For The University of Nebraska-Helikar Lab

GSOC PROPOSAL

2015

This proposal is based on my understanding of your project and provides the details of how and why would I like to contribute towards the welfare of this project. All the information provided in the proposal is correct to the best of my knowledge and the work is my sole effort.

Efforts by Soumitra Agarwal

Guwahati,Assam,India

Phone:+917896878233
Email: a.soumitra@iitg.ernet.in, agarwalsoumitra1504@gmail.com
GitHub:SoumitraAgarwal@github.com

Education

Indian Institute of Technology, Guwahati — Pursuing presently
Bachelor of Technology with Honors in Mathematics and Computing — 2014-2018

India International School, Jaipur — Percentage: 94.3%
12th Grade Examination under Central Board of Secondary Education — 2005-2014

Research Interests

Web Development, Algorithms, Machine Learning, Scientific Computing

Previous Works/Interests

Biotechnology and bioinformatics:

- Worked on a research based project on biological pollution indicators under the SCIPE module of IAPT(Indian association of physics teachers), and was ranked 1st among 10,000 individuals
- Currently pursuing a course on biotechnology at IIT Guwahati under professor Arun Goyal.
- Did a 3 year major course on biotechnology at High School
- Created a working model on animal anatomy at The Central Board of Secondary Education Science Fair

Programming and others:

- Worked on a driver control Machine Learning project on Kaggle. — *Fall* 2014
  Under MS Student: Mr.Pranav Jindal, Stanford, USA
- Created an application for developing games using simple user interface at the Microsoft Hackathon. — *Spring*2014
- Created a web-based application to improve social interaction at the IITG Alcheringa Model United Nations — *Fall* 2014

Technical Skill Set

Programming/Scripting Skills	C++, Java, Python, MATLAB, Octave, C#
Databases	MySQL
Others	HTML5, CSS3, JavaScript(including WebGL, Three.js), Git, Ruby(Rails)

Started contributing for open source software in winter 2014, and has been continuing ever since. Has contributed to societies like SciRuby ,Rails, ROS and Mozilla.

The Project

In lieu of the idea by "The Helikar Labs- University of Nebraska":

Interactive, Real-Time Dynamics Visualization Of Complex Network Graphs.

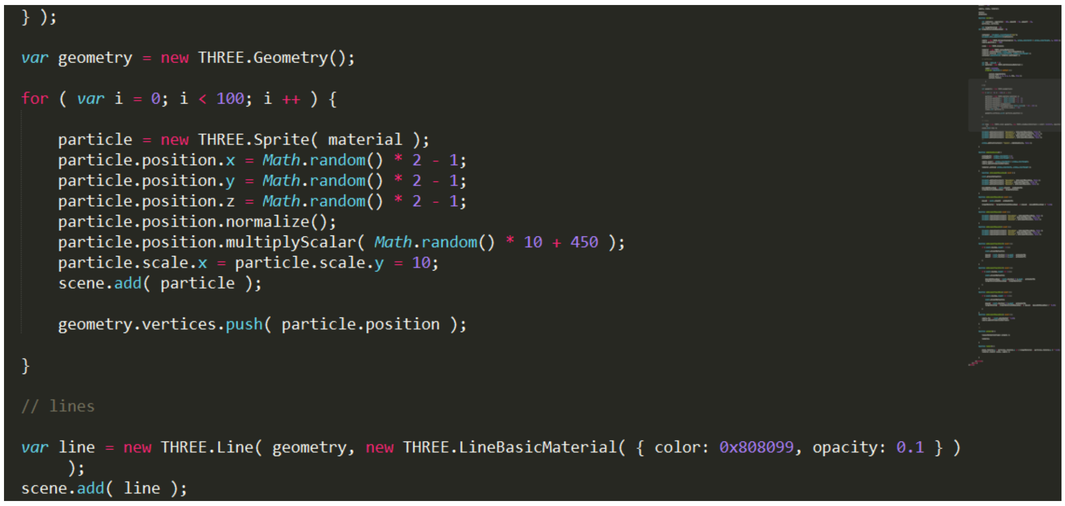
Guide to timeline: Green represents time required for given process, Red represents time gone, Blue represents time left. Assuming 40 hours per week.

I usually run my code in Chrome. The reason is that, most often, Chrome has the best support and performance for WebGL /other renderers and it has a really great JavaScript debugger. With this debugger you can quickly pinpoint problems, for instance, by using breakpoints and console output. Suggestions to edits/additions to the idea are as follows:

1.<u>Making a 3-D model with features:</u>

This would be done using the most simple features from the library three.js and using some of the common renderers. The network will be a 3-dimensional structure with interconnected network points as stated in the original idea .This would be the foundation of what we want to do.

➢ Easy addition and removal of nodes:



```
} );

var geometry = new THREE.Geometry();

for ( var i = 0; i < 100; i ++ ) {

    particle = new THREE.Sprite( material );
    particle.position.x = Math.random() * 2 - 1;
    particle.position.y = Math.random() * 2 - 1;
    particle.position.z = Math.random() * 2 - 1;
    particle.position.normalize();
    particle.position.multiplyScalar( Math.random() * 10 + 450 );
    particle.scale.x = particle.scale.y = 10;
    scene.add( particle );

    geometry.vertices.push( particle.position );

}

// lines

var line = new THREE.Line( geometry, new THREE.LineBasicMaterial( { color: 0x808099, opacity: 0.1 } )
);
scene.add( line );
```

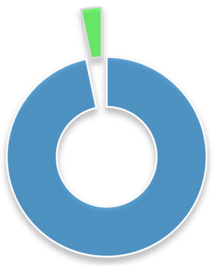
The given image shows how we may add a small number of particles to create a simple network of given data points .I will use the Geometry object from the Three.js library to store set of coordinates all points and using it later to create connections between several nodes.

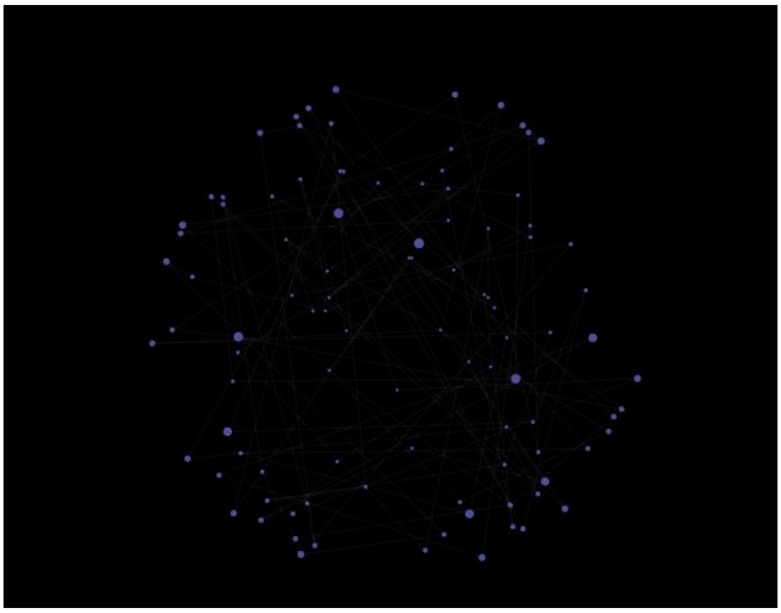
- Shifting of nodes, providing feedback on the change in data on changing the position of this node.

  I will enable shifting of nodes of a by deleting the previous particle from the geometry object and creating a new particle in its place and recreating the line geometry.

Timeline

upto 26th May 15





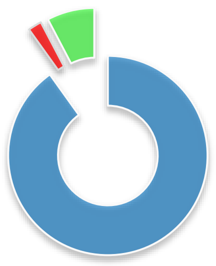
The expected foundation of our project is thus made, and now we can begin making changes to it. The expected network must be comparable to something like the one given beside.

And here we begin the real work.

2.Zooming the plot:

Timeline

upto 2nd June 15

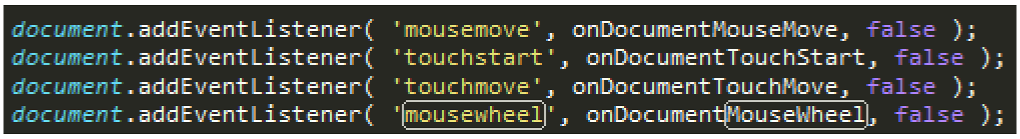


- Scroll zooming:

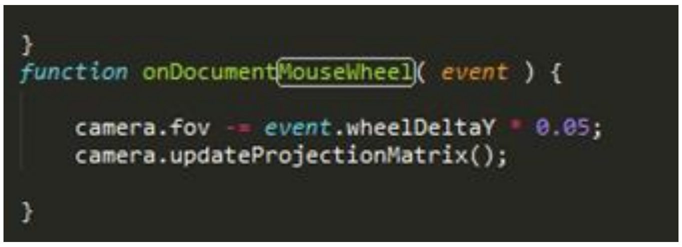
Why this?

Scroll zooming is one of the most common ways that are generally used to zoom graphics. Users tend to expect the illustration to zoom in once they scroll up and when it doesn't happen it gives them the feeling that something is missing. Thus it is utmost essential to include this feature.

How would I incorporate it?



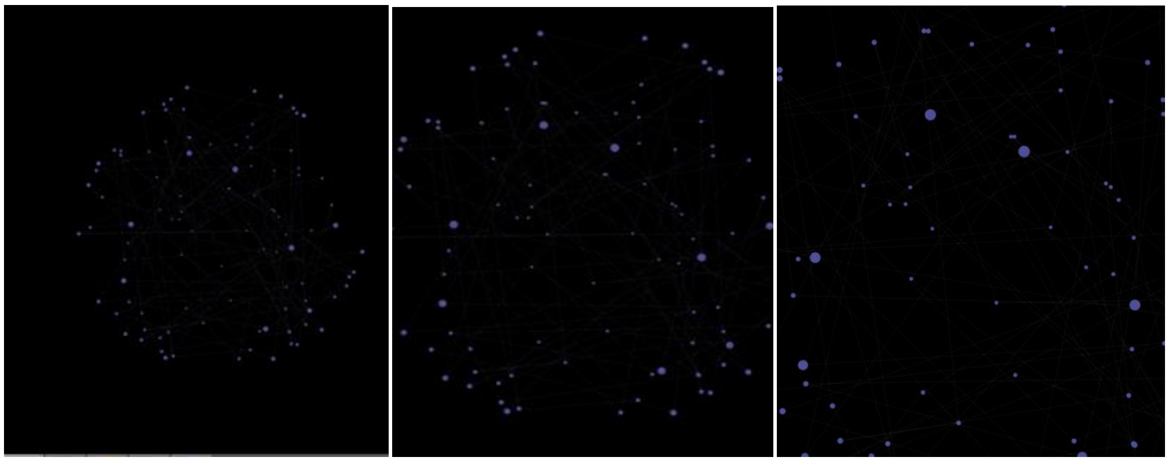
```
document.addEventListener( 'mousemove', onDocumentMouseMove, false );
document.addEventListener( 'touchstart', onDocumentTouchStart, false );
document.addEventListener( 'touchmove', onDocumentTouchMove, false );
document.addEventListener( 'mousewheel', onDocumentMouseWheel, false );
```



```
}
function onDocumentMouseWheel( event ) {

    camera.fov -= event.wheelDeltaY * 0.05;
    camera.updateProjectionMatrix();

}
```

Simple scroll zooming can be performed by using the events handler in JavaScript and render it with camera.fov or camera field of view method from the three.js library.

What would actually happen?

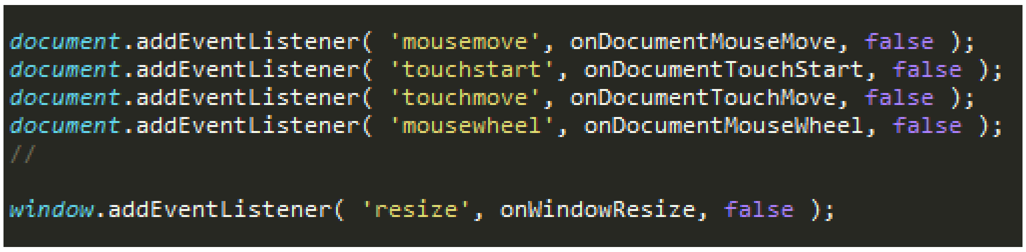


The above shown series of screenshots one after another show what we expect from our illustration. These were taken while continuously scrolling up on the mouse wheel. As this uses no binding of keys, it supports all touchpads with 2 finger scroll enabled.

- Zooming allows free movement:
  Movement is allowed while zooming. As in no binding of mouse keys with scroll keys.

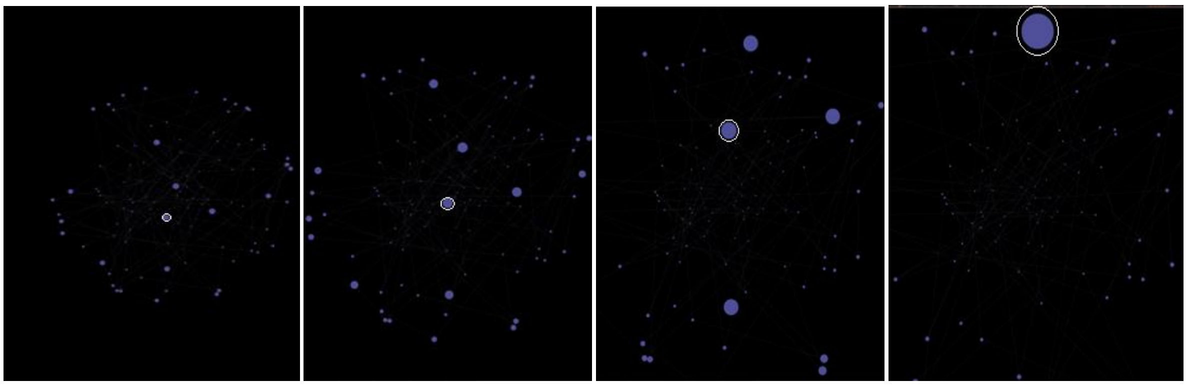
  Why this?
  This allows quick approach to the point/node concerned by simultaneously moving towards the point with zoom and movement.

  How would I incorporate it?
  By assigning zooming and movement to different event lists, like zooming may be assigned to 'onDocumentMouseWheel' whereas movement can be assigned to something like 'onDocumentMouseMove' as shown in the code.



```
document.addEventListener( 'mousemove', onDocumentMouseMove, false );
document.addEventListener( 'touchstart', onDocumentTouchStart, false );
document.addEventListener( 'touchmove', onDocumentTouchMove, false );
document.addEventListener( 'mousewheel', onDocumentMouseWheel, false );
//

window.addEventListener( 'resize', onWindowResize, false );
```

  What would actually happen?

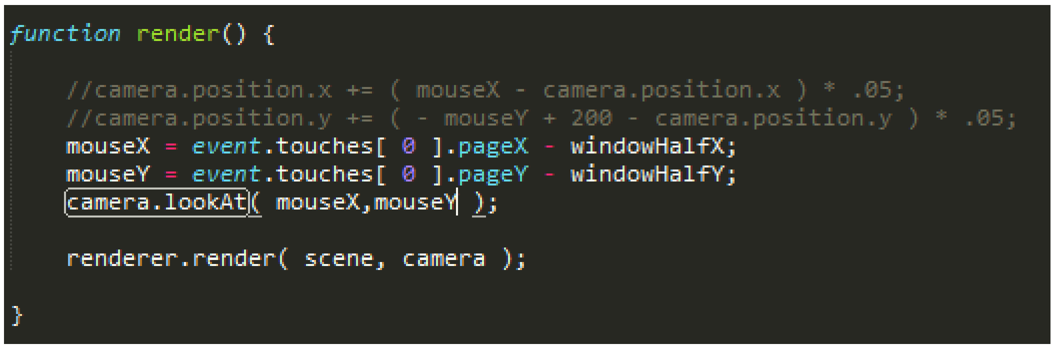


  The point in the white circle depicts the expected movement of a specific particle when it is zoomed using scroll and moved using the mouse buttons.

- Variable approach points of zoom:
  Change the location of where you want to go when you are zooming.

Why this?
This allowed detailed analysis of each and every node/ areas easier as generally we need two use 2 unbound motions of lateral movement and zooming, whereas by setting the approach point, one may directly zoom to the destination. It provides a more systematic way of studying graphs/projections of large amount of data.

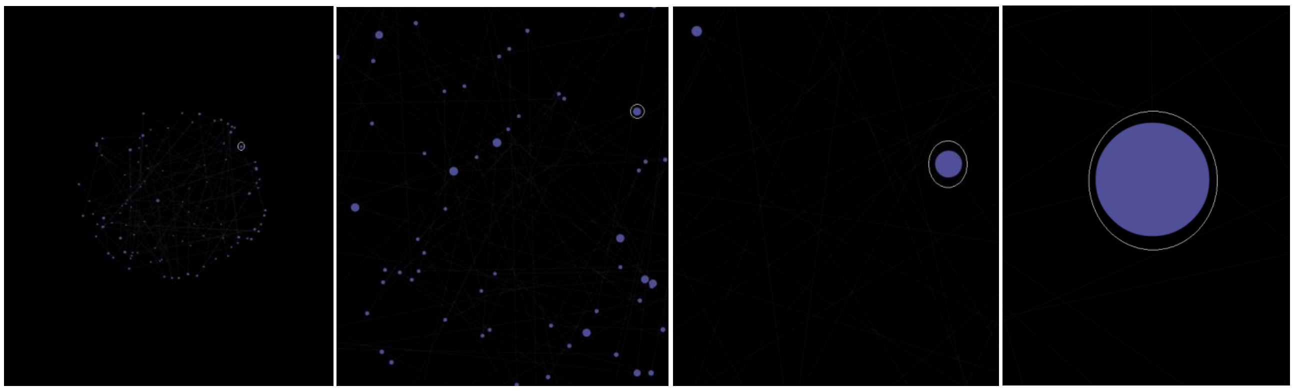
How would I incorporate it?



```
function render() {

    //camera.position.x += ( mouseX - camera.position.x ) * .05;
    //camera.position.y += ( - mouseY + 200 - camera.position.y ) * .05;
    mouseX = event.touches[ 0 ].pageX - windowHalfX;
    mouseY = event.touches[ 0 ].pageY - windowHalfY;
    camera.lookAt( mouseX,mouseY );

    renderer.render( scene, camera );

}
```

As given above, I will use the 'camera.lookAt' function while rendering the scene, and assign the value of x and y coordinates as the coordinates where mouse touched the page. Since the camera is fixed to look at that point it scroll zoom will take it only to that point.

What will actually happen?



The approach zoom enables the particle to eventually come at the centre of the screen. This might help us in easy tracking, marking/unmarking (which will come on later) and study of nodes.

Hurdles?

Presently my thinking allows to achieve this expansion, but the whole geometry rotates (collection of particles).What I wanted to achieve was to fix the geometry while using this approach, and I would try and overcome this situation.

3.Interactive rotation/panning of the plot:

As it was difficult to explain the pan using screenshots of networks, I used a cube instead for all panning examples.

Timeline

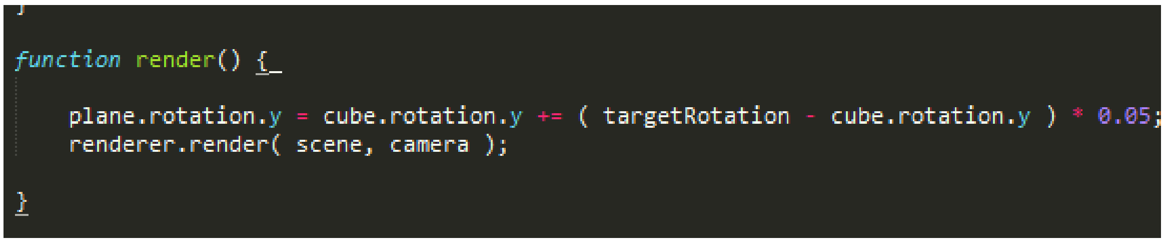
upto 13th June 15

Most of the ideas here are inspired from Google Earths panning options. It gives an realistic experience and also makes one feel that he is in control of the things.

- Free movement:
  Move in any direction/Pan in any combination of coordinates.

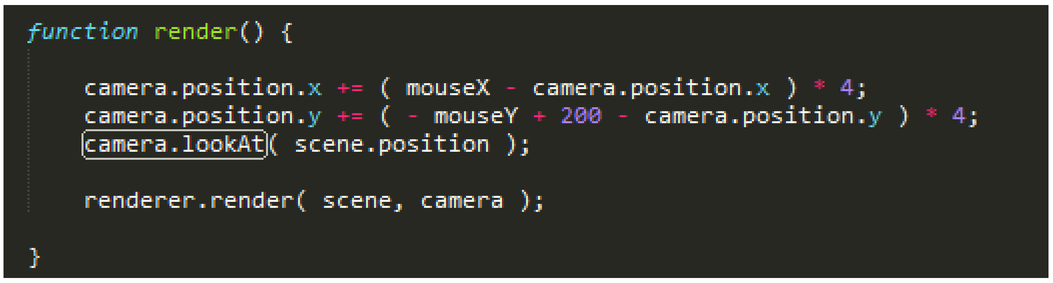
  Why this?
  Why not? It creates an ease over the control and opens up so many different streams where one may take this and sometimes it is absolutely necessary.

  How would I do it?
  For a normal pan one may use the plane to pan right and left according to the mouse movement, as:



```
function render() {

    plane.rotation.y = cube.rotation.y += ( targetRotation - cube.rotation.y ) * 0.05;
    renderer.render( scene, camera );

}
```

For the special pan ,as I like to call it, one may do it by recursively rendering the camera positions, as:

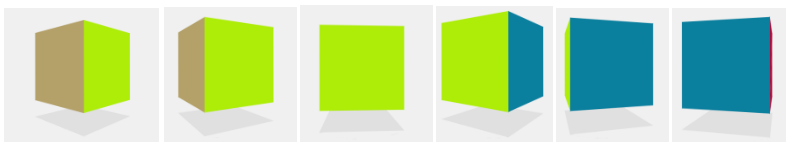


```
function render() {

    camera.position.x += ( mouseX - camera.position.x ) * 4;
    camera.position.y += ( - mouseY + 200 - camera.position.y ) * 4;
    camera.lookAt( scene.position );

    renderer.render( scene, camera );

}
```

What this does is actually pans it in simple ways along both axis, but what is more beautiful is the way it is different from every other pan, is the cinematic feel of the pan.

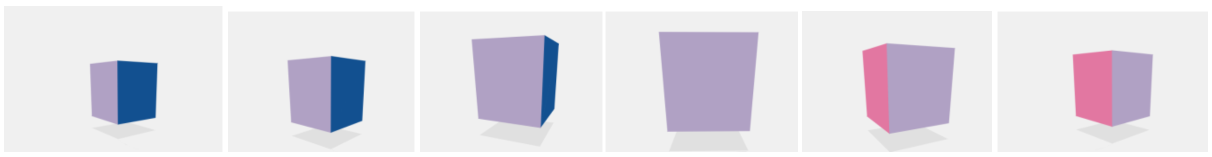
What would actually happen?

Normal Pan:



This example of pan is shown along one axis only. Distance of cube from camera remains constant throughout.

The Special Pan:



This example of pan is shown along one axis only. Distance of camera from cube decreases and then increases again. Gives cinematic feel.

Hurdles?

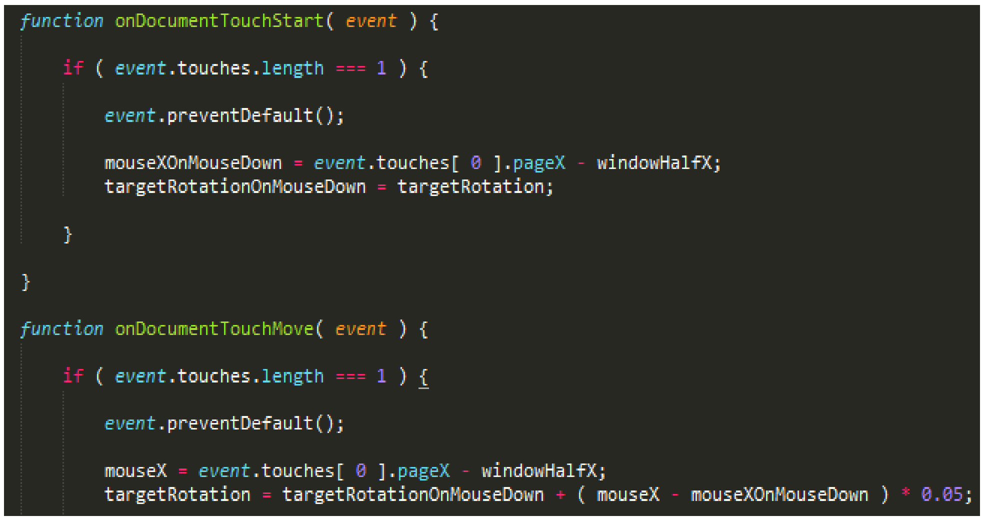
Creating this for one axis is a difficult task in itself for a huge amount of data, and then along all axis makes it difficult to render. Thus I need to think of a better way to pan it along multiple and possibly all axis in the special manner.

- Continuation of movement after the cursor stops .

  Why this?

  Gives a realistic feeling of how a graph must move when a cursor urges it to rotate, the same way Google Earth does. This is what is expected to happen.

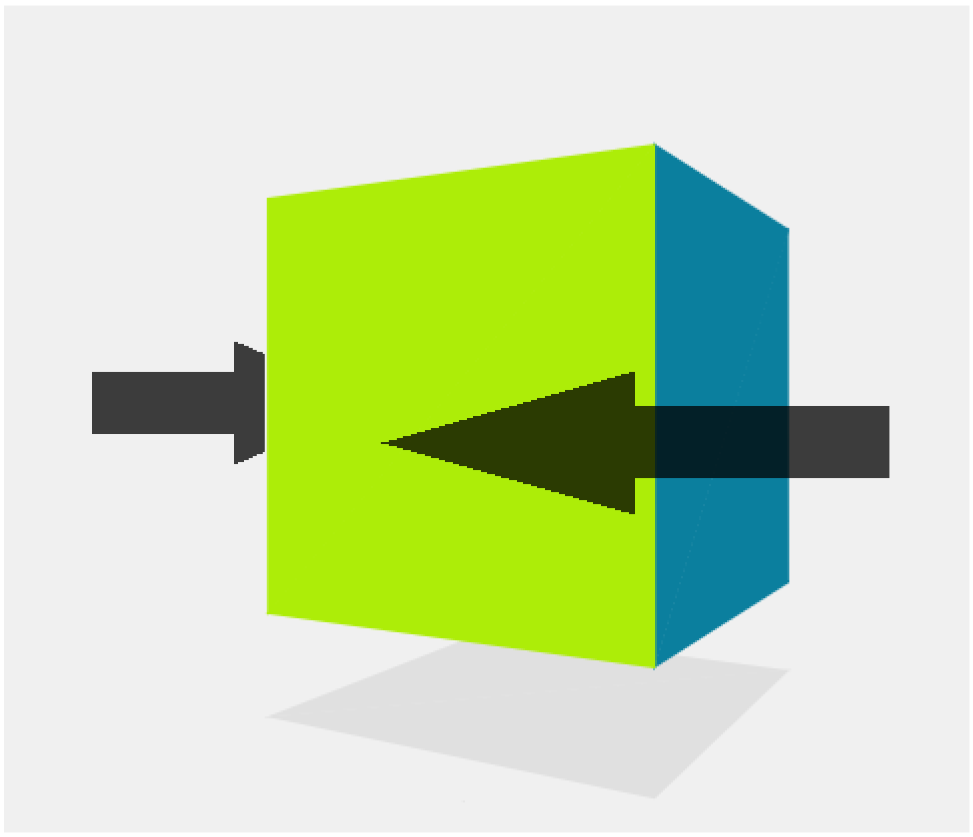
  How would I do it?



```
function onDocumentTouchStart( event ) {

    if ( event.touches.length === 1 ) {

        event.preventDefault();

        mouseXOnMouseDown = event.touches[ 0 ].pageX - windowHalfX;
        targetRotationOnMouseDown = targetRotation;

    }

}

function onDocumentTouchMove( event ) {

    if ( event.touches.length === 1 ) {

        event.preventDefault();

        mouseX = event.touches[ 0 ].pageX - windowHalfX;
        targetRotation = targetRotationOnMouseDown + ( mouseX - mouseXOnMouseDown ) * 0.05;
```

Using the targetRotation variable with the targetRotationOnMouseDown allows us to use the time gap between the present and the time the cursor stopped. I extended the motion of the cube by multiplying it by a reducing factor of 0.05.Thus every time this function steps up, velocity decreases by a factor of 0.05 as show above.

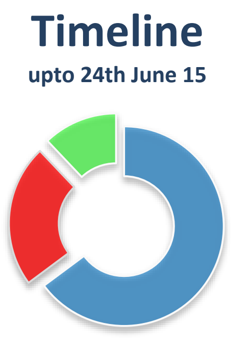
What will actually happen?
After we have once pushed the cube to move, even if we stop the cursor the cube continues its motion until friction takes over(The factor 0.05)



4.<u>Sectioning of the plot to provide better view of inner points:</u>

- Sectioning through axis

Why this?
Data analysis and visualisation is always easier when the amount of data is small. To analyse large amount of data one may divide large data into smaller bits and analyse required piece independently. This analysis. This 'Divide and Conquer' process is generally used by us, developers to tackle difficult problems.



How would I do it?

Using Boolean operations with a library like 'Constructive Solid Geometry' and do a mesh subtraction. Other manipulations can be performed by using algorithms to slice a mesh etc.

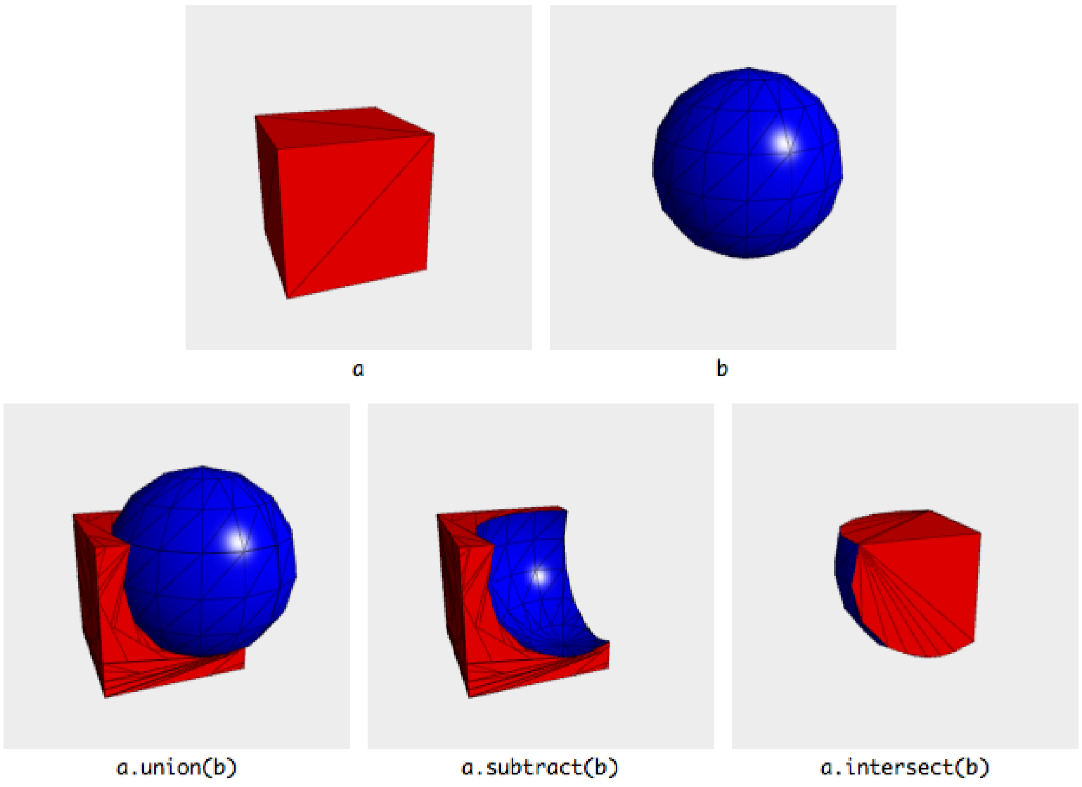
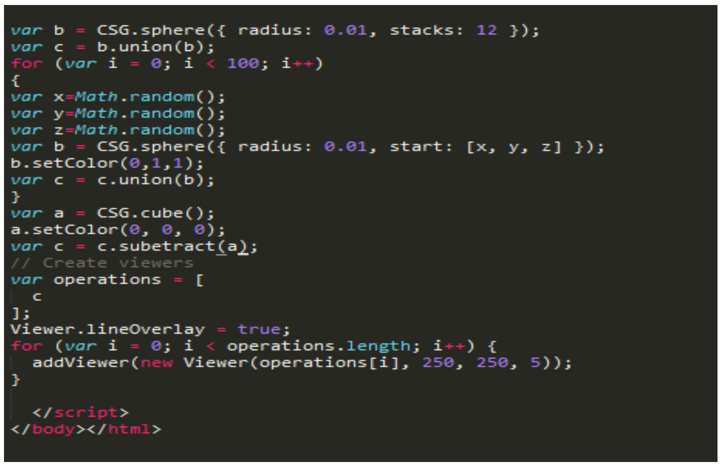


**image source* http://github.com/evanw/csg.js/*

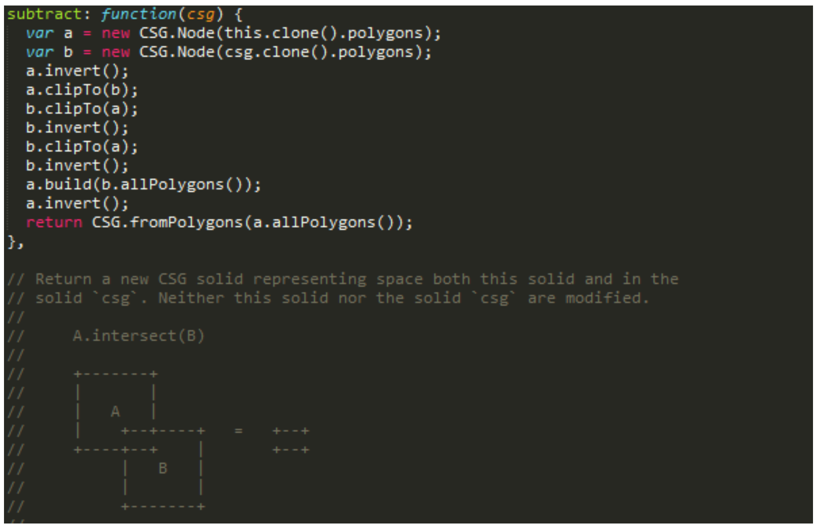
This would require a lot of time and effort as this is something I haven't done in the past but looks like a good idea to work on.

HOW WOULD I DO IT?

What the csg.js library does is it provides us with a very specific set of commands, which allow us to remove certain geometries from the mother geometry. Using the a. subtract(b) command one may simply remove parts of graph that are not required and analyse the rest of it. The commands and usage being something like:



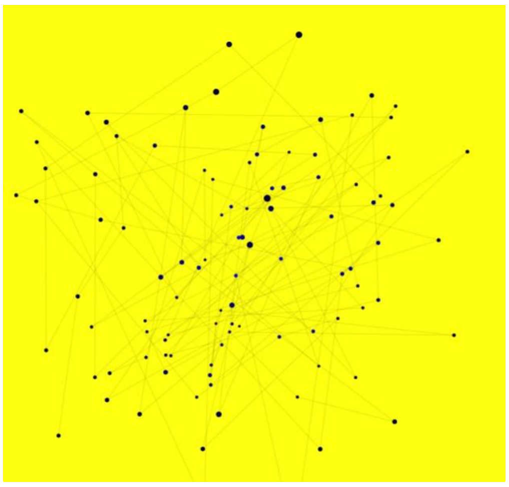
```
var b = CSG.sphere({ radius: 0.01, stacks: 12 });
var c = b.union(b);
for (var i = 0; i < 100; i++)
{
var x=Math.random();
var y=Math.random();
var z=Math.random();
var b = CSG.sphere({ radius: 0.01, start: [x, y, z] });
b.setColor(0,1,1);
var c = c.union(b);
}
var a = CSG.cube();
a.setColor(0, 0, 0);
var c = c.subetract(a);
// Create viewers
var operations = [
  c
];
Viewer.lineOverlay = true;
for (var i = 0; i < operations.length; i++) {
  addViewer(new Viewer(operations[i], 250, 250, 5));
}

  </script>
</body></html>
```



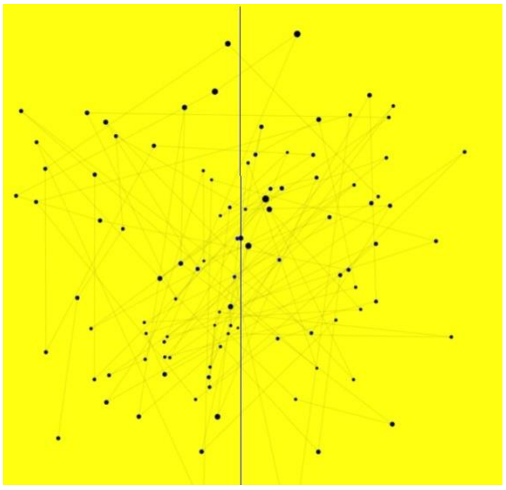
```
subtract: function(csg) {
  var a = new CSG.Node(this.clone().polygons);
  var b = new CSG.Node(csg.clone().polygons);
  a.invert();
  a.clipTo(b);
  b.clipTo(a);
  b.invert();
  b.clipTo(a);
  b.invert();
  a.build(b.allPolygons());
  a.invert();
  return CSG.fromPolygons(a.allPolygons());
},

// Return a new CSG solid representing space both this solid and in the
// solid `csg`. Neither this solid nor the solid `csg` are modified.
//
//     A.intersect(B)
//
//     +-------+
//     |       |
//     |   A   |
//     |    +--+----+   =   +--+
//     +----+--+    |       +--+
//          |   B   |
//          |       |
//          +-------+
```

HOW WOULD IT LOOK LIKE?

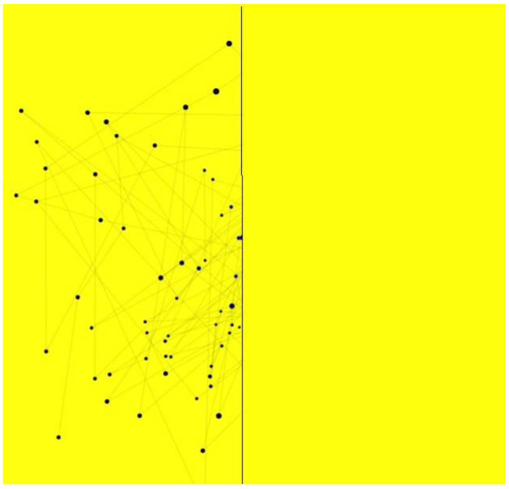
1.



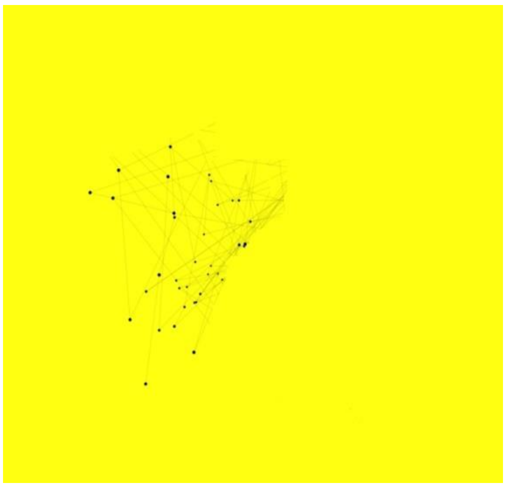
2.



3.

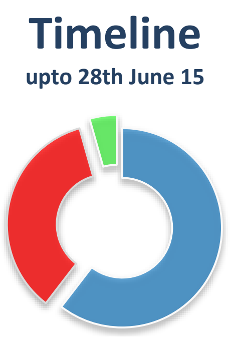


4.



Free hand sectioning: This I suppose would be possible only in 2 dimensional setup and would use the d3.js library to collapse the leftover nodes/ edges to the left/right of a drawn line. It would give us a sense of control over the sectioning part in general. Also we might use the angular visjs library to removed the jagged of nodes/ half cut edges. We may also utilise angular visjs (as mentioned in the additional(s)) to collapse some nodes/edges which are further away from the given point of concern. That may lead to a more efficient and systematic approach as such.

5. Colour gradient/Shadows:

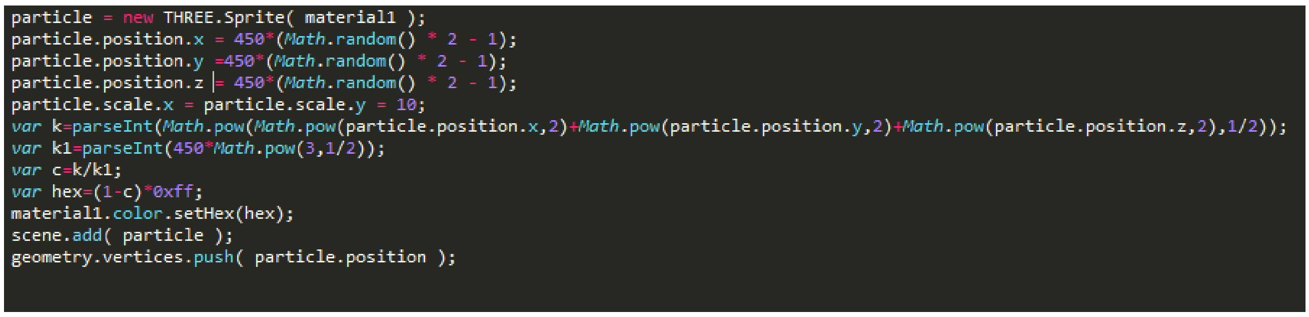


WHY THIS?

Provides better sense of depth , and when used with sectioning the degree of precision is improved. Also as a separate independent tool, it can be utilised well to provide a better view of the network and also follow through the 3 dimensional feel of the overall setup.

HOW WOULD I DO IT?

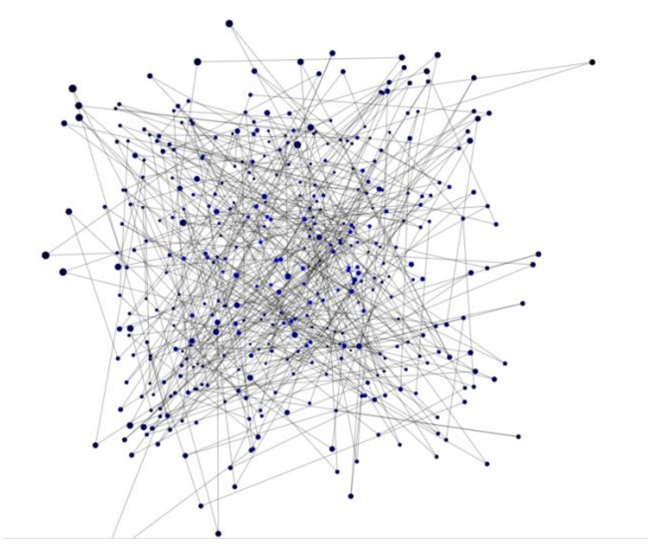
To set the colour gradient, one may use simple code to evaluate the distance of the point from origin and replace the RBG values of the MaterialColour as a function of it.



```
particle = new THREE.Sprite( material1 );
particle.position.x = 450*(Math.random() * 2 - 1);
particle.position.y =450*(Math.random() * 2 - 1);
particle.position.z |= 450*(Math.random() * 2 - 1);
particle.scale.x = particle.scale.y = 10;
var k=parseInt(Math.pow(Math.pow(particle.position.x,2)+Math.pow(particle.position.y,2)+Math.pow(particle.position.z,2),1/2));
var k1=parseInt(450*Math.pow(3,1/2));
var c=k/k1;
var hex=(1-c)*0xff;
material1.color.setHex(hex);
scene.add( particle );
geometry.vertices.push( particle.position );
```

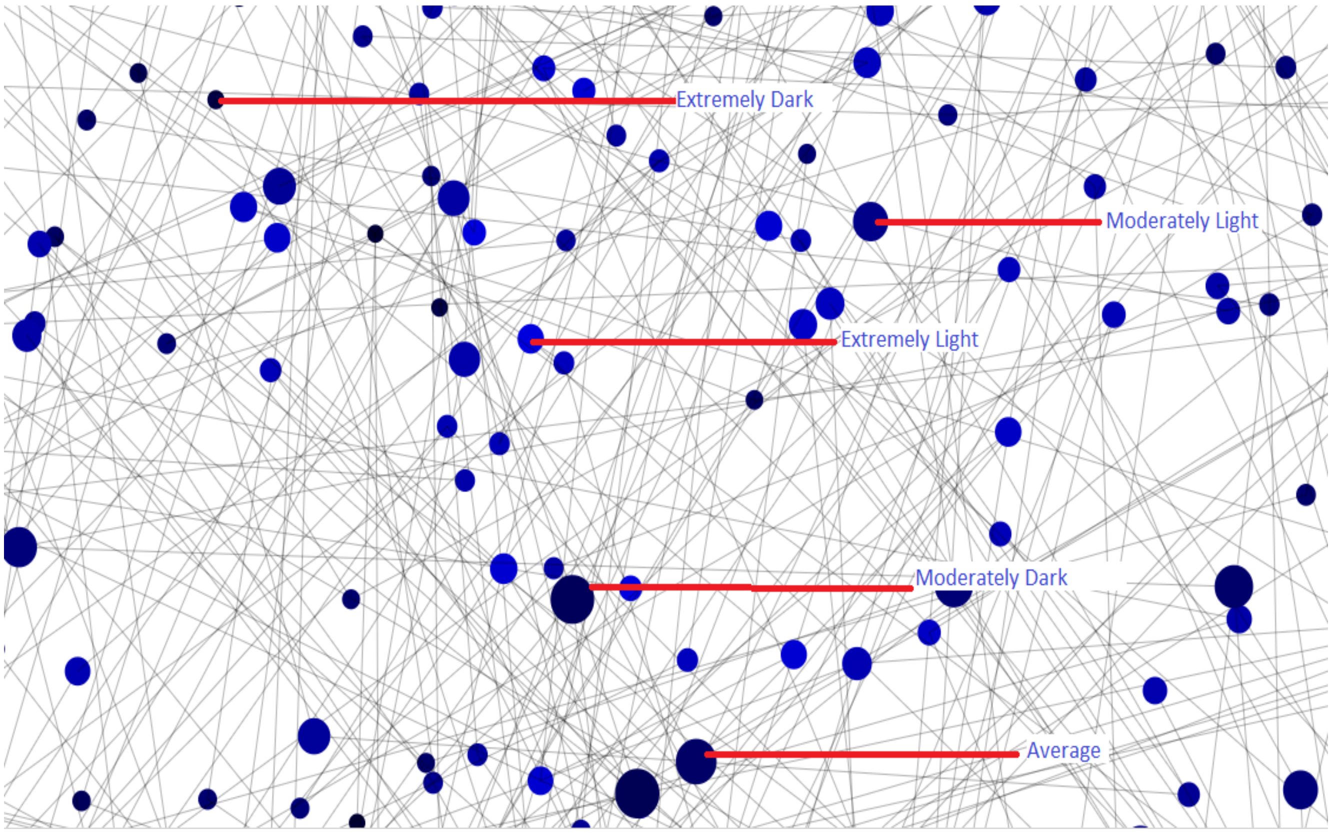
Something like the code given above utilises the fact that we know the upper bound for the distance is known. In cases where it is not known, a simple Max function does the job. Whereas shadows can be induced using the Lambert material and spotlights.

WHAT WOULD IT LOOK LIKE?



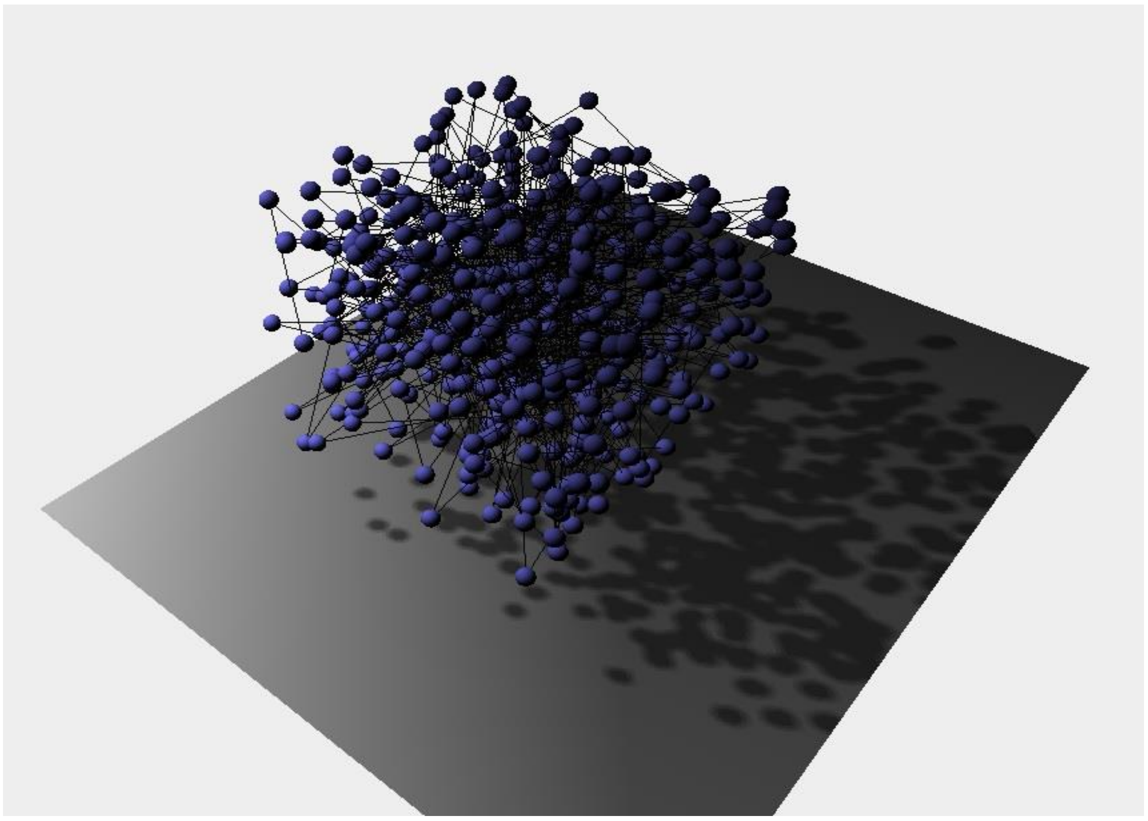
In such a haphazard network with so many points, it becomes extremely tough to analyse their positions when zoomed in. Thus our colour gradient method allows us to catch hold of particles that are inner and outer very easily. On zooming in:

The figure above clearly shows how our analysis has been eased, by the use of colour gradients.



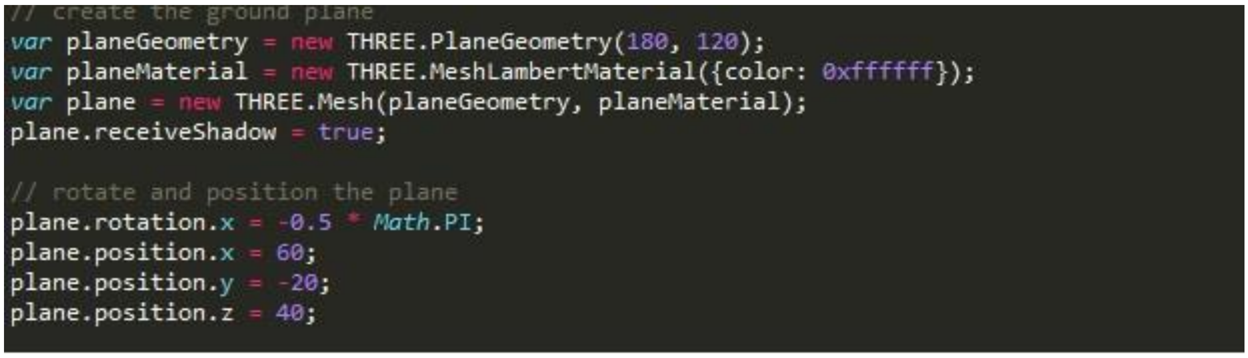
For the shadows part, what I found most intriguing is the way a simple code can create such scenic beauty in our layout and cause a sudden sense of satisfaction.

WHAT WILL IT LOOK LIKE?



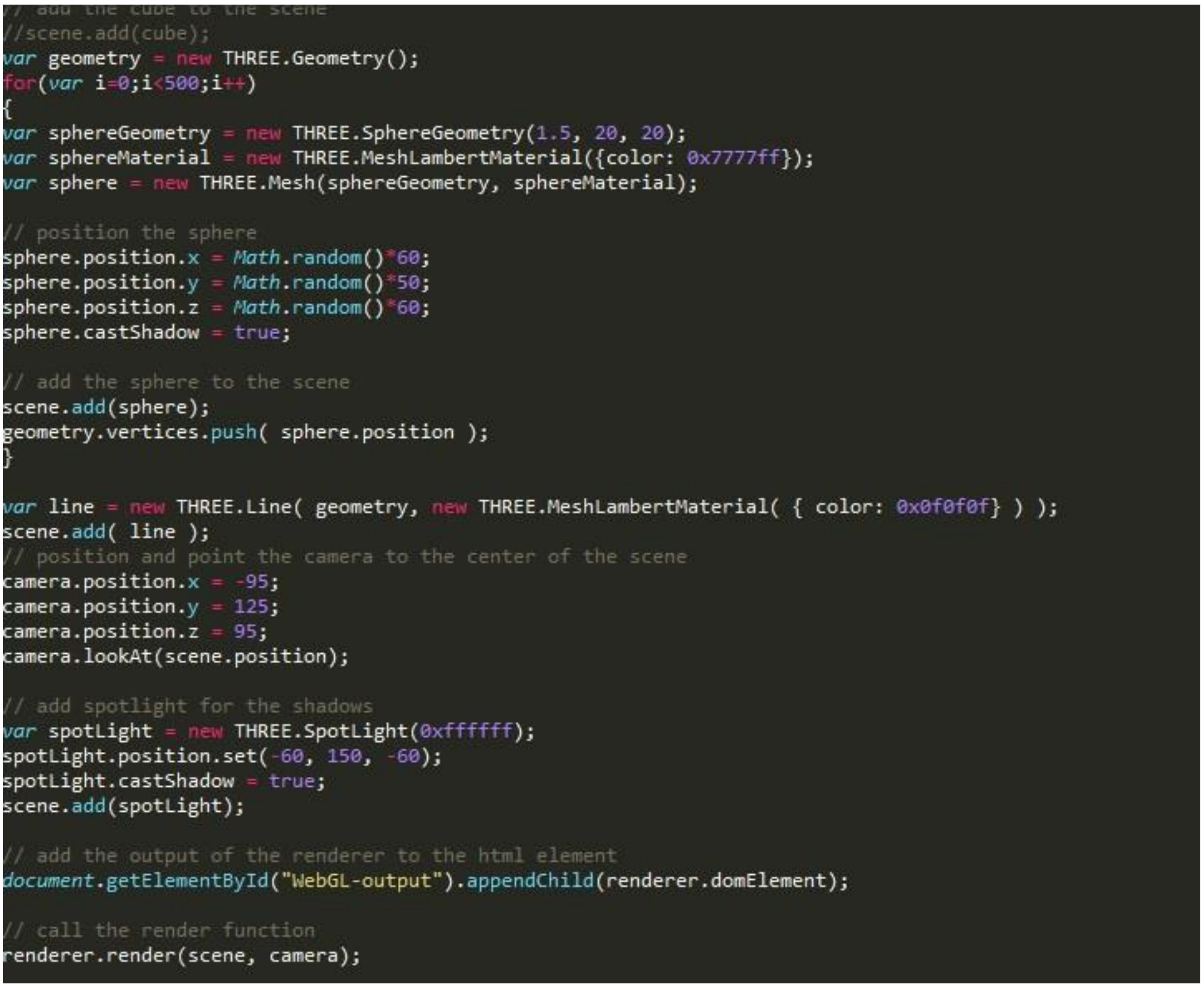
HOW WOULD I DO IT?

Using the three.js library and its functions like spotlight etc. we can create this amazing addition, and certainly it is the best value for code.



```
// create the ground plane
var planeGeometry = new THREE.PlaneGeometry(180, 120);
var planeMaterial = new THREE.MeshLambertMaterial({color: 0xffffff});
var plane = new THREE.Mesh(planeGeometry, planeMaterial);
plane.receiveShadow = true;

// rotate and position the plane
plane.rotation.x = -0.5 * Math.PI;
plane.position.x = 60;
plane.position.y = -20;
plane.position.z = 40;
```

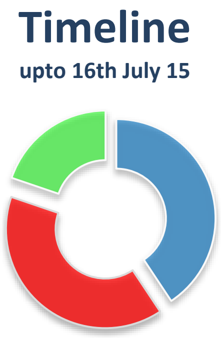
To add the receiving plane (the plane which receives the shadow)



```
// add the cube to the scene
//scene.add(cube);
var geometry = new THREE.Geometry();
for(var i=0;i<500;i++)
{
var sphereGeometry = new THREE.SphereGeometry(1.5, 20, 20);
var sphereMaterial = new THREE.MeshLambertMaterial({color: 0x7777ff});
var sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);

// position the sphere
sphere.position.x = Math.random()*60;
sphere.position.y = Math.random()*50;
sphere.position.z = Math.random()*60;
sphere.castShadow = true;

// add the sphere to the scene
scene.add(sphere);
geometry.vertices.push( sphere.position );
}

var line = new THREE.Line( geometry, new THREE.MeshLambertMaterial( { color: 0x0f0f0f} ) );
scene.add( line );
// position and point the camera to the center of the scene
camera.position.x = -95;
camera.position.y = 125;
camera.position.z = 95;
camera.lookAt(scene.position);

// add spotlight for the shadows
var spotLight = new THREE.SpotLight(0xffffff);
spotLight.position.set(-60, 150, -60);
spotLight.castShadow = true;
scene.add(spotLight);

// add the output of the renderer to the html element
document.getElementById("WebGL-output").appendChild(renderer.domElement);

// call the render function
renderer.render(scene, camera);
```

Final addition of components, light and stage. Rendering it then does the job. Though before the final edit is presentable a lot of testing is required with different camera positions/plane positions and other parts such as kinds/ colours of material, but still the final result makes up for it.

6.Real time change of pattern:



- Pattern change:
  - ✓ Hierarchical
  - ✓ Grid
  - ✓ Hair Network
  - ✓ Point Cloud

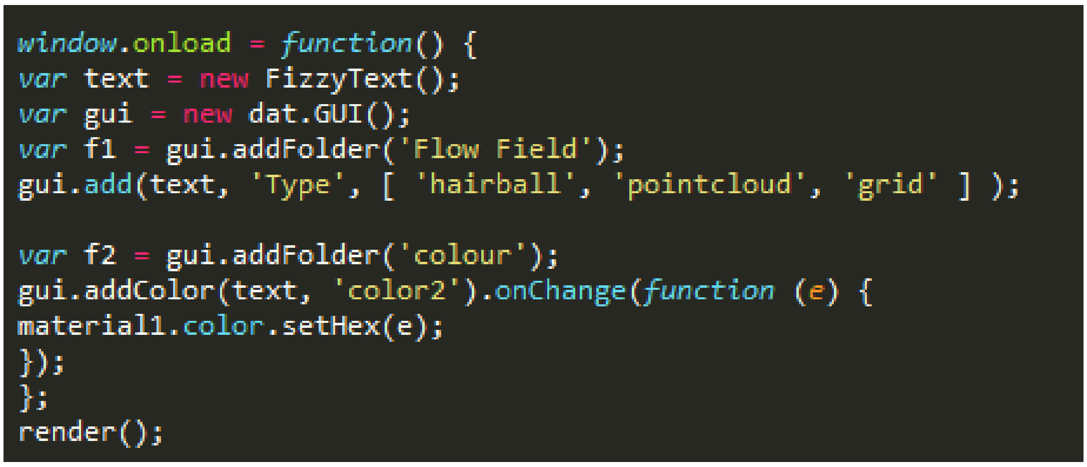
- Colour change:
  - ✓ Negative
  - ✓ Colour scheme

WHY THIS?

Sometimes we encounter data where one type representation is not enough to depict the data totally. More and more representations lead to more and more superiority in the level of understanding.

HOW WOULD THIS HAPPEN?

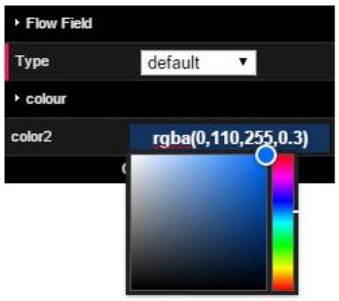
Let us take this one by one. First of all, we change the types real time. To do this we use the library created by some guys at Google, called the dat.GUI which allows you to very easily create a simple user interface component that can change the variables in your code.



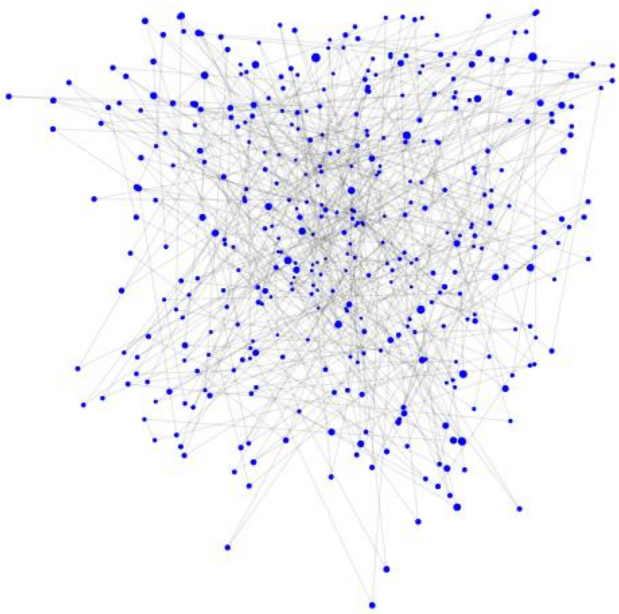
```
window.onload = function() {
var text = new FizzyText();
var gui = new dat.GUI();
var f1 = gui.addFolder('Flow Field');
gui.add(text, 'Type', [ 'hairball', 'pointcloud', 'grid' ] );

var f2 = gui.addFolder('colour');
gui.addColor(text, 'color2').onChange(function (e) {
material1.color.setHex(e);
});
};
render();
```

WHAT WOULD ACTUALLY HAPPEN?

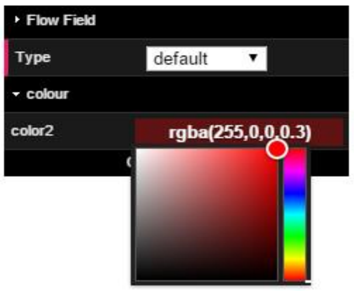
Different types of graphs and colours change real-time, and it looks something like the one given below.



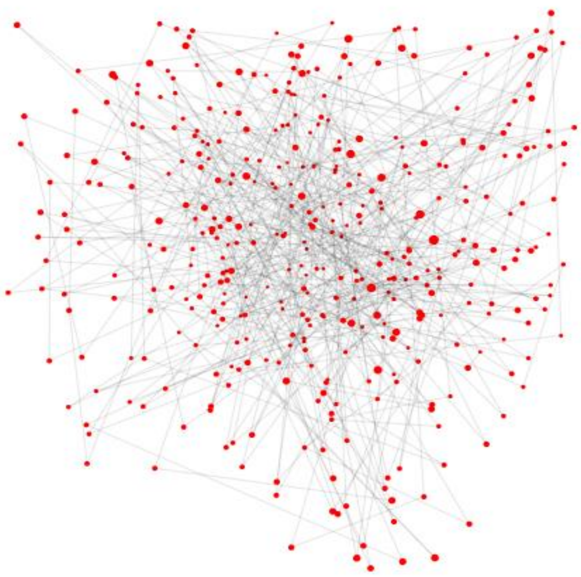
Flow Field
Type default
colour
color2 rgba(0,110,255,0.3)



This is what it looks like in default colours. Now we change the colours using the tools available to us.



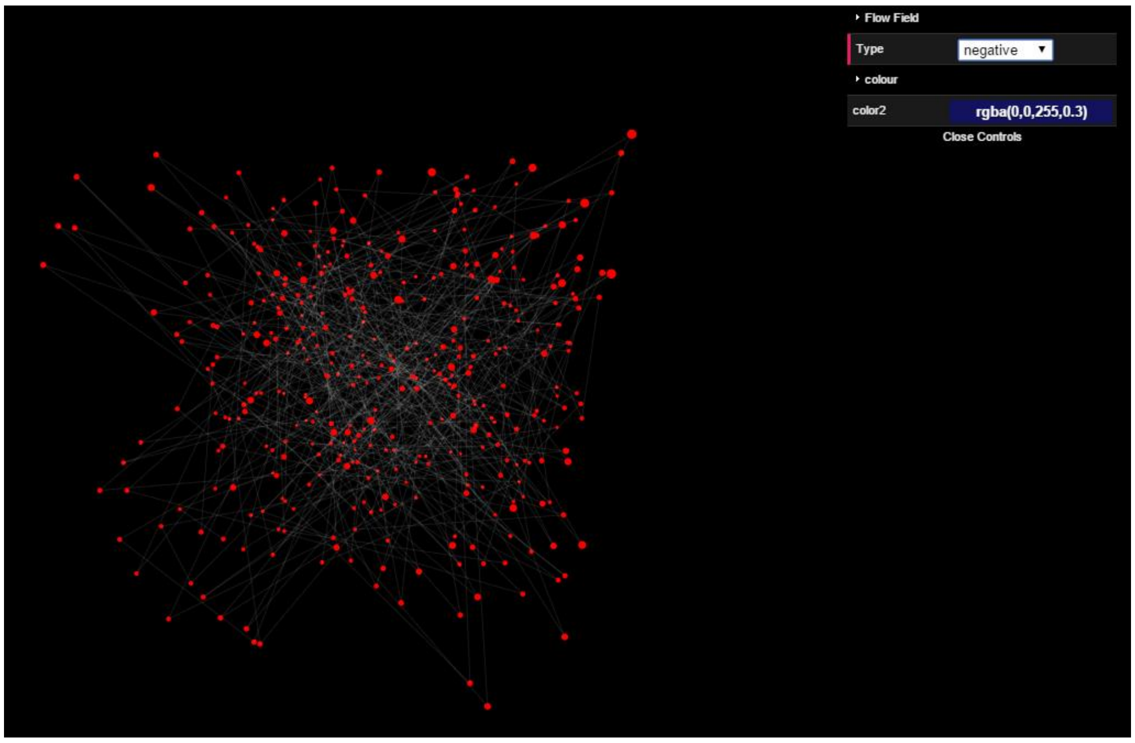
Flow Field
Type default
colour
color2 rgba(255,0,0,0.3)



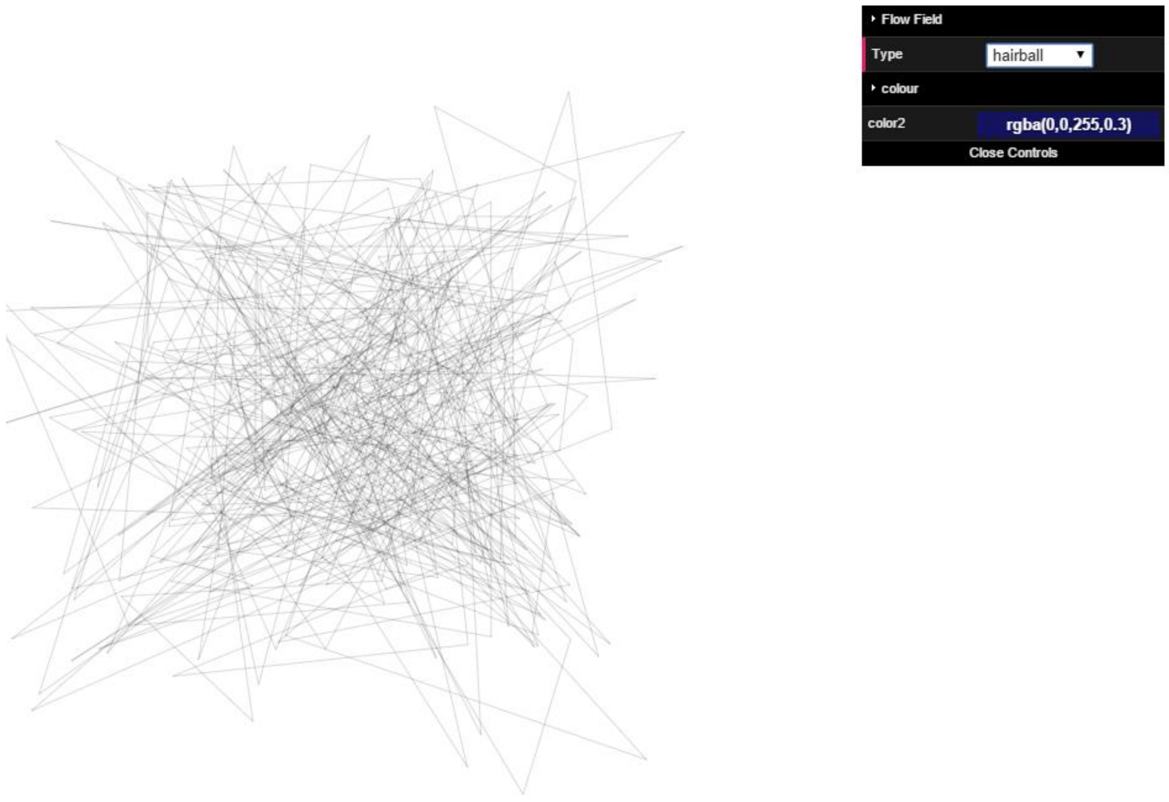
Similarly we might change the type of graphs, and though I wanted to work on more representations, presently these are the only one I could incorporate.

Different kinds of pattern that change on the click:

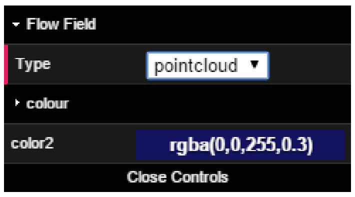
1.Negative:



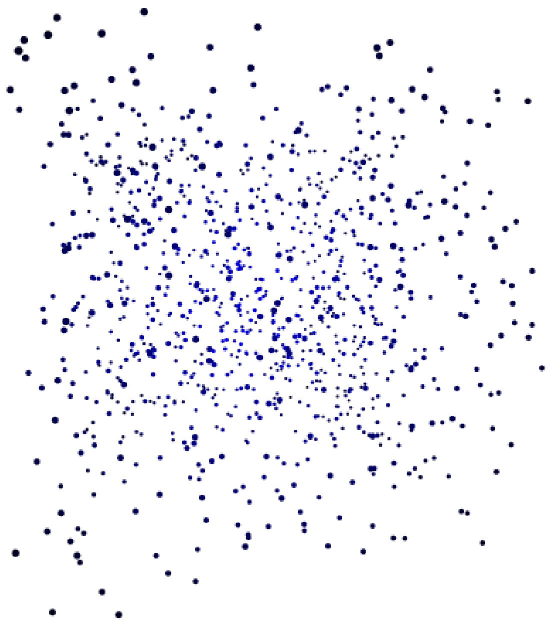
2.Hair network:



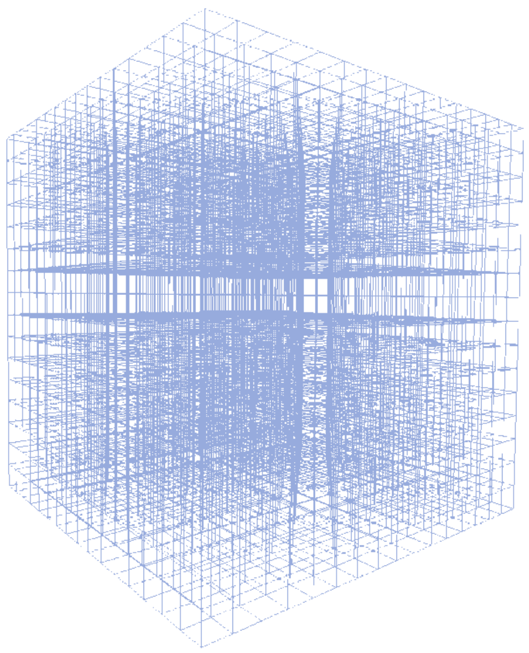
3.Point cloud:



Type: pointcloud
color2: rgba(0,0,255,0.3)

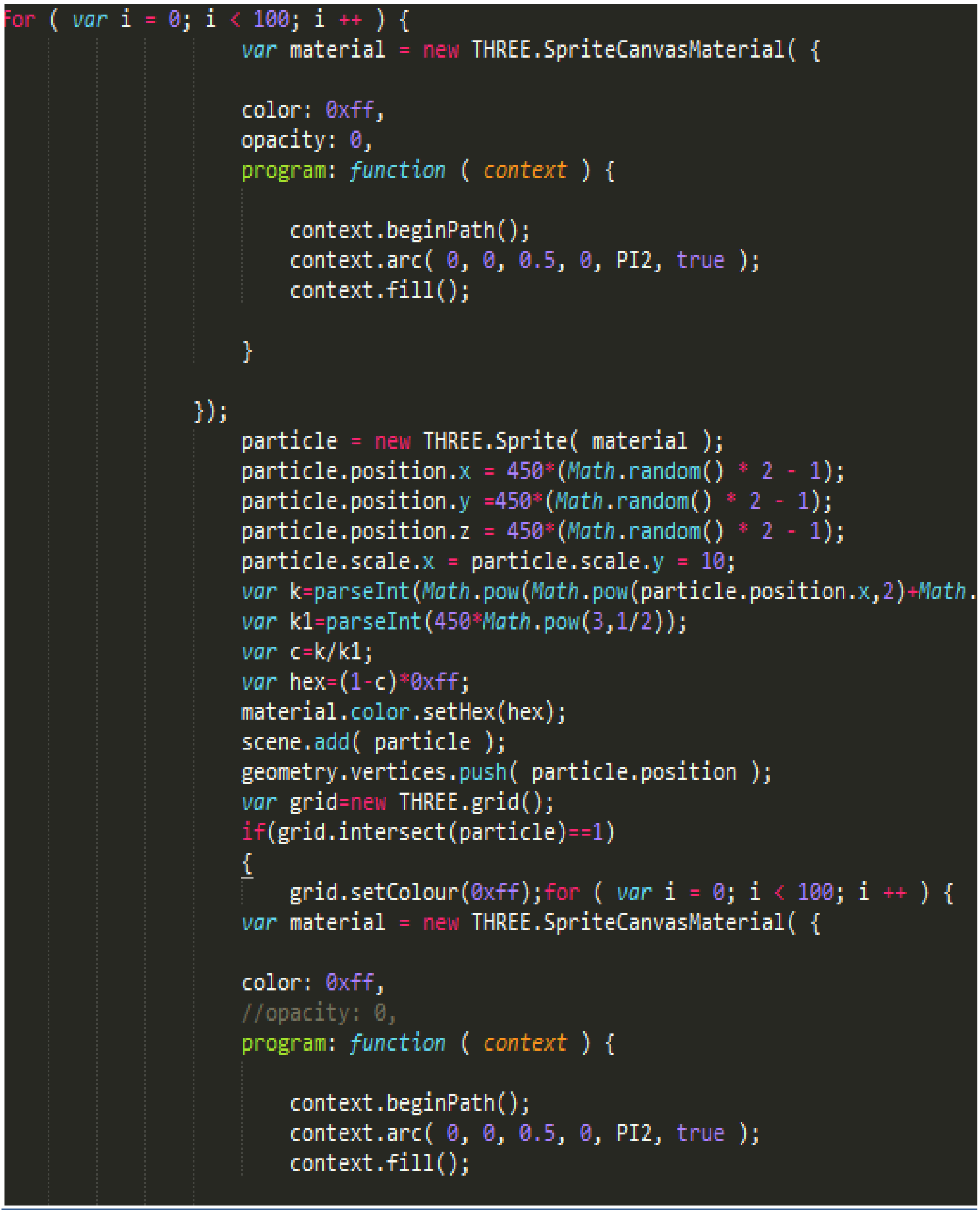


4.Grid:



It is expected to look something like this (with nodes as colour points).

The code used to make the grid layout might look something like this, colouring the cubes that are in approach of a direct node, and exempting the edges for the time being.

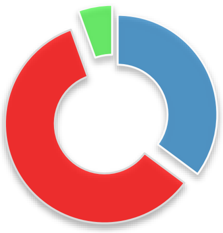


```
for ( var i = 0; i < 100; i ++ ) {
                    var material = new THREE.SpriteCanvasMaterial( {

                    color: 0xff,
                    opacity: 0,
                    program: function ( context ) {

                        context.beginPath();
                        context.arc( 0, 0, 0.5, 0, PI2, true );
                        context.fill();

                    }

                });
                    particle = new THREE.Sprite( material );
                    particle.position.x = 450*(Math.random() * 2 - 1);
                    particle.position.y =450*(Math.random() * 2 - 1);
                    particle.position.z = 450*(Math.random() * 2 - 1);
                    particle.scale.x = particle.scale.y = 10;
                    var k=parseInt(Math.pow(Math.pow(particle.position.x,2)+Math.
                    var k1=parseInt(450*Math.pow(3,1/2));
                    var c=k/k1;
                    var hex=(1-c)*0xff;
                    material.color.setHex(hex);
                    scene.add( particle );
                    geometry.vertices.push( particle.position );
                    var grid=new THREE.grid();
                    if(grid.intersect(particle)==1)
                    {
                        grid.setColour(0xff);for ( var i = 0; i < 100; i ++ ) {
                    var material = new THREE.SpriteCanvasMaterial( {

                    color: 0xff,
                    //opacity: 0,
                    program: function ( context ) {

                        context.beginPath();
                        context.arc( 0, 0, 0.5, 0, PI2, true );
                        context.fill();
```

7.Real time changing of data

Timeline

upto 21st July 15

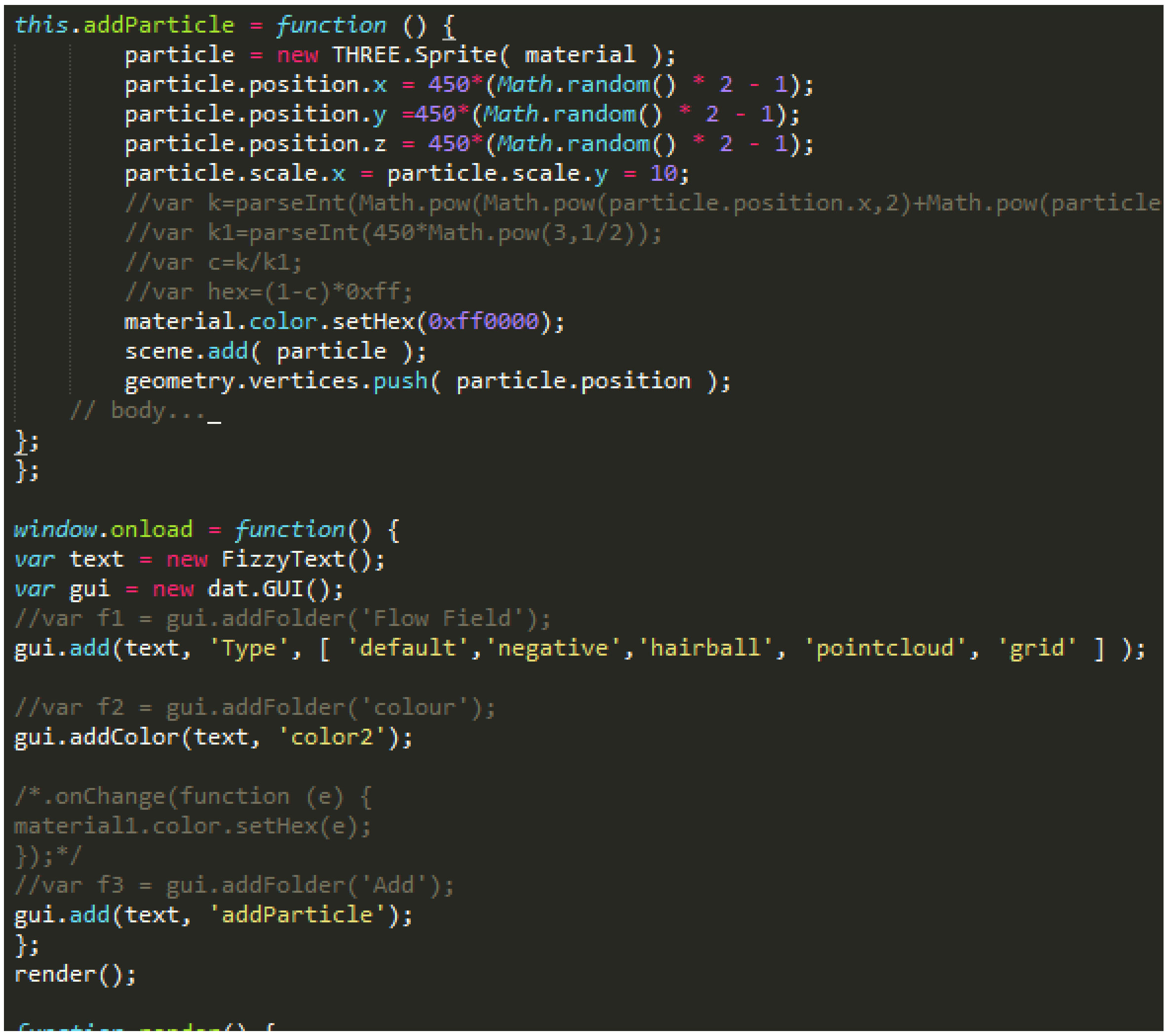


WHY THIS?

We are all humans and making errors one a data that is huge is normal. Therefore for the last minute edits or addition/subtraction of nodes in general must be essential feature of our representation.

HOW WOULD I DO IT?

Similar to the above examples, we will use the dat.GUI library , and give the freedom of adding it to the user.

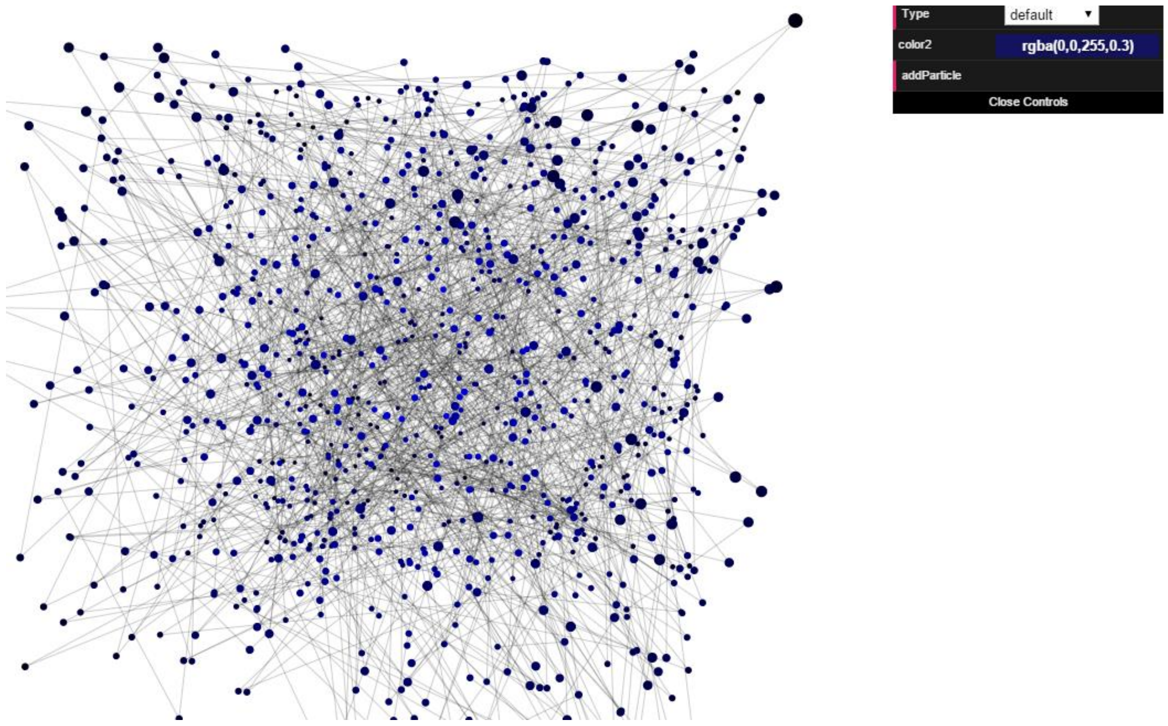


```
this.addParticle = function () {
        particle = new THREE.Sprite( material );
        particle.position.x = 450*(Math.random() * 2 - 1);
        particle.position.y =450*(Math.random() * 2 - 1);
        particle.position.z = 450*(Math.random() * 2 - 1);
        particle.scale.x = particle.scale.y = 10;
        //var k=parseInt(Math.pow(Math.pow(particle.position.x,2)+Math.pow(particle
        //var k1=parseInt(450*Math.pow(3,1/2));
        //var c=k/k1;
        //var hex=(1-c)*0xff;
        material.color.setHex(0xff0000);
        scene.add( particle );
        geometry.vertices.push( particle.position );
    // body..._
};
};

window.onload = function() {
var text = new FizzyText();
var gui = new dat.GUI();
//var f1 = gui.addFolder('Flow Field');
gui.add(text, 'Type', [ 'default','negative','hairball', 'pointcloud', 'grid' ] );

//var f2 = gui.addFolder('colour');
gui.addColor(text, 'color2');

/*.onChange(function (e) {
material1.color.setHex(e);
});*/
//var f3 = gui.addFolder('Add');
gui.add(text, 'addParticle');
};
render();
```

Thus each time on the click event of the control add particle, a new particle is added(at a random position for the time being) but doesn't add the set of edges associated with it. I would work upon this and adding of edges in the deadline set.

Similarly particles can be removed using a similar algorithm.

WHAT WOULD ACTUALLY HAPPEN?



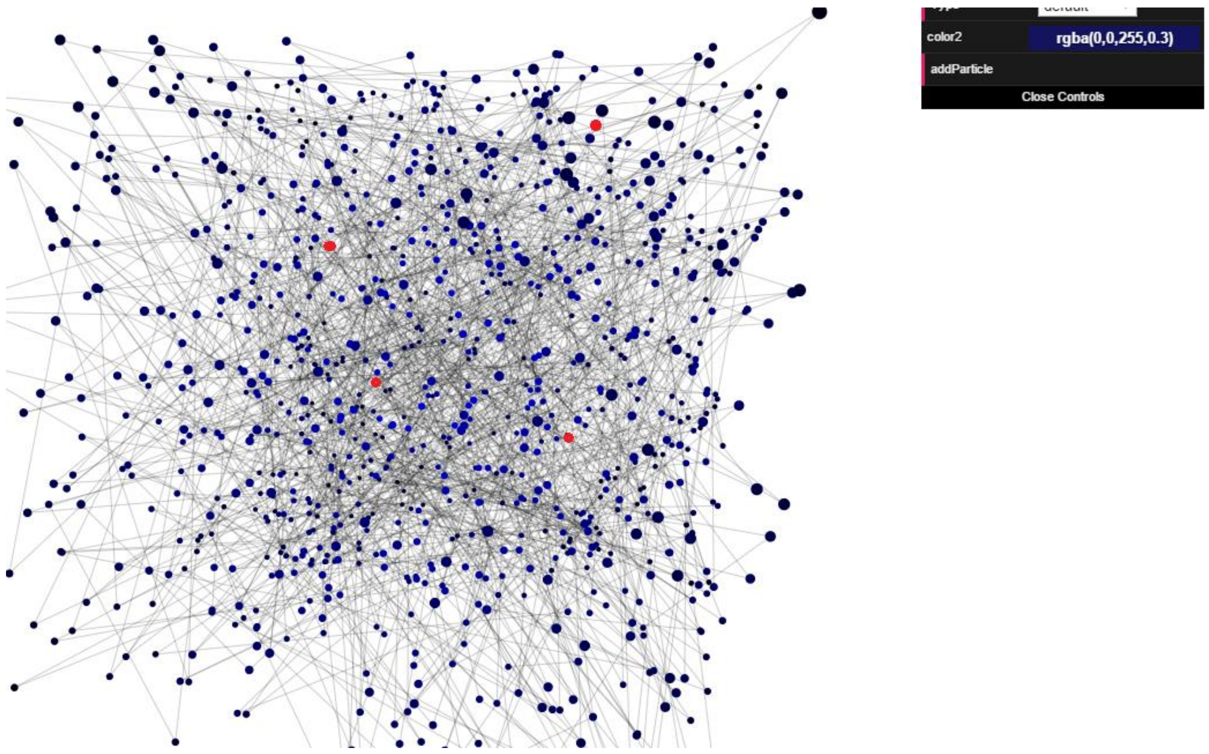
Type default ▾

color2 rgba(0,0,255,0.3)

addParticle

Close Controls

On click event of add particle control, a new red particle is introduced on the stage.



color2 rgba(0,0,255,0.3)

addParticle

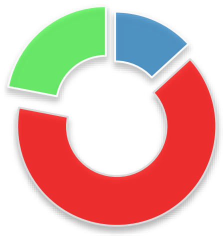
Close Controls

Addition of specific particles (with specified x,y and z coordinates) will be incorporated as the time progresses.

8.Interactivity

Timeline

upto 10th August 15



- Hover and Click event handling.
- Closest node of approach.

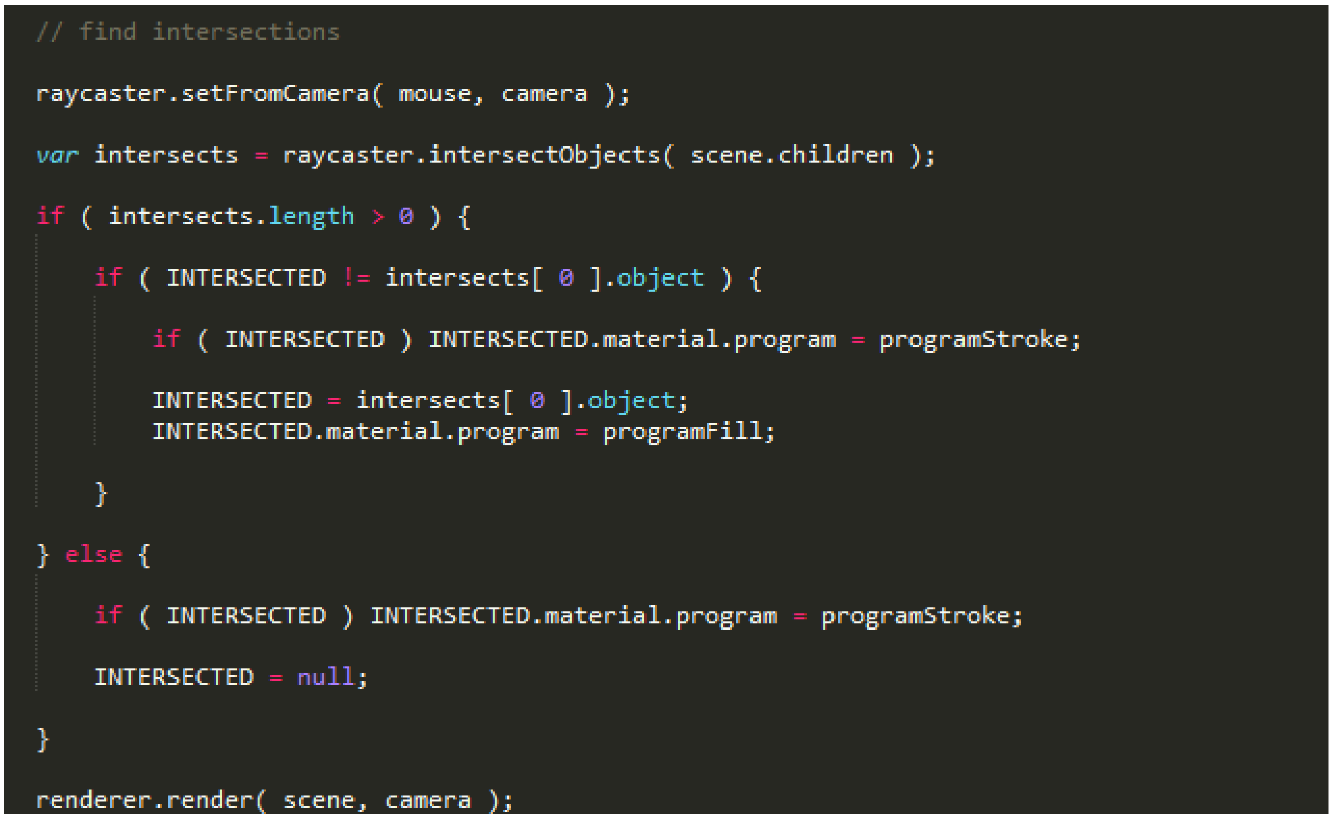
WHY THESE?

Mouse interaction is usually part of the trivia where developers generally want to excel. Hovering and Click interaction provides us with many options to interact with the user and opens up many ideas.

HOW WOULD I DO IT?

✓ *Hover and click even handling.*

Java Script provides us with amazing interface options like intersect etc. which we can use to detect the intersection of mouse and a particle/ edge. While click events are part of the day to day use. What I propose is, there be temporary colouration on hover and permanent on click.

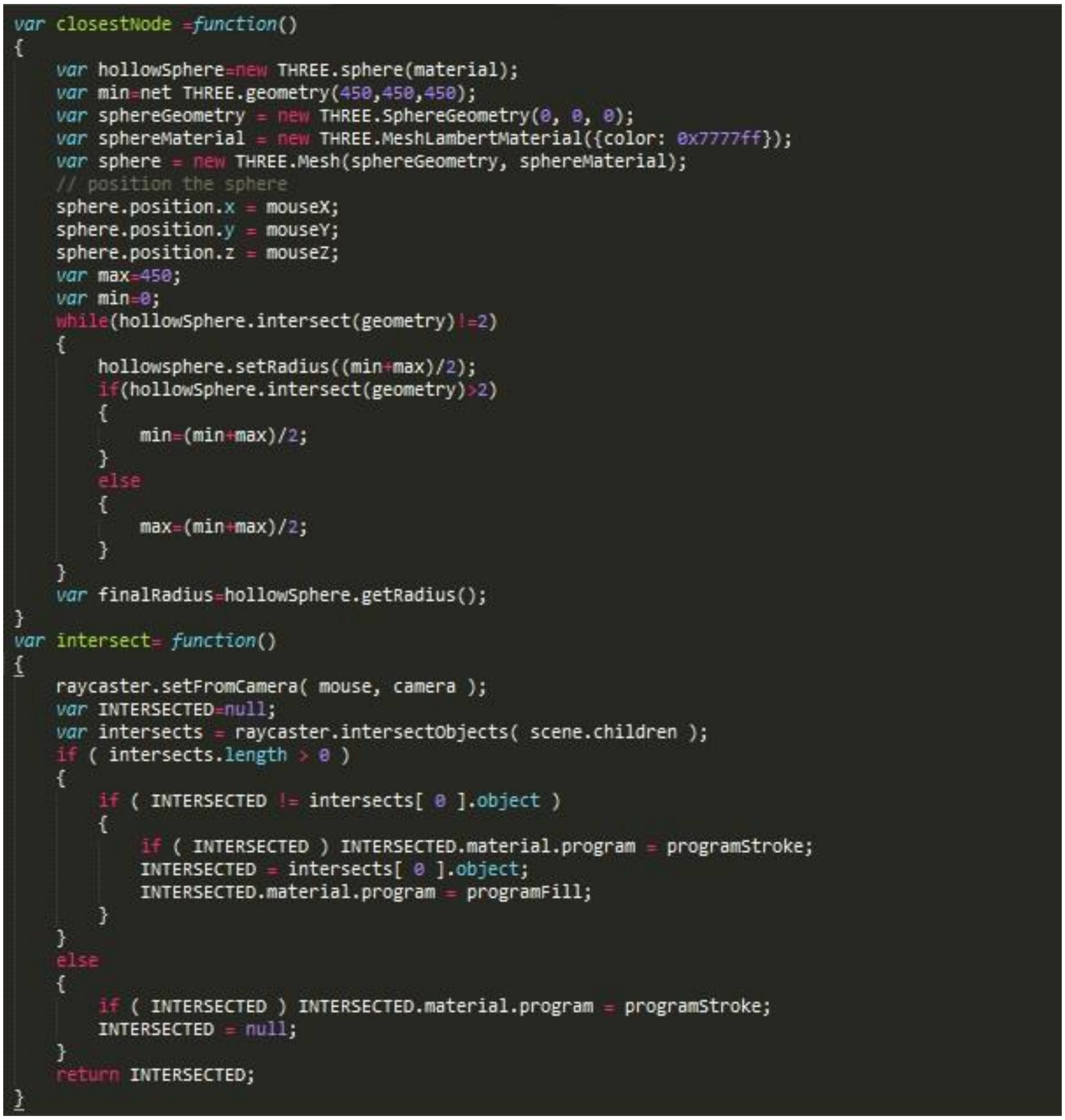


```
// find intersections

raycaster.setFromCamera( mouse, camera );

var intersects = raycaster.intersectObjects( scene.children );

if ( intersects.length > 0 ) {

    if ( INTERSECTED != intersects[ 0 ].object ) {

        if ( INTERSECTED ) INTERSECTED.material.program = programStroke;

        INTERSECTED = intersects[ 0 ].object;
        INTERSECTED.material.program = programFill;

    }

} else {

    if ( INTERSECTED ) INTERSECTED.material.program = programStroke;

    INTERSECTED = null;

}

renderer.render( scene, camera );
```

✓ Closest node of approach.

This is an interesting one. What most people think that the quad Tree approach towards finding the closest particle would be better, I suggest a completely different algorithm. I suggest creating a sphere of radius equal to the distance of

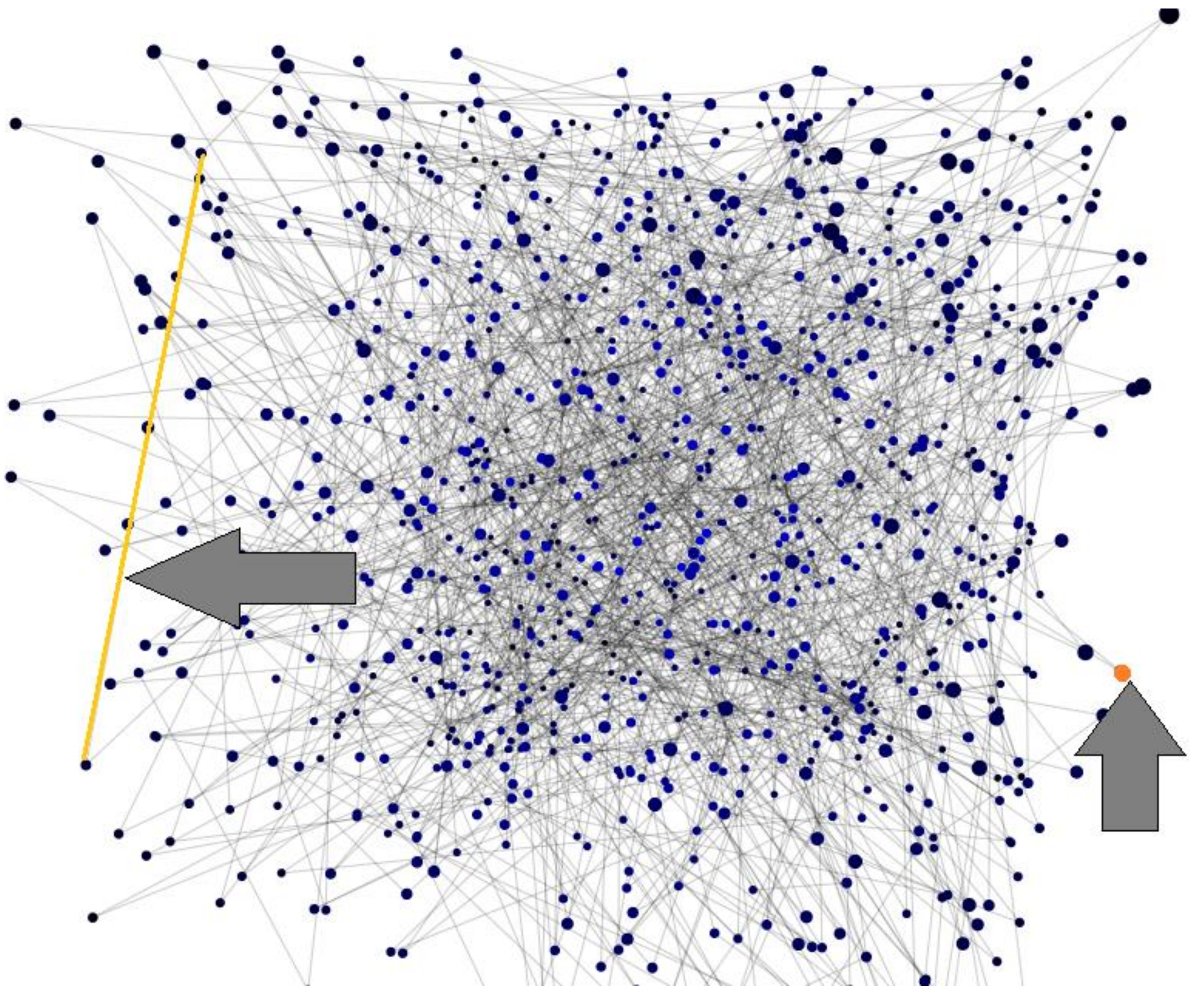
the point from origin. Then, just like binary search, we keep on seeking the mid points of surface of different spheres, until we get 2 lines of intersection. This would be much more efficient than the quad Tree approach for randomly arranged particles.
The actual algorithm in detail follows, which basically utilises the simple selection sort algorithm to provide us with a faster yet efficient approach towards finding the closest node.



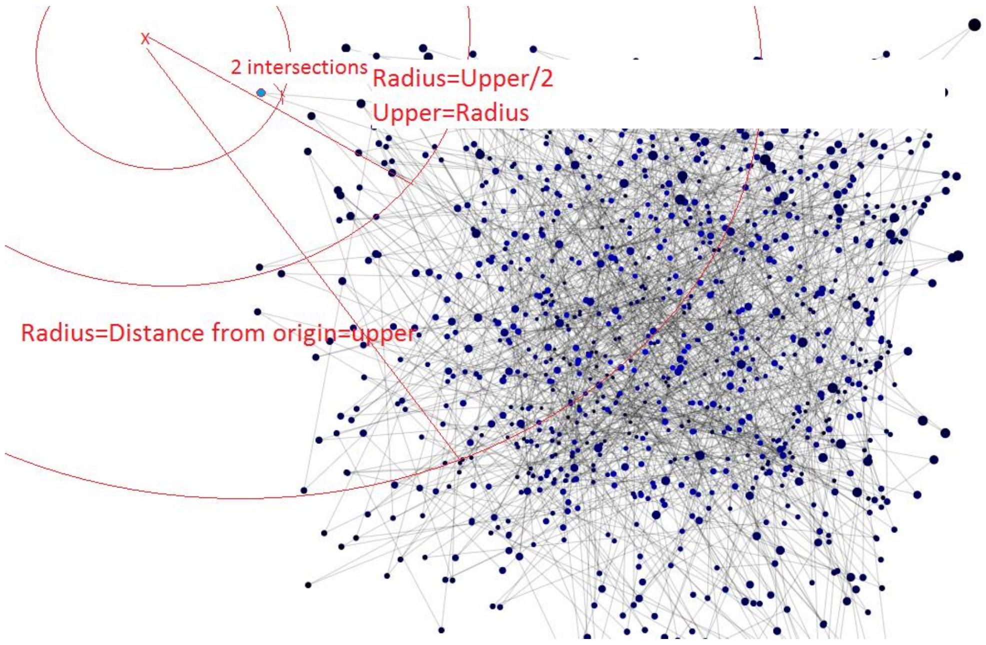
```
var closestNode =function()
{
    var hollowSphere=new THREE.sphere(material);
    var min=net THREE.geometry(450,450,450);
    var sphereGeometry = new THREE.SphereGeometry(0, 0, 0);
    var sphereMaterial = new THREE.MeshLambertMaterial({color: 0x7777ff});
    var sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);
    // position the sphere
    sphere.position.x = mouseX;
    sphere.position.y = mouseY;
    sphere.position.z = mouseZ;
    var max=450;
    var min=0;
    while(hollowSphere.intersect(geometry)!=2)
    {
        hollowsphere.setRadius((min+max)/2);
        if(hollowSphere.intersect(geometry)>2)
        {
            min=(min+max)/2;
        }
        else
        {
            max=(min+max)/2;
        }
    }
    var finalRadius=hollowSphere.getRadius();
}
var intersect= function()
{
    raycaster.setFromCamera( mouse, camera );
    var INTERSECTED=null;
    var intersects = raycaster.intersectObjects( scene.children );
    if ( intersects.length > 0 )
    {
        if ( INTERSECTED != intersects[ 0 ].object )
        {
            if ( INTERSECTED ) INTERSECTED.material.program = programStroke;
            INTERSECTED = intersects[ 0 ].object;
            INTERSECTED.material.program = programFill;
        }
    }
    else
    {
        if ( INTERSECTED ) INTERSECTED.material.program = programStroke;
        INTERSECTED = null;
    }
    return INTERSECTED;
}
```

The quad tree approach is always there as well, though this seemed more innovative, and workable. Similarly we might use the same approach for finding the closest edge, using a standard distanceFrom() function, calculate the distance resulting in a better algorithm altogether, as instead of searching for a quadrant we study for only two edges (for one sphere).

WHAT WOULD ACUALLY HAPPEN?
Hover and click even handling.



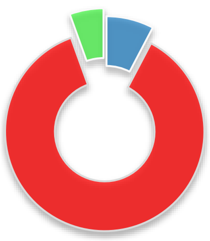
✓ Closest node of approach.



9. Putting everything in place.

Timeline

upto 15th August 15



WHY THIS?

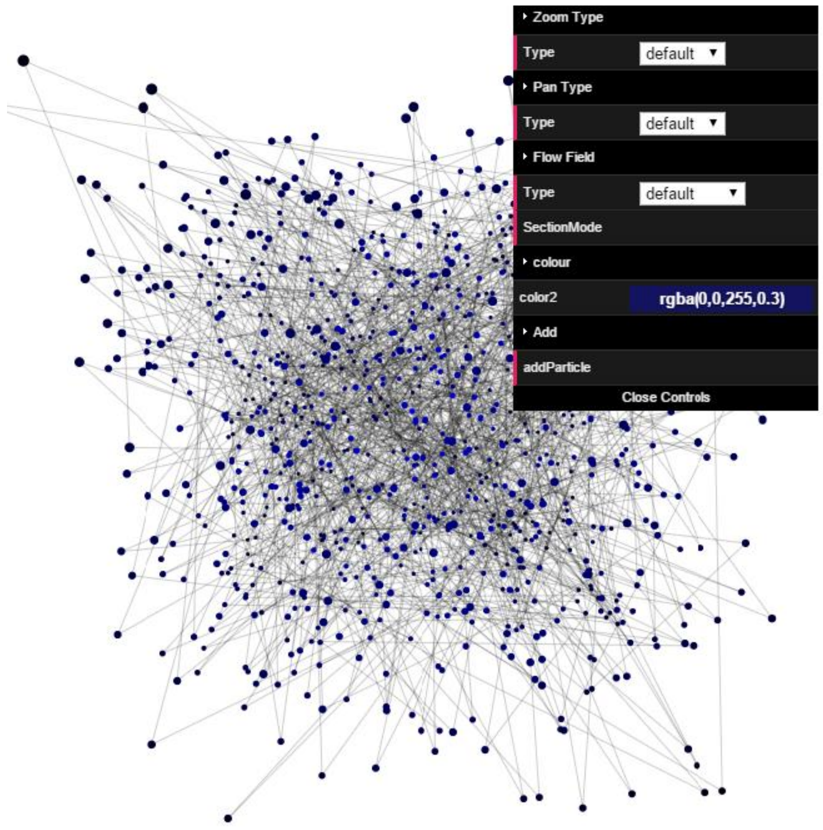
This is one of the most important parts of how our structure would pan out. This is how finally our it would interact with the user, and how well we are able to present different effects, and combine them to give the final look to our network. Also how several different effects can combine together to give a final conclusive adjustment. Also it is very important to get all these things coming together and working. A successful project is the one which has all the small components up and working.

HOW WOULD I DO IT?

Again using the same dat.gui libraries I will provide the different available options, of effects such as panning , zooming , colour gradients, hover and click events and interactivity.

All of these will help us decide the best combination for research.

WHAT WOULD ACTUALLY HAPPEN?



Zoom Type
Type default
Pan Type
Type default
Flow Field
Type default
SectionMode
colour
color2 rgba(0,0,255,0.3)
Add
addParticle
Close Controls

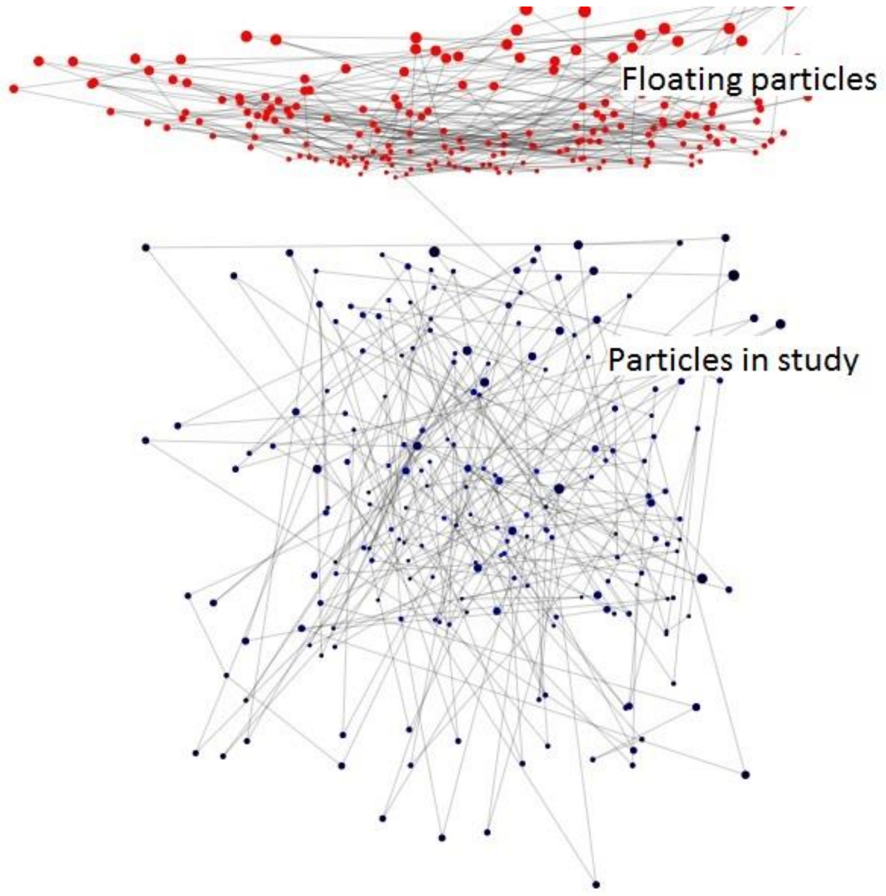
The final is expected to look something like this with more detail and would include several effects, as suggested above, and would help us create a good interactive profile.

Additional(s)

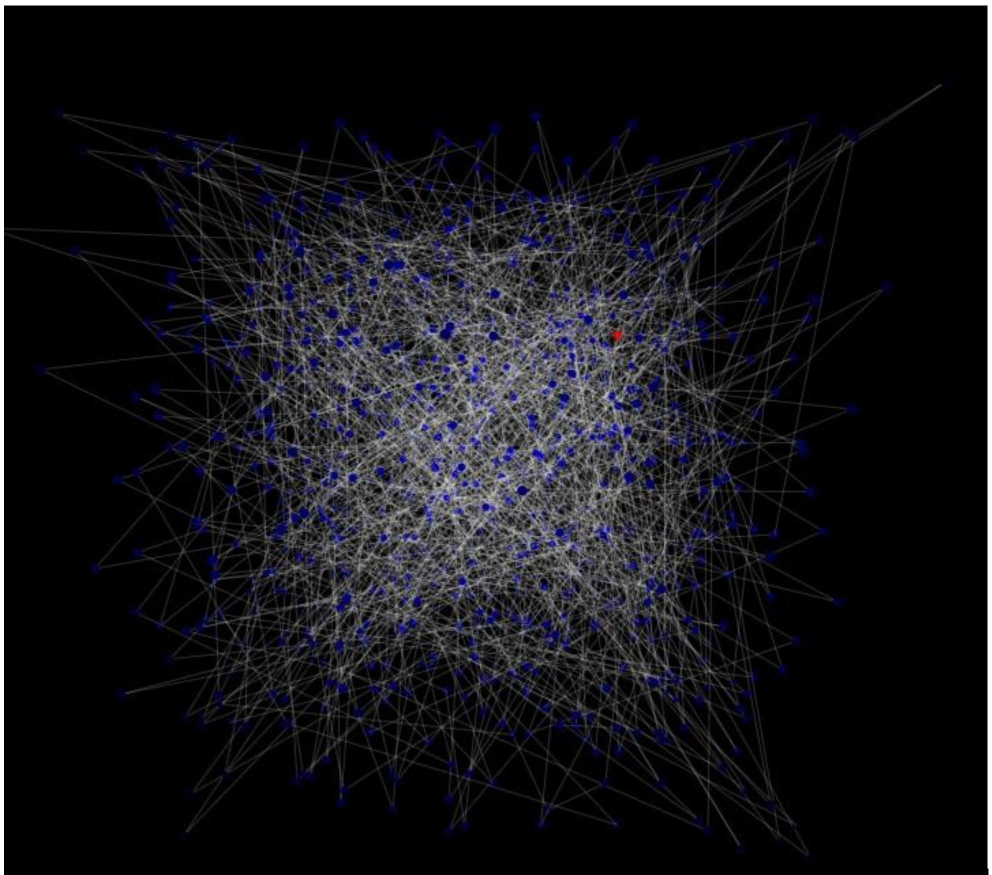
A very important piece of information is that the following might turn out to be the best parts of our project if we find the time to incorporate them and will allow us to stand out.

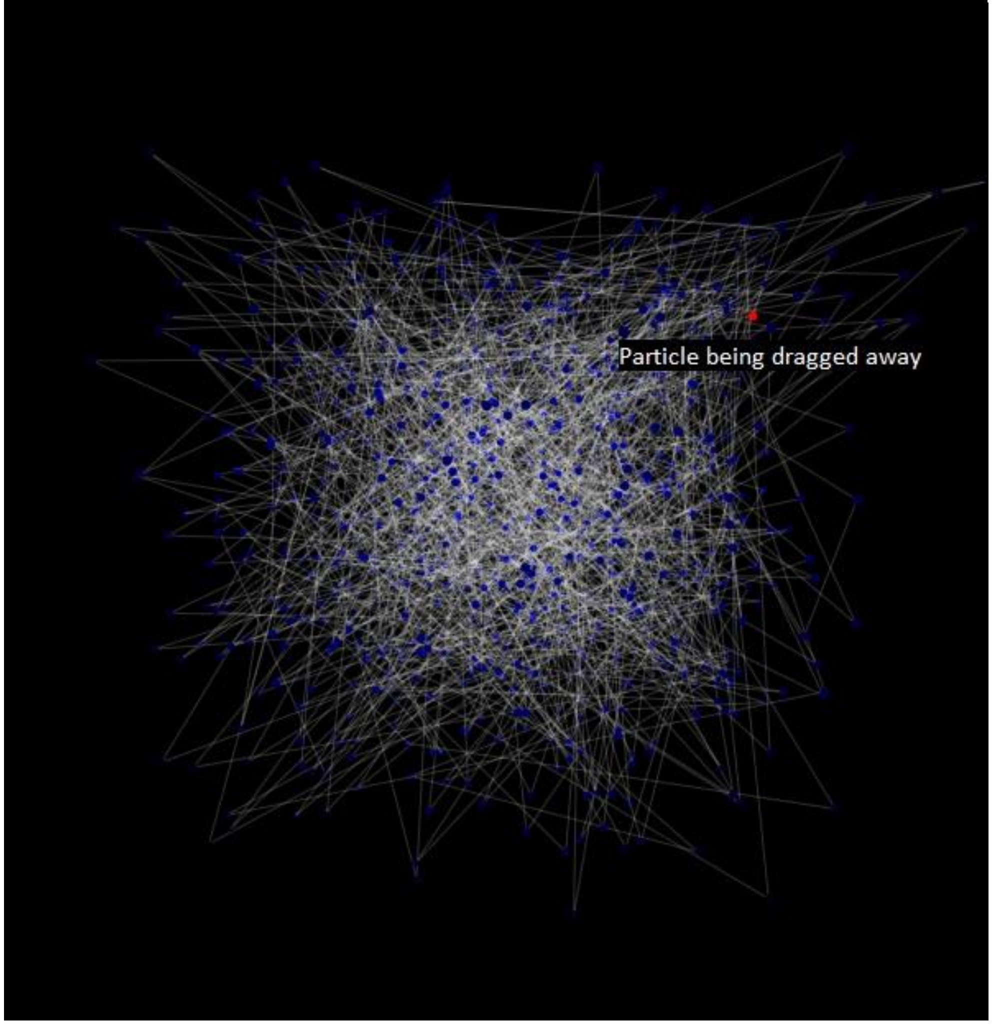
It is always better if a few additional things are done during a project, as it always creates a good ending to our works. The proposed additions are as follows-

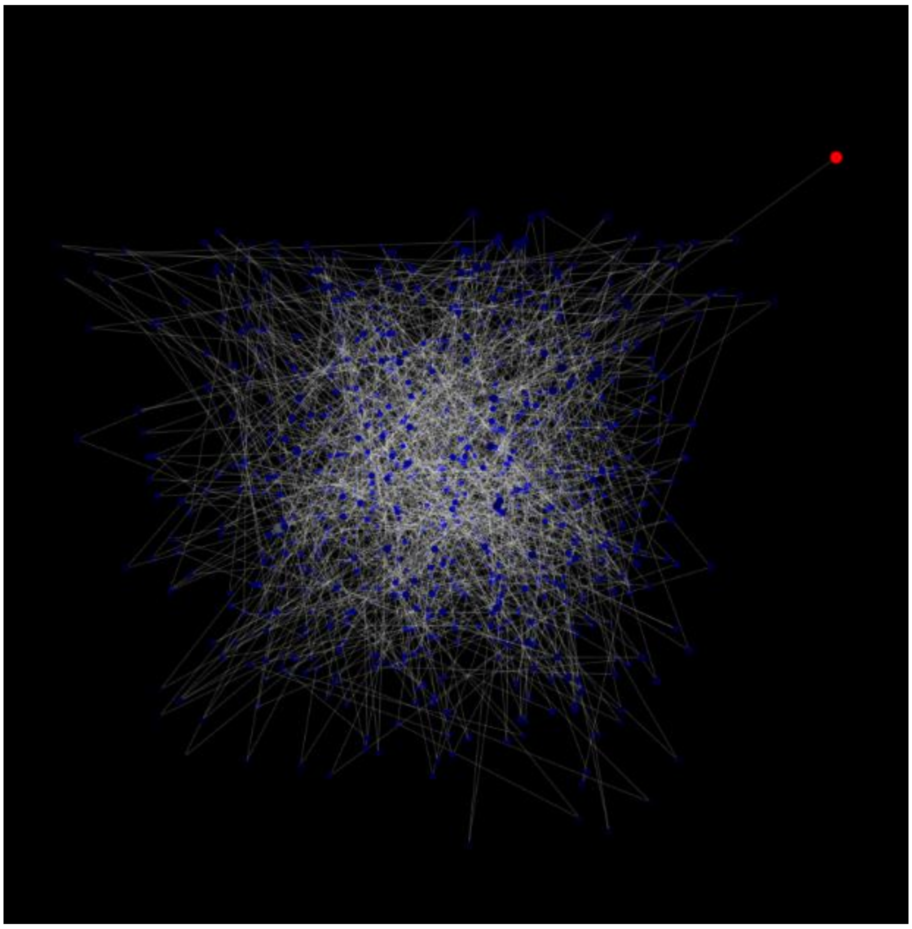
1)Using a combination of the d3.js *(data driven documents)* library and the three.js library, I suggest a layout that allows easy separation of certain particles from the network. What I suggest is adding a float function which allows particles to come above a certain clearing distance off the network and thus make studying only a certain number of important particles easier. It would look something like this, with only particles in study being the blue ones:



2)Similarly, we can create a force field layout, something which the author of d3.js Mike Bostock uses pretty frequently, but implementing it in our own way.What I suggest is free dragging of particles in a mod and then an option to pin them to their location in 3-dimensional space. It looks stunning ad improves the readability and interaction manifolds. It might look something like this when being dragged on and pinned. The changing camera angle doesn't then change the position of the particle after it is pinned.

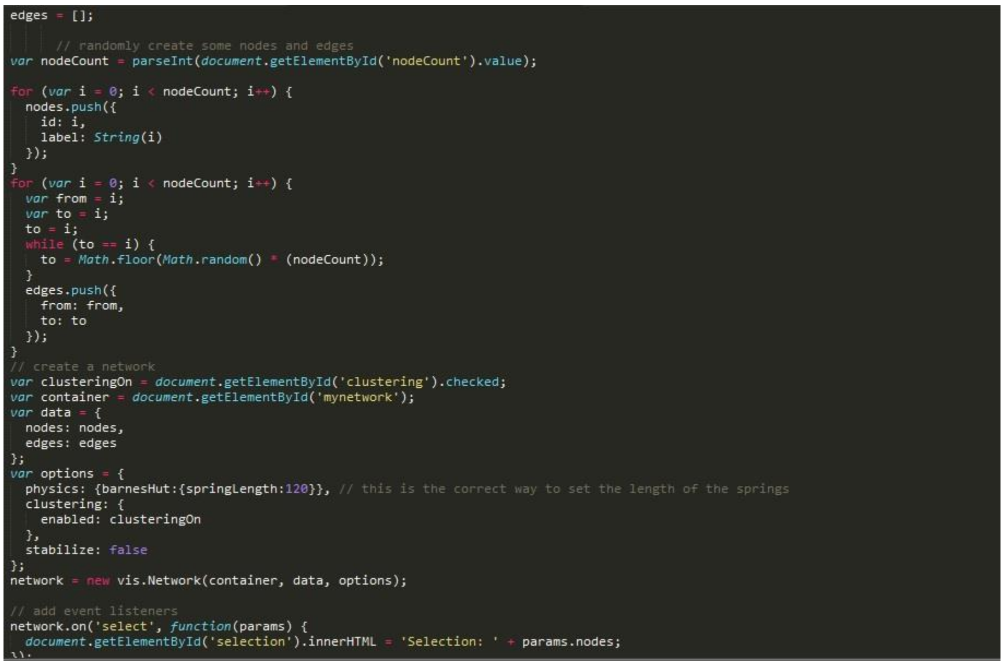



Particle being dragged away



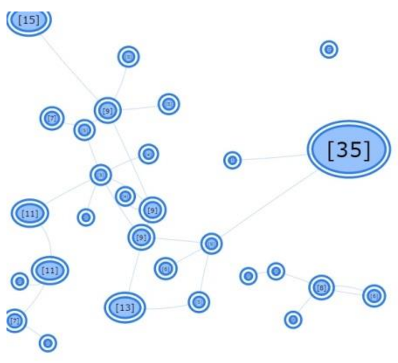
**The red particle represents the particle of the discussion.*

3.Clustering of particles as we move out: This might be very helpful feature if we want to study the population/ generation based plots. It would utilise a new library called the angular-visjs. It pans out beautifully and looks very interactive. It would be done something like this, using simple library functions. The command nodes=[]; and edges[]; is used here to create random nodes and edges which can be customised later on.

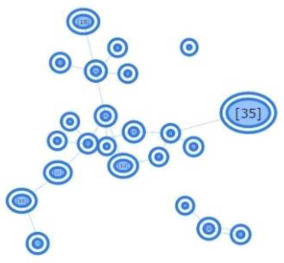


```
edges = [];

      // randomly create some nodes and edges
var nodeCount = parseInt(document.getElementById('nodeCount').value);

for (var i = 0; i < nodeCount; i++) {
  nodes.push({
    id: i,
    label: String(i)
  });
}
for (var i = 0; i < nodeCount; i++) {
  var from = i;
  var to = i;
  to = i;
  while (to == i) {
    to = Math.floor(Math.random() * (nodeCount));
  }
  edges.push({
    from: from,
    to: to
  });
}
// create a network
var clusteringOn = document.getElementById('clustering').checked;
var container = document.getElementById('mynetwork');
var data = {
  nodes: nodes,
  edges: edges
};
var options = {
  physics: {barnesHut:{springLength:120}}, // this is the correct way to set the length of the springs
  clustering: {
    enabled: clusteringOn
  },
  stabilize: false
};
network = new vis.Network(container, data, options);

// add event listeners
network.on('select', function(params) {
  document.getElementById('selection').innerHTML = 'Selection: ' + params.nodes;
```

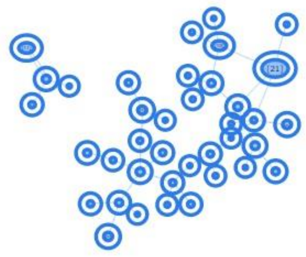
WHAT WOULD IT LOOK LIKE?



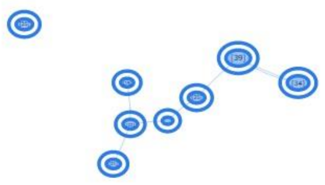
1.



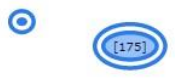
2.



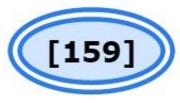
3.



4.



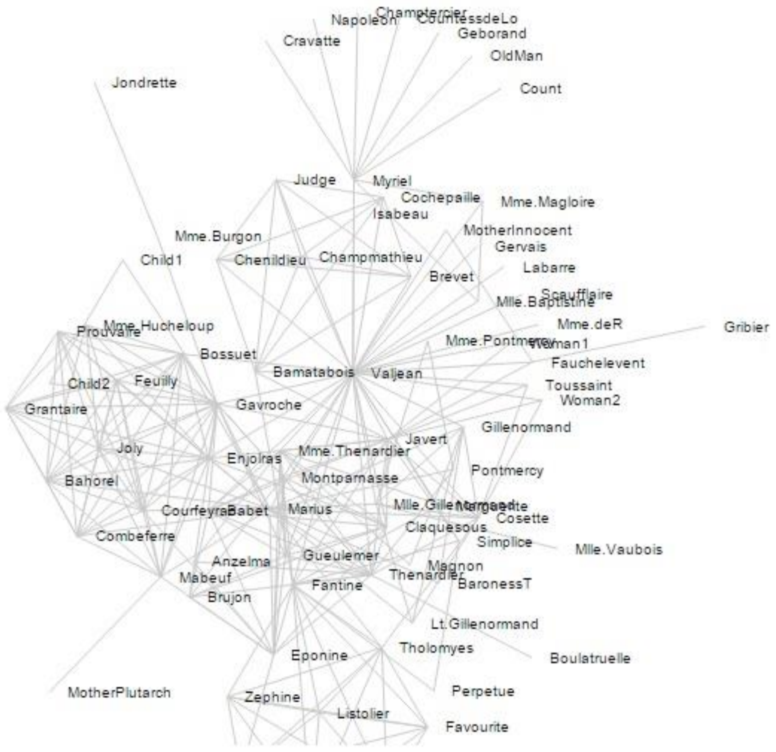
5.



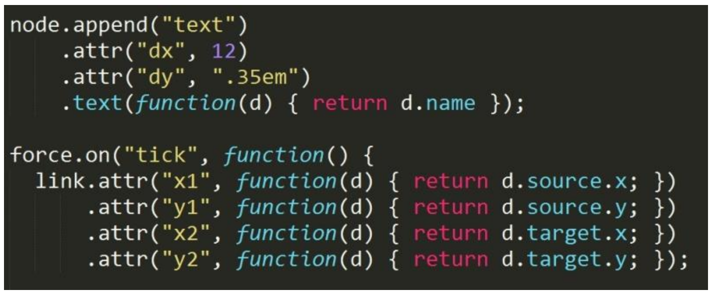
6.

These images show how the layout would look when zooming in/ scrolling up as many particles cluster together to form a heap of particles. This layout has a very specific use and is one of a kind. It would be amazing if we can incorporate such an idea.

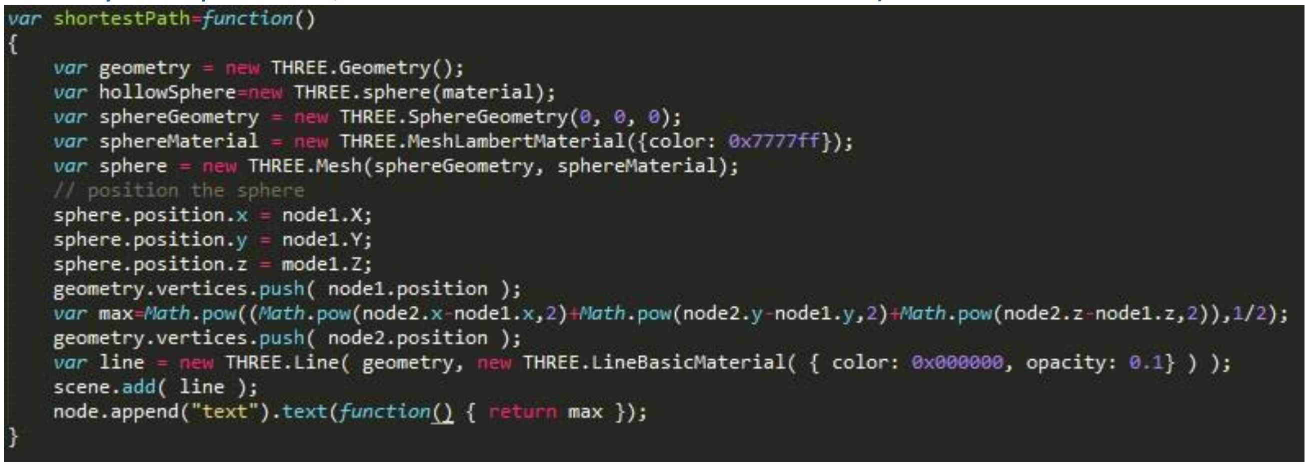
4.Assigning variables: It would always be better if we can assign specific particles of interest to a variable so that they may be called upon later on, in a sense be remembered by the computer for the long run. I might want to work on more detail under after I have done the up given jobs, as this requires using databases. Small variable assigning can be done using the d3.js library in a jiffy. How do I want it to look like:



To do this, we may use the following algorithm


```
node.append("text")
    .attr("dx", 12)
    .attr("dy", ".35em")
    .text(function(d) { return d.name });

force.on("tick", function() {
  link.attr("x1", function(d) { return d.source.x; })
      .attr("y1", function(d) { return d.source.y; })
      .attr("x2", function(d) { return d.target.x; })
      .attr("y2", function(d) { return d.target.y; });
```

This then allows us to search nodes by name, which is another part of our assignment. To highlight the shortest path, we may use the following algorithm (or we may use quadTree, but that is much less efficient here):


```
var shortestPath=function()
{
    var geometry = new THREE.Geometry();
    var hollowSphere=new THREE.sphere(material);
    var sphereGeometry = new THREE.SphereGeometry(0, 0, 0);
    var sphereMaterial = new THREE.MeshLambertMaterial({color: 0x7777ff});
    var sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);
    // position the sphere
    sphere.position.x = node1.X;
    sphere.position.y = node1.Y;
    sphere.position.z = mode1.Z;
    geometry.vertices.push( node1.position );
    var max=Math.pow((Math.pow(node2.x-node1.x,2)+Math.pow(node2.y-node1.y,2)+Math.pow(node2.z-node1.z,2)),1/2);
    geometry.vertices.push( node2.position );
    var line = new THREE.Line( geometry, new THREE.LineBasicMaterial( { color: 0x000000, opacity: 0.1} ) );
    scene.add( line );
    node.append("text").text(function() { return max });
}
```
We utilise the well known geometrical fact that the shortest distance between two points is the straight line distance between them. Also what it does is calculates the shortest distance between the particles and then returns it on screen.

5.Particle history: Generating a database and using it to store every particles history, including creation destruction and previous positions relative to some know particle etc. would obviously help visualising the actual biological use of the graph. If it is not a bioinformatics graph, but a biological network, this is an essential input.

Why am I the best person for this?

All the ideas above are my work. I am creative and have an idea about how data can be visualised and also have a niche above the general creed, in algorithms and their implementation. The ideas above are known in utmost detail to me, as I have worked upon them hard and long. Thus if they are to be implemented well, and are expected to work out as mentioned, I am pretty certain that no one else other than me can do it better.

Conclusion

I would like to say that the 7-8 leftover days would make up for work that was left pending and wasn't completed up to the previous deadline and also would give us the freedom to incorporate any interesting idea that might come up during the coding period (like the additional(s)) as it is not just about hanging in there and completing the job, but it is also about giving it your best. As it is said "Promise only what you can deliver, then deliver more than you promise". As the organization recently introduced its new WebGL renderer, using it as my basis would also be an interesting thing and I would like to work upon it. By this I rest my case, and would hope that this leads to the betterment of the Open Source Community as a whole. Also I would love to learn some Biology this summer, and hope it would be great.

Commitments this summer

I will be completing my examinations on 31st April, followed by the summer break till the end of July. I will be fully committed to working on this project, I have absolutely no school related activities or any other full time job or internship that will influence my progress with work. So, I am confident that I will be able to do well with this project and complete all the required work well in time. Also I can happily devote ***40 (forty) hours every week*** into the betterment of the project. Thanks for your time, looking forward to an interesting summer!