Start: 7:40 p.m.

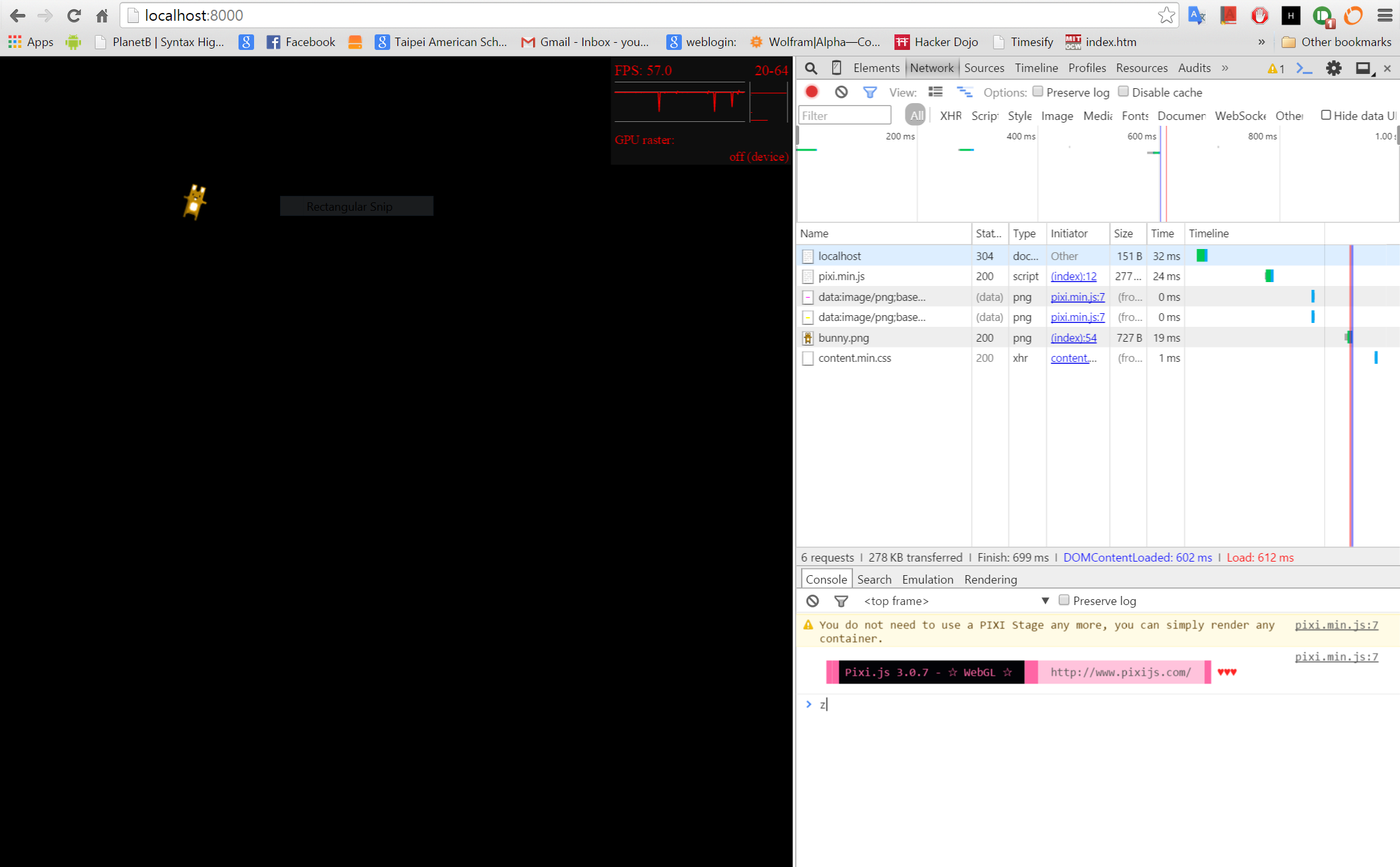
Tasks: Set up javascript to read from shapefile and render pixi graphics objects. Also, verify that all the state:ids are in the correct order while reading. (Note: since the order that shapes/records are read in is important for shapefiles, they should be the same in both javascript and python. Just double-checking)

Setting up node web server (I forgot js was restricted from local files for safety reasons).

Break: 8:00 p.m.

Resume: 9:00 p.m.

9:25 webapp is up and running! Currently serving a simple pixi js demo



Looking at js shapefile libraries

<https://github.com/calvinmetcalf/shapefile-js>

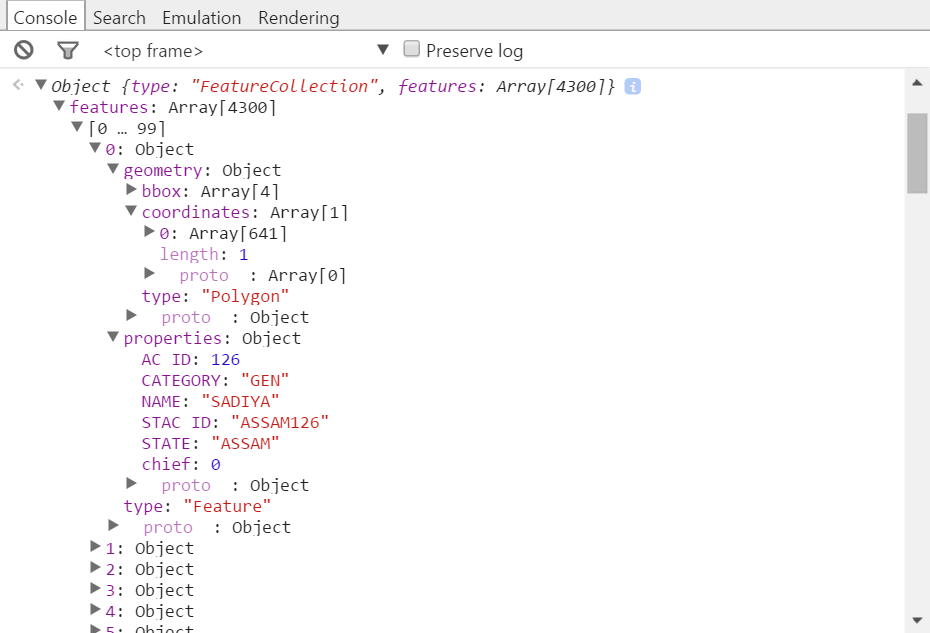
<https://github.com/wavded/js-shapefile-to-geojson>

<https://github.com/mbostock/shapefile>

Decided to go with calvinmertcalf’s shapefile-js, as it is the most recently updated. The others haven’t seen any updates in a year or two. All three of them are somewhat in-progress.

shp.min.js taken from a link on one of the demos: <http://calvinmetcalf.github.io/shapefile-js/dist/shp.min.js> as the current shapefile-js is being worked on.

Read the india shapefile; output looks like this:



It’s an array of objects. Each object has two fields ‘geometry’ and ‘properties’. Geometry has a bounding box and a ‘coordinates’ array. The first entry in the ‘coordinates’ array holds *another* array, this one with the coordinate pairs used to draw the polygon. I believe ‘coordinates’ can hold multiple arrays of coordinate pairs for more complex geographical features. We’re only using polygons, however, so we don’t have to worry about it. Something that might be an issue is the sheer number of data points for each assembly/constituency. ASSAM:126 has 641 points. I don’t really think we need that much detail. In case performance issues arise, two solutions come to mind:

1. Take every nth point and discard the rest. This will generally keep the correct shape, but if sharp corners or vertices are discarded it could visibly alter the polygon
2. Somewhat more complex. Keep track of the slope of the previous line segment that was drawn. Keep track of the new slope if the next set of two points (p1 and p2) were to be drawn. If the slope varies over a certain threshold, plot p1 and remember the p1 -> p2 slope and continue, with p1 = p2 and p2 = next point. By tuning the threshold, we can discard points that are more or less on a straight line, thus reduce memory size while keeping sharp corners/vertices.

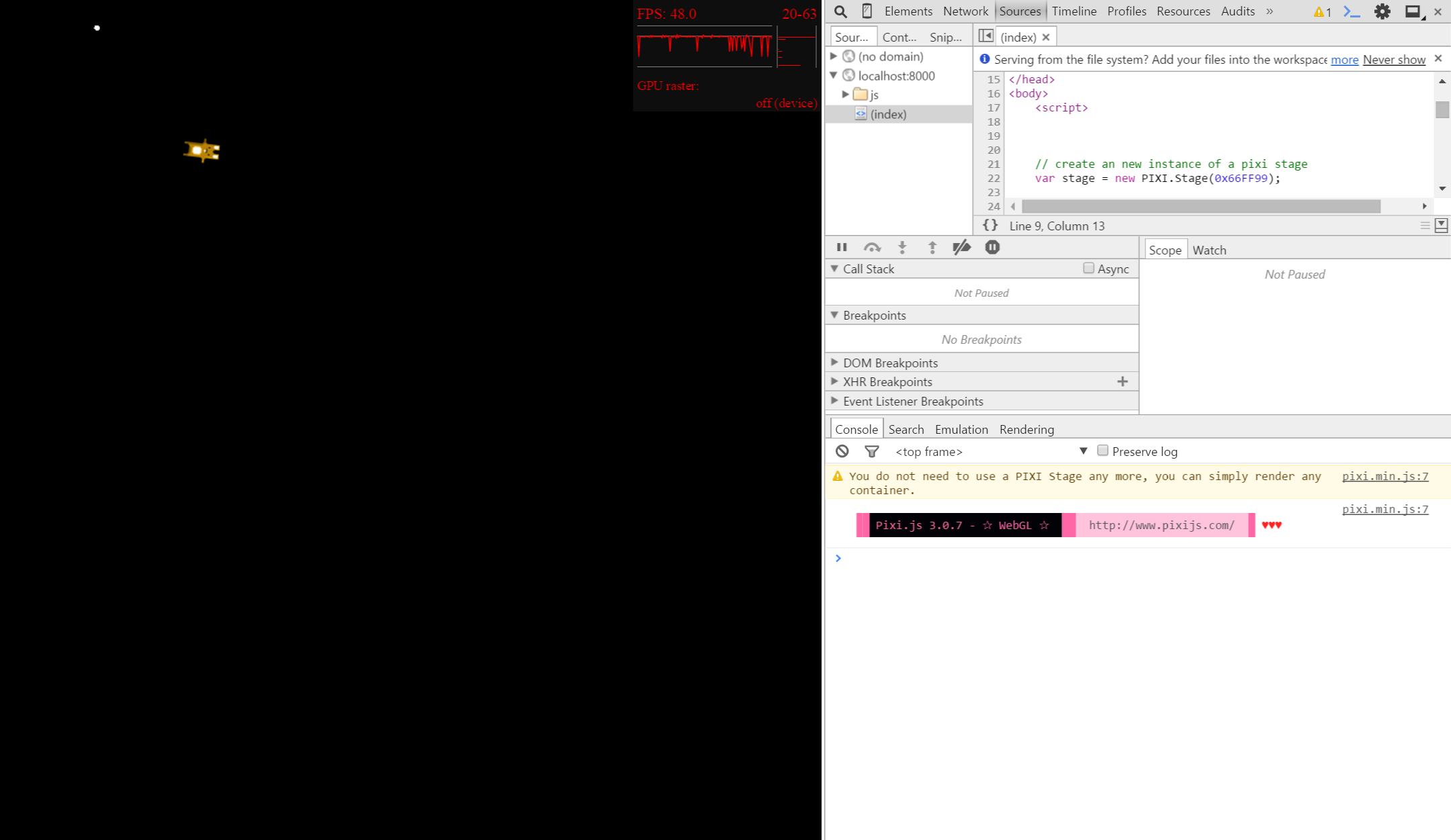
Break: 10:10 p.m.

Resume: 1:20 a.m.

Next up, I’m going to draw the first assembly/constituency (ASSAM:126) using PIXI graphics calls.

Looking at this example: <http://www.goodboydigital.com/pixijs/examples/13/>

2:16 a.m. ASSAM:126 has been rendered! It’s tiny, but it’s there!

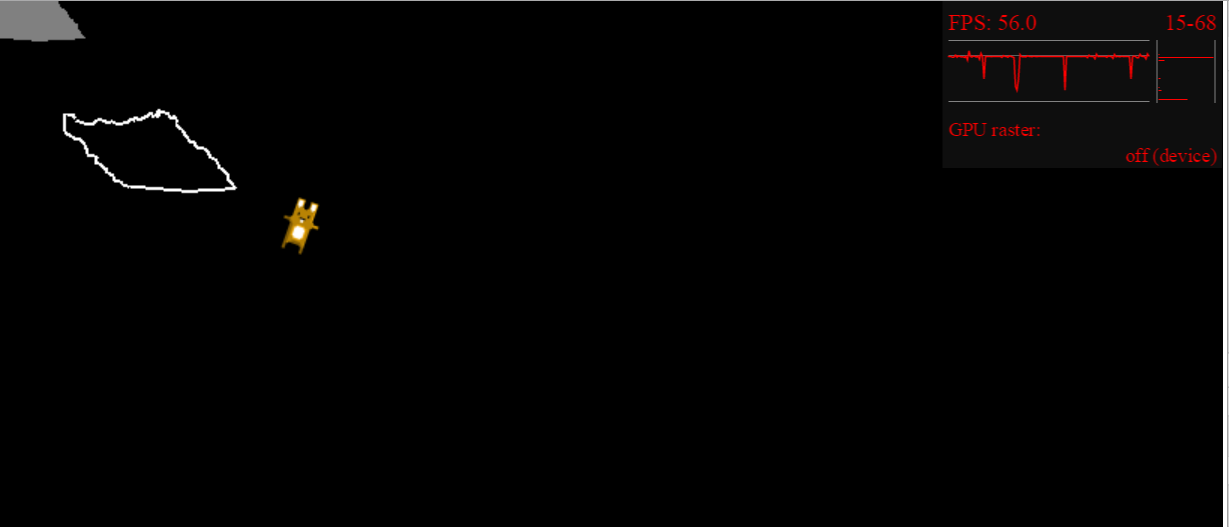


Current code:

1. // load shapefile
2. shp("/data/AC\_India\_Geo/AC\_ALLINDIA").then(**function**(geojson){
3. **var** test\_shape = **new** PIXI.Graphics();
4. test\_shape.beginFill(0xFFFFFF, 0.5);
5. test\_shape.lineStyle(5, 0xFFFFFF, 1);
6. **var** test\_points = geojson.features[0].geometry.coordinates[0];
7. **var** test\_box = geojson.features[0].geometry.bbox;
9. test\_shape.moveTo(test\_points[0][0], test\_points[0][1]);
10. **for**(**var** i = 1; i < test\_points.length; ++i)
11. {
12. test\_shape.lineTo(test\_points[i][0], test\_points[i][1]);
13. }
14. test\_shape.moveTo(test\_points[0][0], test\_points[0][1]);
15. test\_shape.endFill();
16. stage.addChild(test\_shape);
17. });

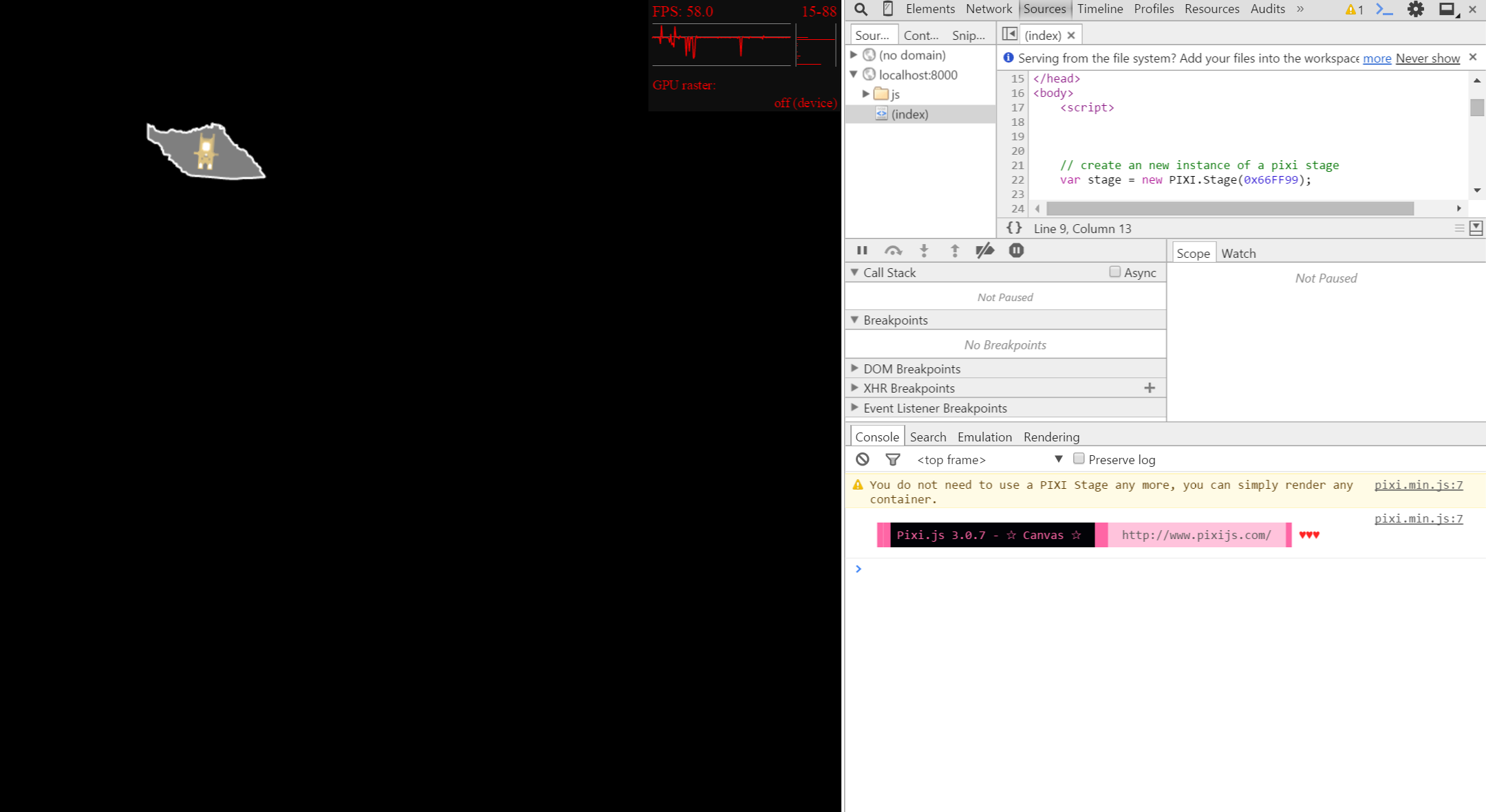
(Inserted into the basic pixi js tutorial here: <http://www.goodboydigital.com/pixijs/examples/1/> )

After some tweaking (scaling, mostly), assam 126 now looks like this:



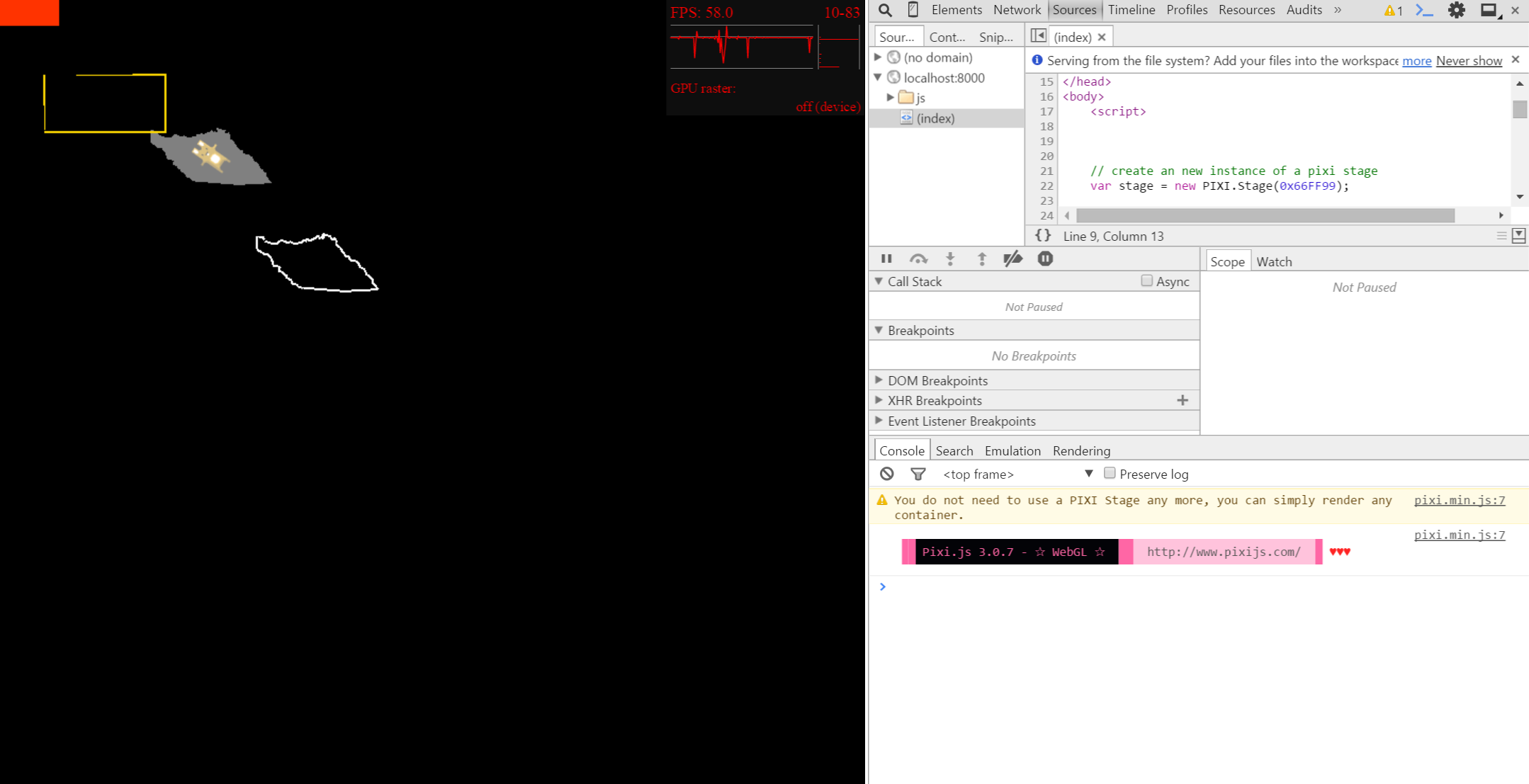
The fill doesn’t match up with the line boundaries.

Fixed! Turns out there’s something wrong with the webGL renderer. Switched to canvas renderer. Might cause performance issues, but for now assam 126 is being drawn correctly from a shape file in JavaScript!



Same issue shows up in firefox. However, rendering simpler shapes in the webgl renderer seems to work fine!

Maybe try rendering the bounding box and see if the issue is because of the data/webgl or because of the massive amount of data points. Issue is definitely due to the large amount of scaling. The bounding box also experiences the same effect when drawn.



Scaling all of the points before drawing lines/fill should fix this issue.

Finish: 3:40 a.m.

Total: 3.8 hrs