column used for filter/colourby/marker/photo.
- Should have much more flexible mapping for generating graphs etc
  - all following WGS [2]
  - xyz can be overridden by compute# below, but need more autoscale etc details
- May be sensible to combine filter/colourby/etc
  - eg compute/# operator in PRTV [1]
  - often more efficient and useful than separate select (js filter) and compute (js map)
  - aside warning: javascript map on typed array must produce array of same type.

TData, CSV/relation implementation

Todo: research existing in-store javascript databases to see if we can use one.
In particular Steve has asked about anndata.
https://anndata.readthedocs.io/en/latest/fileformat-prose.html#dense-arrays

CSV is essentially a tabular data equivalent to relations). It was originally held in 'natural' array of javascript objects, but that was inefficient in storage and processing.

Data is now (from around 25/11/2020) held in column format (very similar to Ray Lorie's)

incremental load

- CSV files are loaded, parsed and prepared with incremental load
    - This is more efficient, and means files can be loaded that could not be with 'bulk' load
    - Feedback and any preparation processing is done during incremental load
        - including painting of points so far

in-memory structures

- Hold column oriented data
    - Incoming CSV files converted to columns data during load/parse
- Each column in a typed array, currently (27/11/2020) always Float32Array
    - Numbers are held as Float32 in column array
    - Character data uses value sets; one per column
        - value set, maps value->enum and enum->value
        - the columns array entry is a tagged NaN, tag points into value set.
    - This allows every column to be a hybrid of number and character data if required.
    - The float32Array columns can be used directly to create three.js/GL attribute buffers
        - for very quick presentation of the data to custom shaders
- TData also holds extra information about objects, for each column in **tdata.ranges[colname]**
    - **min, max, mid, range:** minimum, maximum, and half-way value and difference
    - **numNum, numStrs, numNull**: number of number, string and null values in the column
    - **mean, sd, sum, sum2**: mean and standard deviation, and sums used to compute them
- 'pure' character columns could use UInt8Array or UInt16 array as needed
    - columns that are almost all character with some numbers could do this, and put the numbers into the value set.
- 'free' columns (eg comments) with very large number of different values
    - tbd, probably js array of strings, would not save a huge amount
- Columns commonly used together are combined
    - eg x,y,z or r,g,b, into 3*#rows Float32Array
    - The combined data may well be usable directly for main graphics attributes
- Equal value comparison with strings could be converted to comparison at the vs key level.
    - value-sets could be sorted so that string comparison order was the same as vs key order
    - That would mean > < comparisons could also be carried out at the vs key level.
    - Comparisons at the vs key level could easily be implemented in shaders.

This column structure requires a little more intelligence in compiling filters/colourby expressions etc,
but is a huge space and time saver (especially with lazy column load).
Should be good for many mega-rows of data.

Proxy for JSon objects

TData provides a proxy that presents internal TData as Json style objects, so programs can use **data[columnName][index]** format.

- This can be used for backwards compatibility for programs already using Json format (eg CIView, MLV).
- This provides the benefits of much lower Javascript heap demands than 'real' Json objects
- plus the benefits of lazy column load (minor changes to the app) and incremental load (bigger app changes needed)
- but provides slower data access then real Json objects.
    - informal performance tests indicate this is not a very large impact
- The proxy does NOT provide iteration over the objects.
- Apps can migrate towards full TData use by mixing use of the proxy with rewrites of performance critical parts.
- The use of a proxy can be detected and the TData object (if any) obtained from **data[i]._tdata**

CSV persistent storage format

- persistent storage is very close to in-memory storage
    - one file per column or column group (xyz)
    - one 'control' file in yaml format
        - contains all the value sets
        - yaml better than json at handling circular references
        - NOT DONE value set data could be held separately for each column,
            - either within the columns data file or as a separate column file
            - for datasets so far encountered this would not have any benefit
- column structure can be saved into files (save columns button, TData.savefiles())
- Column data loaded lazily on demand by client
    - I guess most columns will not be used at all in most sessions
    - could zip entire set, but that loses lazy column load

Persistent storage location

Web browser security limitations place restrictions on data location that do not apply to installed programs.

Persistent storage (for csv and for other files) may saved

- co-hosted with xyzviewer and accessed by url
- hosted elsewhere and accessed by url, subject to CORS limitations
    - or a proxy used to evade CORS limitations
    - the PDB database permits such access
    - the archaeology data service archives do not permit such access
- be in local files
    - browser limitations mean these need user interaction each session.
    - files may be drag-dropped or opened for immediate processing
        - multiselect permitted
    - a directory may be drag-dropped
        - the processable files are put into a dropdown list where they can be selected for processing.
- NOT YET SUPPORTED be held locally in indexedDB
    - could have issues with disk space being used but 'hidden' from user

filter, colours and columns

Data can be filtered and coloured (currently 30/11/2020 some of this applies only to CSV data).

The control box provides several special functions. Each line of the multiline control box may have an appropriate prefix.

- data filtering by javascript expression (**?**)
- choice of data columns to display (**X:**,**Y:**, **Z:**)
- second point for lines (**X1:**,**Y1:**, **Z1:**)
- data colouring by column name (**COL:**), for second end of lines (**COL1:**)
- data colouring by javascript expression (**COLX:**) for second end of lines (**COLX1:**)
- general Javascript (no prefix)

The contents of the control box are compiled to a Javascript function. The compiled function is displayed in green below the control box; mainly for diagnostic purposes during development and debug.

Lines are executed in order. Colour, filter and general lines may be interleaved in any order. Where possible putting filters earlier will be more efficient, though the difference is likely to be small.

field and shortcut names

Fields can be referenced by their name (column heading). There are currently limitations where the field names are not valid Javascript names. There are also shortcut expressions

- **<name>:** the system will adjust;
    - for predominantly alpha fields it will use the enumeration value (**_EN**)
    - if XYZ.autorange is set (default) it will use the normalized value **(_N**)
    - if XYZ.autorange is not set it will use the raw value (used for Star Carr) (**_R**)
- **<name>_N**: normalized value assuming numeric field, currently normalized with 0 and 1 being 1.5 standard deviations from the mean.
- **<name>_R**: raw value (as in source data) assuming numeric field
- **<name>_E**: enumeration value assuming character field (0..n-1 for n values)
- **<name>_EN**: normalized enumeration value (0..(n-1)/n for n values)
- **<name>_C**: character string value
- **_R**: red channel, **_G**, green channel, **_B**: blue channel
- **_L**: current or most recently defined lasso
- **_L<n>**: (n single digit) lasso number n

Also shortcut 'functions'

- `RGB(<r>, <g>, <b>)` where **<r>** etc are javascript expressions, often just a normalized field, will set the colour to the given rgb values.
- **VX(<k>)** to multiply rgb values by a value in expression <k>
    - for example **VX(0.5 + _L);** will darken non-lassoed values and brighten lassoed ones,

filter

A filter is a line prefixed by **?** and consists of a Javascript boolean expression: eg

- **?x>0**
- **?_L**

colouring

n.b. bug: colour setup does not behave correctly when moving between different datasets.

A colour choice is set in a line prefixed **COL:**. The value is a field name.
COL: yr

Colours may also be chosen from the Colour by dropdown of field names and the colour picker next to it. This will set the **COL:** field in the control box.

Columns that contain predominantly character values are coloured by that value. A key of colours is shown. Initially colours are set from a (small) standard colour set. The colour scheme may be modified by loading (eg by drag-drop) a **.cols** file. This is a YAML or JSON file containing a two level dictionary: the top level is field name and the second level is field value, with the value being a three.js (CSS?) colour constant. Field directories from different files accumulate, inner value directories are overwritten by subsequent .cols files.

Columns that contain predominantly number values are coloured by the number value; using a range of 2 standard deviations each side of the mean. TODO: add tailoring of the colour scheme.

More complex expressions may be written with `RGB(<r>, <g>, <b>)` where **<r>** etc are javascript expressions, often just a normalized field.

display columns

By default xyzviewer will use columns named x, y, z respectively for the position values. This may be overridden using **X:**, **Y:**, **Z:** prefix lines in the box; each takes a Javascript expression.

Currently rows that do not have valid x, y, z, value will not load; so cannot be displayed even with the overrides. There is currently no auto ranging for columns other than x, y, z.

Javascript

The control box may also contain lines of arbitrary Javascript. These can access data and make computations as long as they follow our compilation conventions. These conventions are subject to change at any time. The current compilation can be seen as a guide in green below the control box.

picking

click

Picking is currently by click. The top 10 hits are summarized in a list, and hovering over a pick in that list will bring up details of the object.

There are several issues with this. There is no gui for pick radius. Pick does not allow for displayed point size.

lasso

creating lasso

A 3d lasso is painted onto the screen. (details here subject to change). Select the 'lasso' checkbox and paint one or more lasso segments on the screen (each mousedown, mousemove, mouseup sequence). Deselect lasso checkbox when all segments complete.

Lasso currently only applies to xyz style point data.

The painting may be in different modes, which may be interleaved. (results may be odd if you change mode during painting one segment).

- default, additive mode
- **alt key** held, subtractive
- **ctrl key** held, XOR
  - XOR can paint exact shapes, and interesting Picasso-like patterns, but it is very difficult to correct any errors once made.

The 3d lasso remembers the screen map and the transformation matrices under which it was drawn.

Multiple lassos may be held. A new lasso is created each time the lasso checkbox is checked, and completed when the checkbox is unchecked. Display of last or earlier lassos not sensible after view transformation has been altered.

applying lasso

The current or most recent lasso is used in the filter/colour control box, referenced by '**_L**'. e.g.

- **`?_L`**: select just the lassoed points.

More complex usage such as reference to multiple lassos can be made, writeup pending easier compilation

There are a couple of experimental shortcuts to apply lasso. Details will change

- If the shift key is down during mouse move, the lasso colouring is automatically applied as the lasso is painted. (NOT suitable for large datasets)
- When the lasso is complete and there is no reference to lasso in the control box, the lasso is applied as a filter.

lasso implementation in shaders

NOT YET AVAILABLE

- apply as direct filter in point cloud vertex shader (almost certainly near realtime for 8m points)
- or as filter to generate filter mask
    - filter mask logically 1d texture of width of #rows
    - folded to 2d texture for implementation limitations
    - filter mask itself may be used by other shader, or read back and used in javascript

pick feedback

Still probably need picking where information about selected points is fed back in tabular form.
Picking against large datasets should also use gpu, similar options to lasso above

graphics details

efficiency

Update graphics for efficiency.

- use buffers, not higher level geometry.vertices etc (done for Points)
- options for 'raw' points (not circle sprite)

GPU and shaders

Several features could be implemented in shaders to save interaction time if appropriate.
Lasso and pick (as above), also filtering and colour by.

relationship to other software

MLV

See separate section: MLV integration

CSynth

- There may well be value merging xyzviewer into CSynth.
- Should be pretty easy.
- There is a (small) amount of overlapping code

LMV (large matrix viewer)

- No immediate connection
- LMV may also be useful for adding to MLV (not sure enough of aims of MLV to be sure)
- Like xyzviewer, LMV handles very large volumes of point cloud data
    - one point per matrix entry for
    - applied 4 times for the 4 views

phone interface and speech

- ? what does Google Earth do on cardboard?
- tentative speech interface, limited instruction set for now
  - needs correct interpretation of orientation
    - should forward allow y/up change?
  - using interim to stop continuous action sensible; but …
  - needs undo
  - sound issues with phone in cardboard (bluetooth might help)

structure and plugins

xyzview has an original primary function of showing xyz points from csv or similar files. This has gradually been added to. The current directory structure contains

- main files and support for csv (and related) in root directory
- support files (mainly borrowed from elsewhere) in the **./jsdeps** directory
- support for additional file types in the **./plugins** directory
  - plugin dependencies are also in the **./plugins** directory
- extra demonstrations in the **./extras** directory (currently the early virus folding demo)

plugins

Plugins to support a particular file extension (eg xxx) should be in the form of javascript modules, under the name **./plugins/xxxreader.js**
The plugin should register itself with a call to **`addFileTypeHandler`**, imported as
`import {addFileTypeHandler} from '../basic.js';`

`addFileTypeHandler` takes an extension and function, eg **`addFileTypeHandler('.xxx', xxxhandler)`**
`xxxhandler` should have two parameters, **data** and **fid** (file name)
data will generally be read by xyzviewer infrastructure and passed as a string to the handler.

In some cases it can be passed as another datatype (eg and array buffer) (details to be clarified)
If **xxxhandler.rawdata** is set the data will be a File which it is the responsibility of the handler to read. This is needed for incremental loading of files.

When the handler is called for a particular file it can create three.js objects, and then attach them to the three scene with eg **`addToMain`**`(chainlines, 'chainlines');`
addToMain takes two parameters, a three.js object and a name. The name will appear in the gui, with a visibility checkbox.

A plugin can register multiple file extensions, but only can be automatically loaded from it's primary extension. For now, one way to organize automatic loading from secondary extensions (eg xxx) is to make stub javascript modules, eg **./plugins/yyyviewer.js** to contain **export{}; import {} from './xxxviewer.js';**

multiple views of the same data, selection etc

It is currently only possible to have multiple views of a dataset by loading the dataset multiple times. We expect to arrange a more formal and efficient view structure soon.

Selection of a view/object enable parameters of that object to be modified in the gui. There is currently special case code for xyz objects … to be clarified/formalized for other objects.

common issues

apply to xyzviewer, CSynth, and more generally

- saving and sharing user settings (collaboration)
  - code fragments
  - plugins
- CORS and navigation of public data
- federation

todo

- hover help
- get multidimensional/shader interface working better
- better match between javascript 'filter' and shader version
- correct lasso when scaling, moving camera etc
  - do not show lasso as region when it will be wrong anyway
- consider OrthographicCamera
- remember filter settings for each active xyz, and show as current is changed
  - improve selection of 'current'
  - do NOT remember filter etc in saved yaml ???, or get it right
- graphics interface enhancements as in 'graphics detail'
- storage structure enhancements, as in 'data storage and manipulation'
- pick feedback on lines
- lines as well as points in 'standard' xyz
- lasso, new mousedown/mousemove while filtering result of previous mouseup
  - helped by lasso shader
  - have 'fast' version of lasso capture, or better shader version of capture
  - prevent piik if doing lasso (probably have other ways?)

MLV integration

We intend to integrate xyzviewer with MLV. Notes on possibilities/progress/etc

experimental status

We have a first bodged integration that requires some 'unnatural' steps, but at last allows experimentation.

1. launch Chrome (or other) avoiding web security (CORS), eg
   a. **`chrome --disable-web-security --user-data-dir=%temp%\mlvxyz`**
   b. Do NOT use that Chrome instance for running your 'normal' browser tasks
   c. It should launch separately from your standard Chrome, you can run both in parallel
   d. **`"%LOCALAPPDATA%\Google\Chrome SxS\Application\chrome.exe" --disable-web-security --user-data-dir=%temp%\mlvxyz`**
2. open developer tools
3. run MLV, e.g. https://mlv.molbiol.ox.ac.uk/projects/multi_locus_view/1590
   a. I've only run it on that example, but it should be generic?
4. just once (it will be sticky between sessions as long as developer tools is open before MLV run)
   a. open graphs.js (? graphs.js?382e). ctrl-P, and 'graph' should find it
   b. on the line after 'class WGLScatterPlot extends WGLChart{' and let 'self = this' (?line 1986)
   c. add conditional breakpoint as below
   d.
```
loc = 'https://csynth.molbiol.ox.ac.uk/csynthstatic/xyz/';
import(loc + 'patchxyz.js').then(x => x.register(loc, self));
console.log('importing', div)
```

        e. once that breakpoint is added, restart MLV
5. select a scatterplot and click '3' key
    a. this will convert it to xyzviewer graph
    b. X, Y and COL taken from scatter-plot, Z fairly arbitrary for now
    c. double-click on xyzviewer graph to toggle xyzviewer gui
        - xyz currently only sees aliased field names (such as field33), not 'user' ones
    d. **This can only be performed once in each session …**
6. cross-filtering works both ways,
7. colour by works (field name only), on initial setup and on subsequent MLV menu change

summary of xyzviewer changes made

These changes will be needed for long term implementation, though details will need to change.

- patchxyz.js created for temporary bridge MLV=>xyzviewer
    - quite a few comments in patchxyz.js
- added xyz.useJson() to allow MLV to make data available to xyzviewer
    - xyz makes its own internal column based copy, as it does for reading CSV files
- modified so it can place its canvas under host dom element
    - resize etc mostly working
- added ability to read back lasso data (filter) as callbacks, and send to MLV (getCallbacks())
- added ability to accept filter from other charts (xys.ids = filter_from_outside)
- extract parts of xyz.html for xyzviewer gui under MLV
    - middle term will more cleanly separate standalone xyz gui from
    - longer term we may not need xyz gui at all when run inside MLV
- remove dependency on columns x,y,z existing

possible short term future (after 24/01/2021) changes

1. xyz currently only sees aliased field names (such as field33), not 'user' ones
    a. not sure where (or why) the aliasing is done.
2. If we host a version of xyzviewer under the same domain name as MLV we do not need step 1 above.
    a. avoids CORS issues
3. Add patch in WGLScatterPlot similar to breakpoint above
    a. maybe conditional on some detail such as 'XYZ' in location.search()
4. Make new class WGLXYZPlot similar to WGLScatterPlot with calls into XYZ
    a. Stephen to define interface (along the lines suggested by Martin)
    b. clearly correct longer term solution
    c. gradually add MLV-style GUI for more complete xyzviewer use
    d. phase out need for xyzviewer's own gui in this context

Currently (28/01/2021) working towards #4.

known issues

Most of these issues show up as uminplemented purple in the next section
- colour schema/ranges not passed/used on setColor()
- some errors in implementation of 'hide' and mixing with 'Make Opaque' (in MLV)
- only 'hide' implemented in xyz version
    - need to know function to implement to choose Hide/Make Opaque
- ranges on X,Y,Z not passed/used correctly
- multiple lasso needs cleaning up in xyzviewer
- more work needed on multiple instances of xyzviewer, and view/data relationship

xyzviewer pending API

Here is a tentative xyzviewer API for use by MLV.
purple indicates suggested, black for implemented

- **setJSON(jsondata[, fieldMap])**
  - set data from json object
  - **fieldMap gives** information about the MLV mapping from real field name to field33 etc
    - dictionary from name in jsondata to real user name
  - jsondata short term use, will be replaced by ? depending how MLV decides to store data
- **setField(id, fieldName, [low], [high],[log])**
  - todo: add update parameter (or explicit redraw())
  - **id** is X, Y, Z, R, G, B, integer, (maybe COL)
    - integer for multidimensional (probably not used in early MLV integration?)
  - **fieldName** is name of column, or a constant value, or undefined (use current)
  - fieldName may be able to be an expression in some cases
  - **low, high** give range
  - **log** is true, false, or function
- **getField(id)**
  - returns {fieldName, low, high, log}
- **setColor(fieldName, colourSetup)**
  - colourSetup includes schema for single field colour, and range, details tbc
- **getColor()**
- **setPointSize(pointsize)**
  - ? POINTSIZE may become a field id, and this will become redundant
- **getPointSize(pointsize)**
- **setFilterStyle(?)**
  - ? see gui comments below
- **getFilterStyle()**
- **hide(ids)**
  - hides the given ids, calls through to _hide
- **filter(ids)**
  - calls through to _filter,
    not sure what it does or under what circumstances it is called
- **onFilter(filterCallback)**
  - filterCallback is (ids) =>...
- **setHostDOM(domElement)**
  - eg currently the host is <div class="grid-stack-item-content" id="filter-chart-...."
  - MLV provides the host under which xyzviewer will create a canvas
- **getHostDOM()**
- **? camera control**
  - xyzviewer currently always uses perspective camera, MLV may often prefer orthographic
- **setSize(w, h)**
  - set size of the gl/xyzviewer canvas
- **setSize(pair)**
  - pair is [w, h]
- **getSize(w, h)**
  - returns [w,h]
- **setBackground(r = 0, g = r, b = r, a = 1)**
- **setVR(bool)**
  - display this xyzviewer panel in VR
- **getVR()**
- **captureImage(fid, [xsize, ysize])**
- **dispose()**

    - API implemented but as dummy
- **? axes**
    - for now xyzviewer doesn't show axes
    - SVG style axes won't work well? Great if they could

limitations of API/implementation

xyzviewer does not support multiple instances of 'charts',
I think that will involve a (significant) implementation change to xyzviewer implementation.
and not much (if any) to the API?

MLV gui comments

some comments on my experience with MLV gui

- I'm confused by how filtered points is working at the moment
    - ? want separate choice of how MY filter is displayed and how OTHER filter is displayed
    - does it hold separate 'hide' and 'make opaque' filters on each graph?
    - aside: xyzviewer can use multiple lasso filters on same data (taken from different views)
- scatter plot, SHOW does something odd to pointsize
- scatter plot, clicking and unclick log scale does odd things (and raises exception)
- I'd expect to be able to specify range on Color By in the same way as range on X, Y
- more hover help on settings panel (and elsewhere?) would be helpful
- png captured panels should probably be bigger
- would like key (F10?) to toggle each pane to normal position or full window
- I'd like the colorBy dropbox to take effect as soon as an item selected (no need for OK)
    - that would make it easier to try out different fields just using up/down cursor keys.

general comments

Here are some thoughts on issues involved.

- LMV currently uses internal JSON style object to hold data
    - that could be improved by column style lazy load of typed arrays (as discussed)
    - **to simplify/phase migration a Proxy could provide the old `data[id][field]` interface**
        - then only the loading code should need change?
        - the rest of the code could be changed later for better efficiency if necessary
- xyzviewer currently sets up a game loop style render
    - may want render on demand (eg after data/parameter change, rotation, etc)
- xyzviewer is designed for one or more datasets imposed on a **single canvas**
    - if we want multiple xyzviewer instances in MLV we need to decide how to use multiple canvas
    - ? have single three.js canvas/renderer, and copy result image each frame
    - multiple three.js canvas/renderer; then decide how to share eg DataTexture between them
- xyzviewer does not provide any axes, just 'raw' data
- tidy up xyzviewer use of multiple lassos, and exploit sensibly in MLV
- align xyzviewer's way of scaling data with MLV's
- xyzviewer needs to understand MLV colour schema
- we can easily present xyzviewer pane in VR

references

(very biased set)

- [1] PRTV, 1975, Peterlee Relational Test Vehicle. Relational query system.
  First serious relational query system to handle any amount of data

65k records: trivial by today's standards

- [2] WGS, 1983, Winchester Graphics System. PRTV + 3d graphics + bridge
  Used for molecular and archaeology data exploration (= data manipulation + data visualization)

The column based storage approach is very close to separate work by Ray Lorie, Norman Winterbottom and Michel Adiba in the 1970s.