Mastodon → Blender/SciView

NG protocol: context, thoughts, plans

Follow us here: https://imagesc.zulipchat.com/#narrow/stream/327470-Mastodon/topic/Blender

v0.1: May 12, 2022, written by Vladimir Ulman
v0.x: you’re welcome here!

Some notes from my early days with SciView and SimViewer

- I’m operating with the notion of **Source** and **Sink**
- Both operate with a set of **objects**

- **Source** – holds a set of **all original objects**
- **Sink** – holds **own display objects** requested to be displayed
    - Two **independent sets** of objects, the only **link** between them was an **ID attribute**
    - One original object may have associated multiple display objects
    - There’s not necessarily a pure one-to-one mapping
    - I used to use **categories** of displayed objects
        - Shape-driven (spheres, lines, vectors)
        - Function-driven
            - Normal local content – a placeholder display of a cell, e.g. Sphere
            - Debug local content – aux display related to the cell, e.g. a flag or vector that shows e.g. movement vector, trajectory
            - Normal global content – content not related to any cell, e.g. a frame of a scene
            - Debug global content – similar to the normal gl. Content
        - User at Sink could show/hide various categories
        - Source submitted all display requests, user at Sink filtered what to display
        - Source could decide not to send everything to save communication bandwidth
        - Amount of data to be displayed was thus controlled both at Source and Sink

- Message: I like that user at the Sink have additional own control
- SciView didn’t have a notion of time → Was showing only the *current* content (but I believe the notion of time can be introduced)

Some notes from my early days with SciView and SimViewer → Consequences & Wishes

- **Source** – holds a set of **all original objects**
- **Sink** – holds **own display objects** requested to be displayed
- Two **independent sets** of objects, the only **link** between them was an **ID attribute**
- I used to use **categories** of displayed objects
    - Shape-driven (spheres, lines, vectors) — performance gain – instancing
    - Function-driven — another resolution axis – finer control when aux data is displayed
- New thing: **Time attribute** — Blender has time axis, we can have a “history bag”
    - Moving along time axis in Blender is really user comfortable (one keypress! ...or with a mouse)
    - Can show development of the content
    - Can show “versions” of the same data
        - It’s an abuse of “time” semantics → widens the semantics and takes-in the “function-driven” aspect from categories
        - Deeping the abuse: Time as a string! → **Visibility containers** (aliased to Buckets, see next slide)
        - Buckets… the abuse turned into a feature, yay!

- Source submits **display requests** to the Sink — (show these 5 spheres)
- Sink may send back **update requests** and **event requests** — (update position of this sphere, focus on that sphere)
- The requests act on a displayed object — (either it defines it, or reports an activity on it)

- Within a Sink, every displayed object should know its:
    - **Source** → for a feedback line, facilitates user’s filtering ability
        - **Bucket** → semantics grouping, one (and only) additional level to facilitate user’s filtering ability
            - Category → technical necessity, should be renamed to (Reference) **Shape**

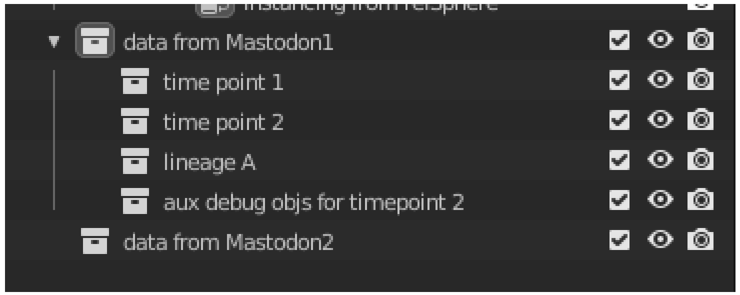
“Buckets” at Blender side

- Technically it is a Collection
- There can be a **hiearchy of Buckets** (Collections)
  - Two levels: 1st level list of sources, 2nd level list of buckets --------→
  - Two levels: **Source level**, **Bucket level**

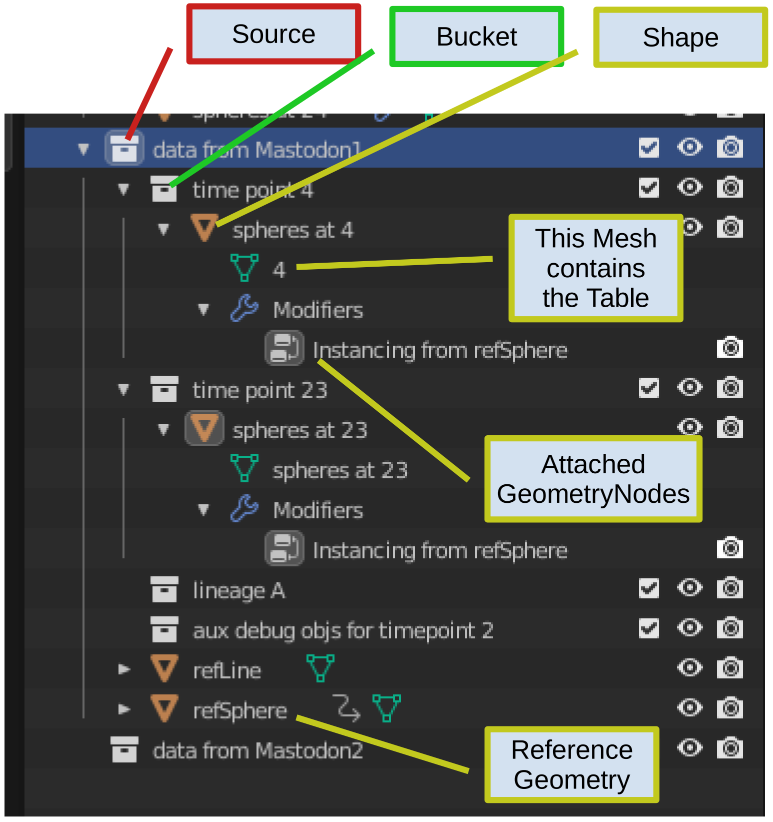
- Bucket holds objects from one source (e.g., Mastodon, EG)
  - … that shall be visible only at exactly one frame (integer number)
  - … **or** that shall be visible always, at any frame (integer -1)
    - Possible extension: Interval? Easy for V.U. atm.
    - Possible extension: Set? Hard for V.U. atm.
  - … **and** that shall be semantically related (only for the user filtering comfort)

- Every Bucket has own **identification** name (string), e.g. “projectNick TP = 4”
  - Identification must be unique within its Source level

- Bucket may contain **Shapes**
  - These Shapes are “from” the same Source
    and belong semantically together
    and are thus visible under the same condition
  - A Shape Blender object is technically… well… a **Table**
    - One row is one position/occurence of the Shape
    - Columns are position, color, Shape related extras, etc.
    - Blender displays (owing to us-provided Geometry Nodes)
      - One instance of the Shape for every row in that table
      - At the position
      - The Shape is possibly further adjusted as a result of
        - The Geometry Nodes pipeline
        - Shape related extras attributes in the table



```
▾ data from Mastodon1
    time point 1
    time point 2
    lineage A
    aux debug objs for timepoint 2
  data from Mastodon2
```



```
▾ data from Mastodon1
    ▾ time point 4
        ▾ spheres at 4
            4
            ▾ Modifiers
                Instancing from refSphere
    ▾ time point 23
        ▾ spheres at 23
            spheres at 23
            ▾ Modifiers
                Instancing from refSphere
      lineage A
      aux debug objs for timepoint 2
    ▸ refLine
    ▸ refSphere
  data from Mastodon2
```

“Buckets” at Blender side

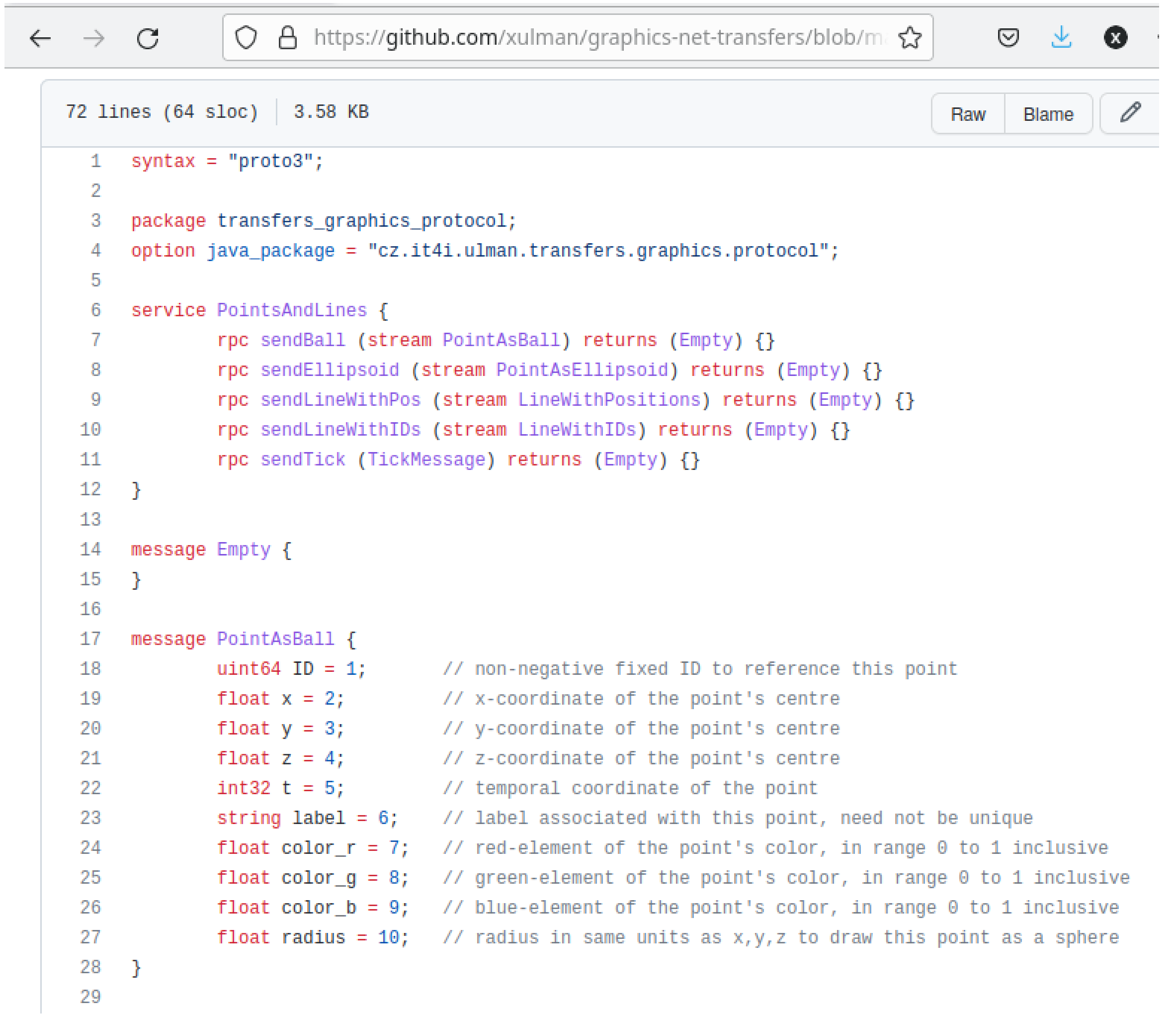
- An arriving object to be displayed should provide:
- Source (IP + showName string,
  this provides the underlying messaging subsystem)
- **Bucket** to which it belongs (within the Source): String
- Reference shape: Should be enum
  - Let’s encode this into the name of the message
  - DisplaySphere(…), DisplayLine(…)
- **Params** of that shape (e.g. centre coordinate, radius)
- **ID** of this under which the Source will recognize it

- The Source and Shape are in fact implicit parameters
- Message needs to contain only the data in bold

- **If this layout is Okay, we could move to discussing the communication scheme**
- Examples:

- How to communicate Bucket, Colors (color palletes)?
- Define next timepoint always from scratch, or as an update from some other timepoint, or allow both?
- Delete obj. messages yes/no?

Example protocol to be improved:



https://github.com/xulman/graphics-net-transfers/blob/m

72 lines (64 sloc) | 3.58 KB

```
syntax = "proto3";

package transfers_graphics_protocol;
option java_package = "cz.it4i.ulman.transfers.graphics.protocol";

service PointsAndLines {
        rpc sendBall (stream PointAsBall) returns (Empty) {}
        rpc sendEllipsoid (stream PointAsEllipsoid) returns (Empty) {}
        rpc sendLineWithPos (stream LineWithPositions) returns (Empty) {}
        rpc sendLineWithIDs (stream LineWithIDs) returns (Empty) {}
        rpc sendTick (TickMessage) returns (Empty) {}
}

message Empty {
}

message PointAsBall {
        uint64 ID = 1;       // non-negative fixed ID to reference this point
        float x = 2;         // x-coordinate of the point's centre
        float y = 3;         // y-coordinate of the point's centre
        float z = 4;         // z-coordinate of the point's centre
        int32 t = 5;         // temporal coordinate of the point
        string label = 6;    // label associated with this point, need not be unique
        float color_r = 7;   // red-element of the point's color, in range 0 to 1 inclusive
        float color_g = 8;   // green-element of the point's color, in range 0 to 1 inclusive
        float color_b = 9;   // blue-element of the point's color, in range 0 to 1 inclusive
        float radius = 10;   // radius in same units as x,y,z to draw this point as a sphere
}

```