Introduction to Web-based Particle Visualizations

Aaron M. Geller

Senior Data Visualization Specialist
Research Computing and Data Services

Research Associate Professor
Center for Interdisciplinary Exploration and Research in Astrophysics (CIERA) and Department of Physics & Astronomy

CIERA CELEBRATING 15 YEARS

Northwestern
INFORMATION TECHNOLOGY
RESEARCH COMPUTING AND DATA SERVICES

Motivation

For interactivity

- Interrogate a data set in real time
- Build intuition about your data
- Enable serendipitous discovery
- Can be very engaging for the public

For using the web

- OS agnostic (just need a browser)
- Easy to share online
- Many existing libraries for UI/X and real-time interactions
- “Endlessly” customizable

Introduction to three.js

Main components of a three.js interactive:

scene object containing all the different items that you want to draw (`THREE.Scene`)

renderer draws the scene onto your computer screen (`THREE.WebGLRenderer`)

camera sets the viewing position and angle (`THREE.PerspectiveCamera`)

controls allows users to move the camera around with the mouse and/or keyboard (`THREE.TrackballControls`)

Introduction to three.js

My usual code layout:

1. Read in some data file (if relevant), using [d3.js]
2. Initialize the scene, renderer, camera, controls
3. Initialize a gui if needed (e.g., dat.gui: https://github.com/dataarts/dat.gui)
4. Draw each item (i.e., each mesh) and add them to the scene
   - A mesh consists of a geometry and a material
5. Start the animation loop
   - Checks for any updates from the controls, keyboard, etc.
   - Redraws scene in your browser each refresh time (typically 60 times per second)
   - Even if you don’t change anything in the scene, this is still running the background

Introduction to three.js

Mesh Objects:

- Geometries:
  - Geometries are defined by x,y,z vertices that combine to draw triangles.
  - Geometries define the shape of your object .
  - Three.js has many different 3D polygons (e.g., `THREE.SphereGeometry`, `THREE.BoxGeometry`, etc.).
  - You can also construct 2D shape (`THREE.Shape`) .
  - You can also build your own custom 3D shapes by specifying vertices, or extruding from a shape, etc.
- Materials:
  - Materials define the look of the object (e.g., the color, shininess, texture, etc.).
  - Three.js has many different materials, each with many different options to choose from. The most basic is `THREE.MeshBasicMaterial`.
  - One particularly useful for us: if you want to plot a bunch of points in 3D space, you can use a point cloud method (`THREE.PointsMaterial`).
  - You can also define your own custom "shaders" to further manipulate the look of each geometry.
  - You can apply a "texture" (i.e., an image) to a given geometry via the material.

simple_cube_example

index.html

```
<html>
    <head>
        <title>My first three.js app</title>
        <style>
            body {margin:0;}
            canvas {width: 100%; height: 100%;}
        </style>

        <script type="importmap">
            {
                "imports": {
                    "three": "https://cdn.jsdelivr.net/npm/three@v0.169.0/build/three.module.js",
                    "three/addons/": "https://cdn.jsdelivr.net/npm/three@v0.169.0/examples/jsm/"
                }
            }
        </script>

    </head>
    <body>
        <script type="module" src="/main.js"></script>
    </body>
</html>
```

Set up the webpage with some styles. Make the WebGL canvas fill the screen.

Link to the three.js library.

Link to the script module.

simple_cube_example

main.js

```
import * as THREE from 'three';

var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1,  1000 );
camera.position.z = 5;

var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild( renderer.domElement );

var geometry = new THREE.BoxGeometry(1, 1, 1);
var material = new THREE.MeshBasicMaterial({color: 0x00ff00} );
var cube = new THREE.Mesh( geometry, material);

scene.add( cube );

var animate = function () {
    requestAnimationFrame( animate );
    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;
    renderer.render( scene, camera );
};

animate();
```

Import the THREE library.

Define the scene, camera and renderer.

Define the geometry and material, and add it to the scene.

This animation loop rotates the cube.

simple_cube_example

To run on your machine:

$ cd <directory with `simple_cube_example/` files>

```
$ python -m http.server
```

Then point your browser to http://localhost:8000/

simple_particle_example

This demo uses (outdated) supernovae detections. We'll look at the code together in VS Code. (It's OK if you don't understand everything!)

To run on your machine:

```
$ cd <directory with simple_particle_example/ files>
$ python -m http.server
```

Then point your browser to http://localhost:8000/

Firefly

https://firefly.rcs.northwestern.edu/ *,*
https://ui.adsabs.harvard.edu/abs/2023ApJS..265...38G/abstract

- Browser-based interactive particle viewer, capable of rendering millions of points in real-time (and to explore larger datasets with octree)
- Javascript frontend (three.js, d3.js, etc.)
- Python backend (Flask)
- Additional Python tools to format data for Firefly
- Built primarily by Alex Gurvich and Aaron Geller (mostly pre-2021)
- Example notebooks here : https://github.com/ageller/Firefly/tree/main/src/firefly/ntbks

Firefly:

A WebGL tool to explore particle-based data

Aaron Geller / Alex Gurvich / Northwestern

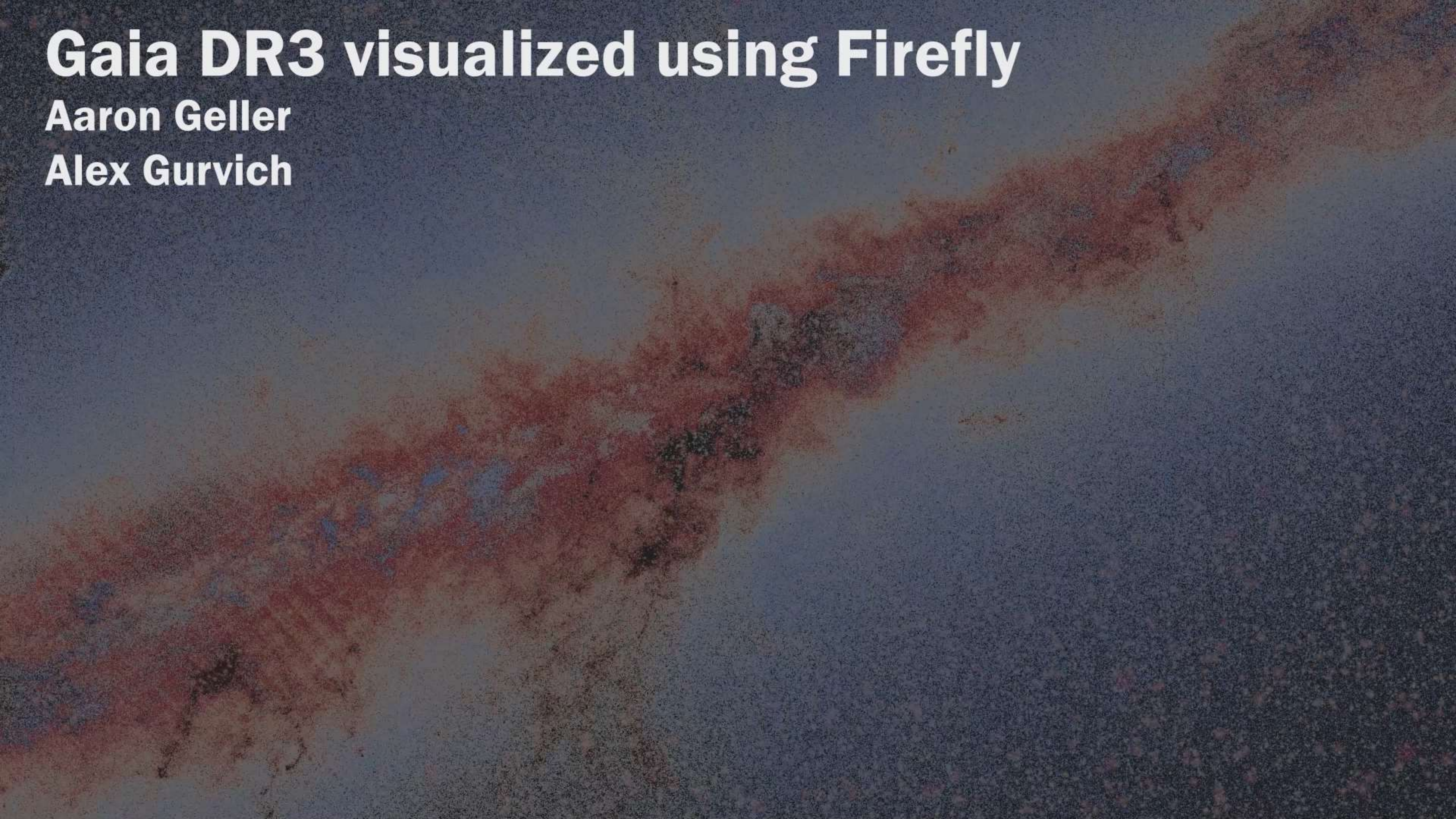
Instructions:

- Right-click and drag with the mouse to rotate your view.
- Use the mouse wheel to zoom in and out.
- Click the Controls bar on the top left to show/hide a user interface.
- Detailed instructions can be found on the Firefly GitHub page.
- **h**: toggles this help screen on and off.

Click to begin.

Gaia DR3 visualized using Firefly

Aaron Geller
Alex Gurvich



File Edit View Insert Cell Kernel Widgets Help

Notebook saved Trusted Python 3

Run Code

Try looking at this in Firefly

In [29]:
```
# some_file.py
import sys
sys.path.insert(0, '/Users/ageller/Visualizations/Firefly/dataReader')

import pandas as pd
from simpleReader import simpleReader
import requests
from IPython.display import IFrame
```

Start the server

In [30]:
```
%%bash --bg --proc bg_proc
python /Users/ageller/Visualizations/Firefly/FireflyFlaskApp.py
```

Show Firefly in an IFrame

In [ ]:
```
url = "http://localhost:5000/combined"
IFrame(url, width=1650, height=700)
```

Format the data and send it to Firefly

In [32]:
```
mx, my, mz = np.mean(resampleX), np.mean(resampleY), np.mean(resampleZ)
sx, sy, sz = np.std(resampleX - mx), np.std(resampleY - my), np.std(resampleZ - mz)
s = max([sx,sy,sz])
coords = np.array([(resampleX - mx)/s, (resampleY - my)/s, (resampleZ - mz)/s]).T
print(coords.shape)

name = "part1"
outDict = simpleReader(name, coords, [1,1,0,0.7], 1, colorArray=resampleRGBA)
print(outDict['parts'][name]['Coordinates'].shape)
```

Firefly

Time for you to try on your own machine.

I recommend creating a conda environment for firefly:

```
conda create --name firefly-env python=3.11 jupyter numpy=1.26.4
conda activate firefly-env
pip install firefly
```

We'll start with `minimal_firefly_example.ipynb`

Questions?

Exercise

1. Download/generate/use your own data that has x,y,z and other attributes
    - *Suggestion*: Gaia data (e.g., for a galactic open cluster)
2. Ingest this into Firefly and explore it in a Jupyter notebook
    - Make sure to include attributes to color/filter by
3. Use the (new) data selection tool to select a portion of your data using Firefly
4. Create a plot of these selected data using Python.
5. Share the result with us!