Fabric Engine

Point Distribution

Written by:

Narendra Kumar Vadapalli
narenandu@gmail.com

Context

With reference to the Software Challenge about Point Distribution to be implemented in Fabric Engine's Canvas Standalone application utilizing the KL language. Given the lack of a weekend and limited amount of time (after office hours and I have 2 year old daughter at home), was able to implement only two point distribution mechanisms: *Hammersley* and *Halton*.

Before reading the email about the software challenge, I had absolutely no idea what Fabric Engine and KL language was. Didn't even know what the mentioned point distribution mechanisms were. I will be detailing the report in chronological order illustrating the challenges I faced and embedding the results along with the code created.

Basic Requirements

- Should utilize KL within Fabric Canvas Standalone application (or Splice Standalone if you are using Fabric version < 2.0).
- Should experiment with at least two or three distribution mechanisms.
- Should utilize the parallel processing capabilities that are built into the KL language.

Approach

- Getting familiar with Fabric Engine
  - ✔ Using Canvas
  - ✔ KL Language
- Learn about Point Distribution algorithms
- Implement the algorithms in KL language

Implementation Details

Day 1: *21 Feb 2017*

- Downloaded the latest *FabricEngine-2.4.0-Linux-x86_64* on my laptop (running *Ubuntu 16.04* LTS with *KDE Plasma* flavor on top of it)
- Set up the environment and launched Canvas to see how to use the interface and different windows
- Searched for Fabric Engine documentation and Point Distribution algorithms on internet and keeping all the URLs ready (I use Google Keep as my note taking tool) for next day

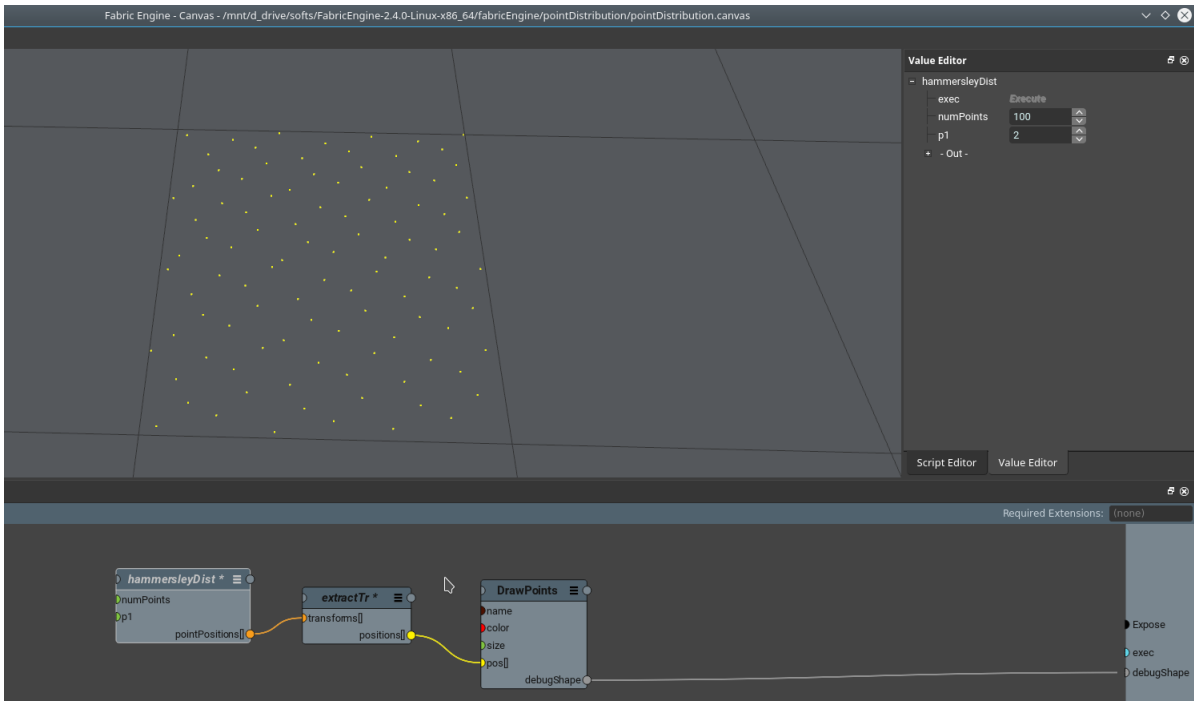
At this point of time I have no idea if I will be able to finish the task as I have no idea which algorithms to implement

Day 2: *22 Feb 2017*

- Wrote few basic scripts in KL by following the [Fabric Engine's documentation]:
- Opened up few samples from Fabric Engine's *Samples* folder in Canvas UI and learned about using various nodes to display simple shapes in the viewport.
- During the process learned about the containers to be used for holding the geometry and about Display Handles
- Studied the Halton Point Distribution algorithm from https://en.wikipedia.org/wiki/Halton_sequence
- Implemented the Halton and Hammersley distribution algorithms in KL (outside Canvas), put few *report* statements and checked for outputs using smaller number of samples
- Learned about writing KL in the function node of Canvas (difference between *dfgEntry* vs *Operator Entry()*

Day 3: *23 Feb 2017*

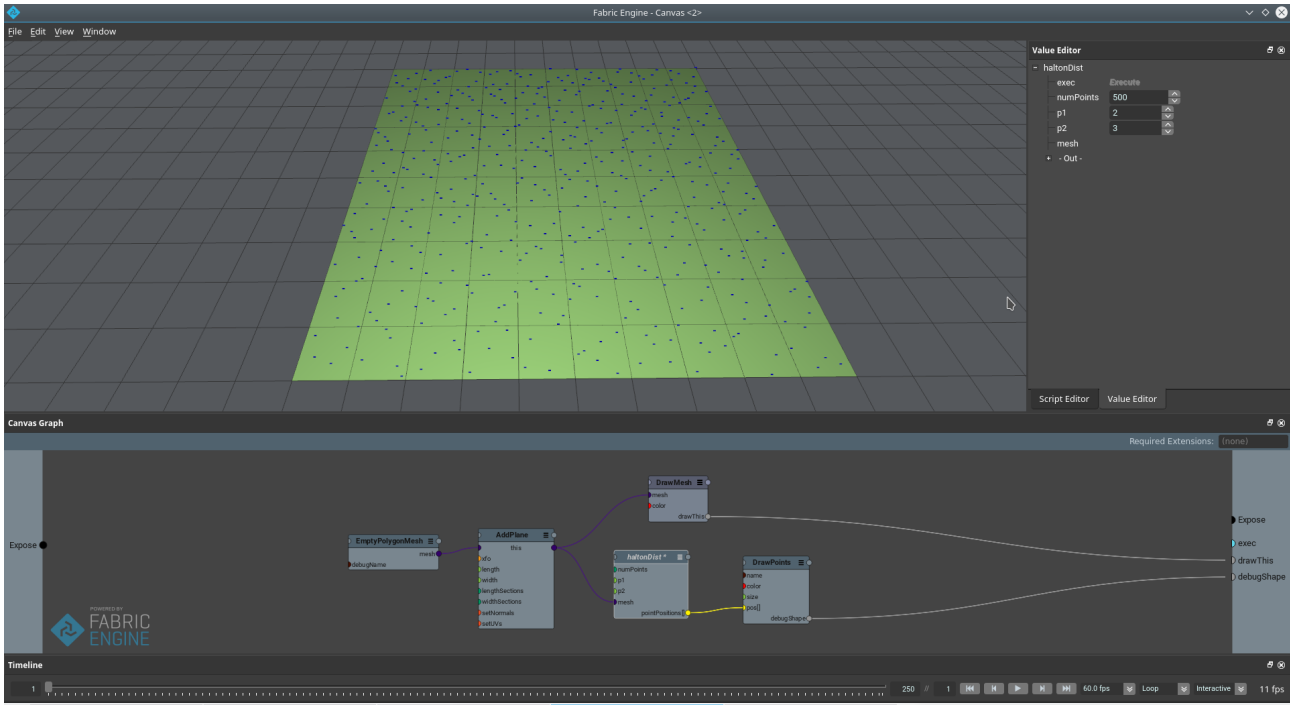
- Wrote the KL function inside Canvas for Hammersley and Halton method and hooked it up with DrawPoints node to see how it is working with in the range of (0,1). Example screenshot from this exercise

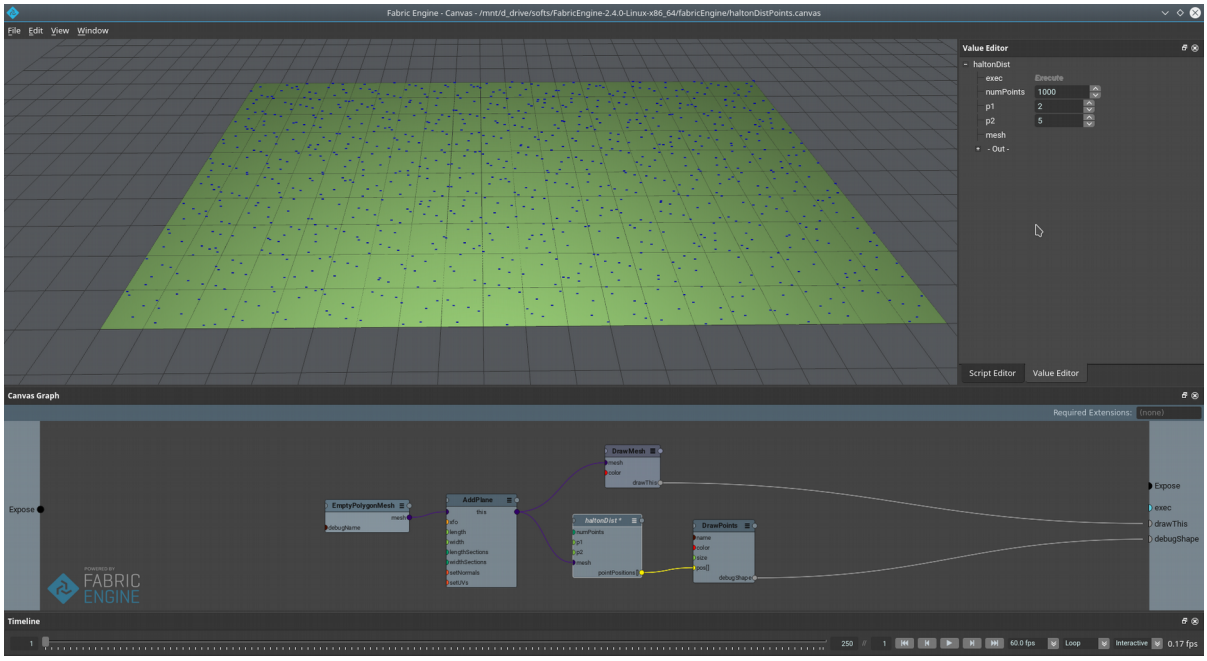


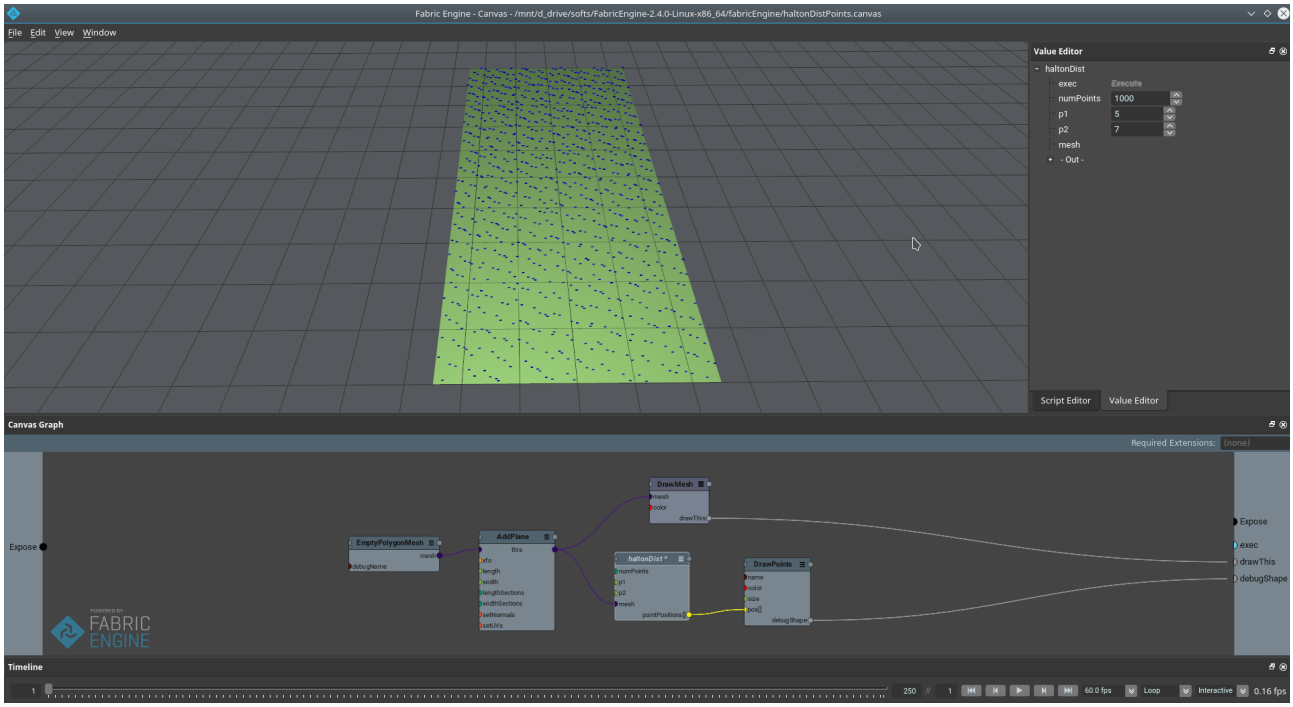
Day 4: *24 Feb 2017*

- Extended the written KL function nodes to take the plane's mesh as input and mapping the points distribution to the extents of the plane
- Following are the results from the exercise with different sizes of the plane and using different parameters for points distribution

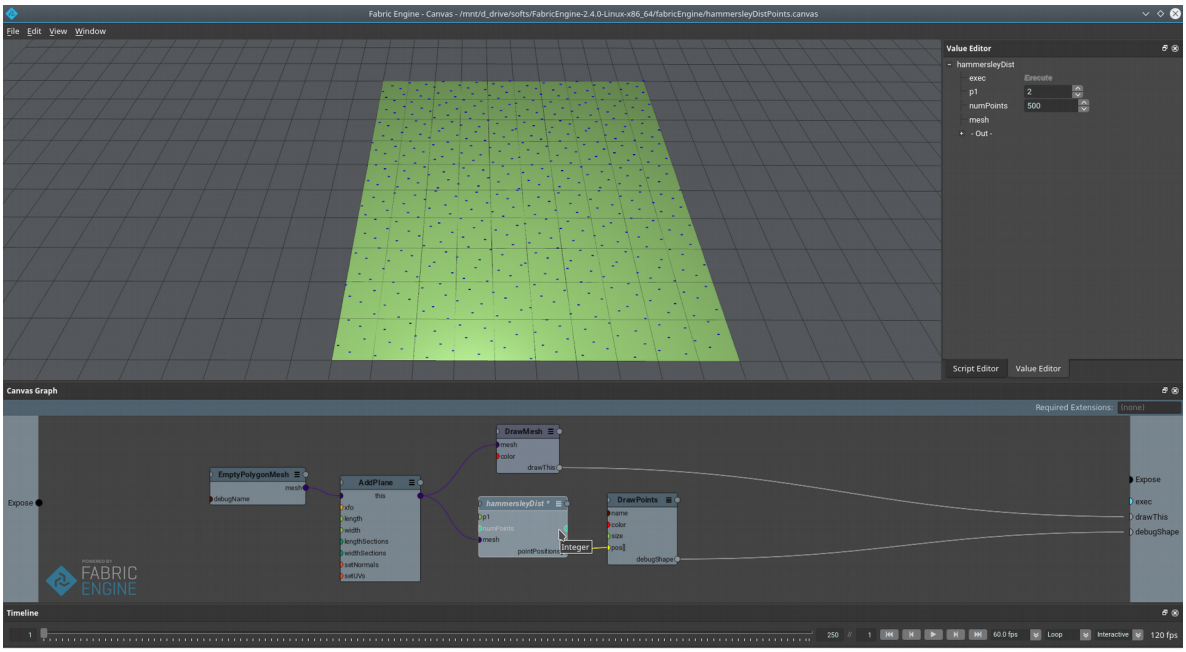
Halton Points Distribution

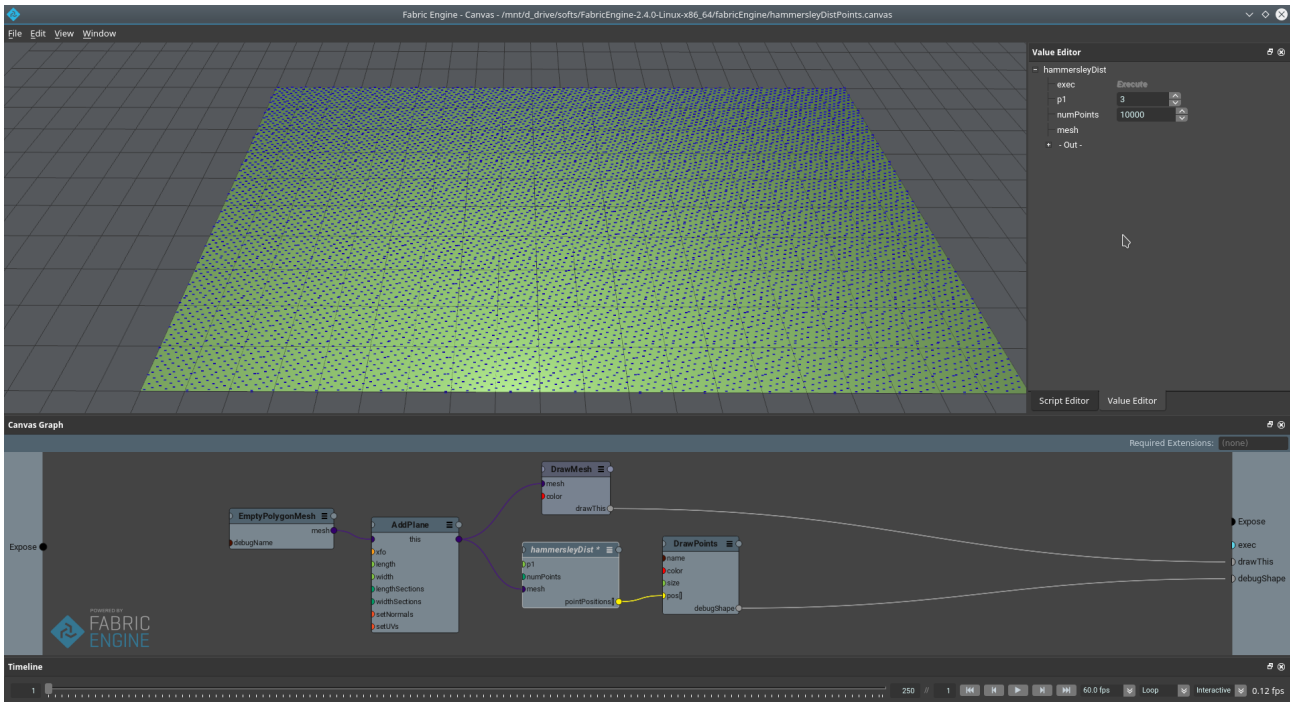


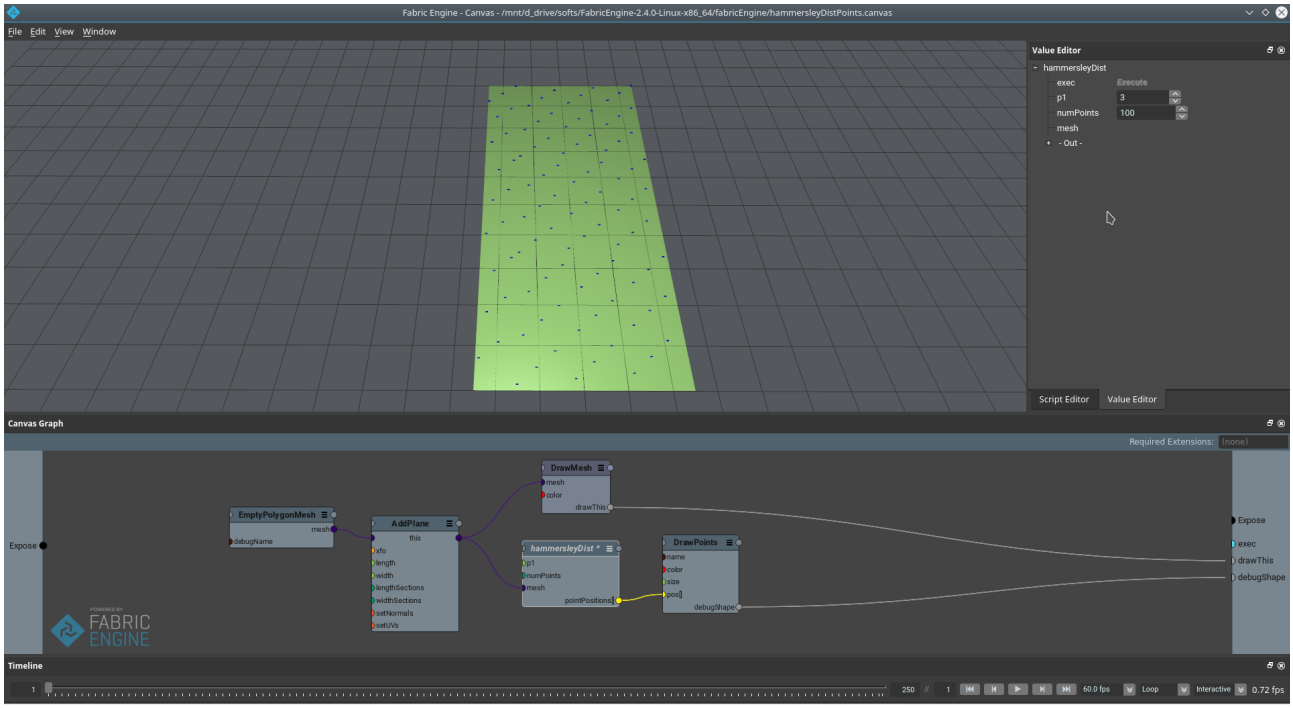




Hammersley Points Distribution

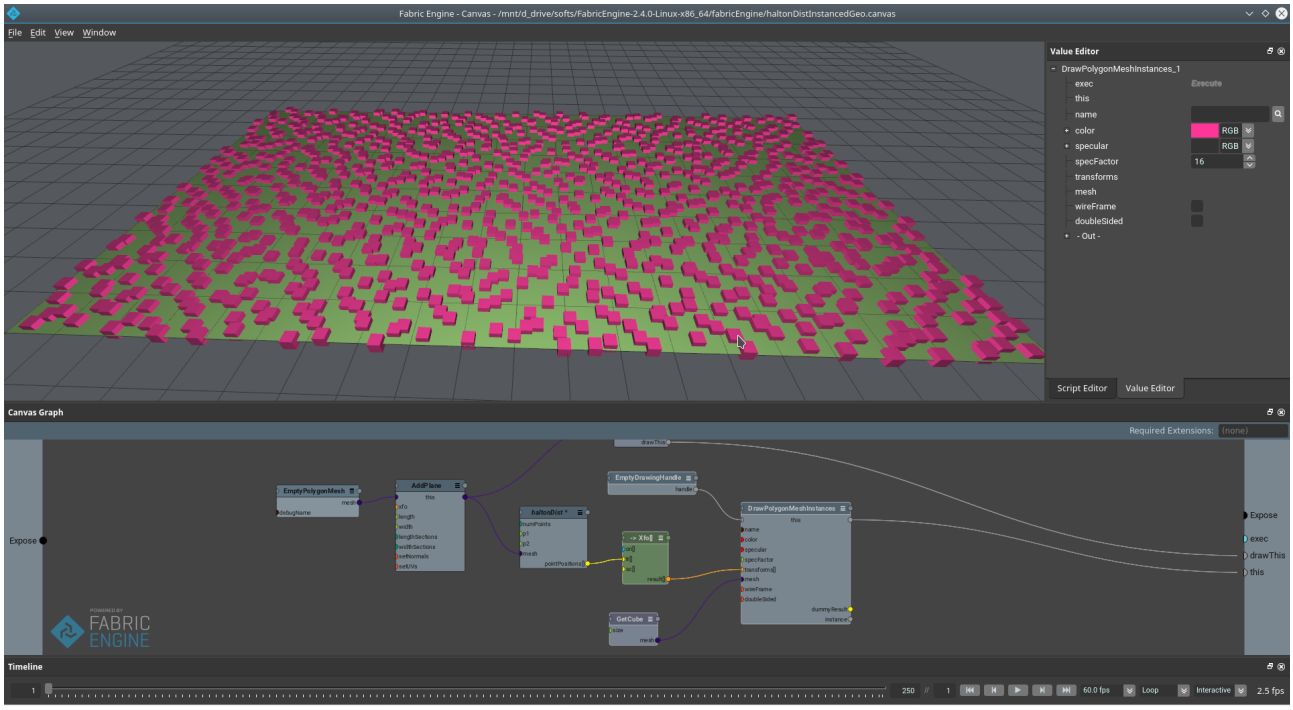


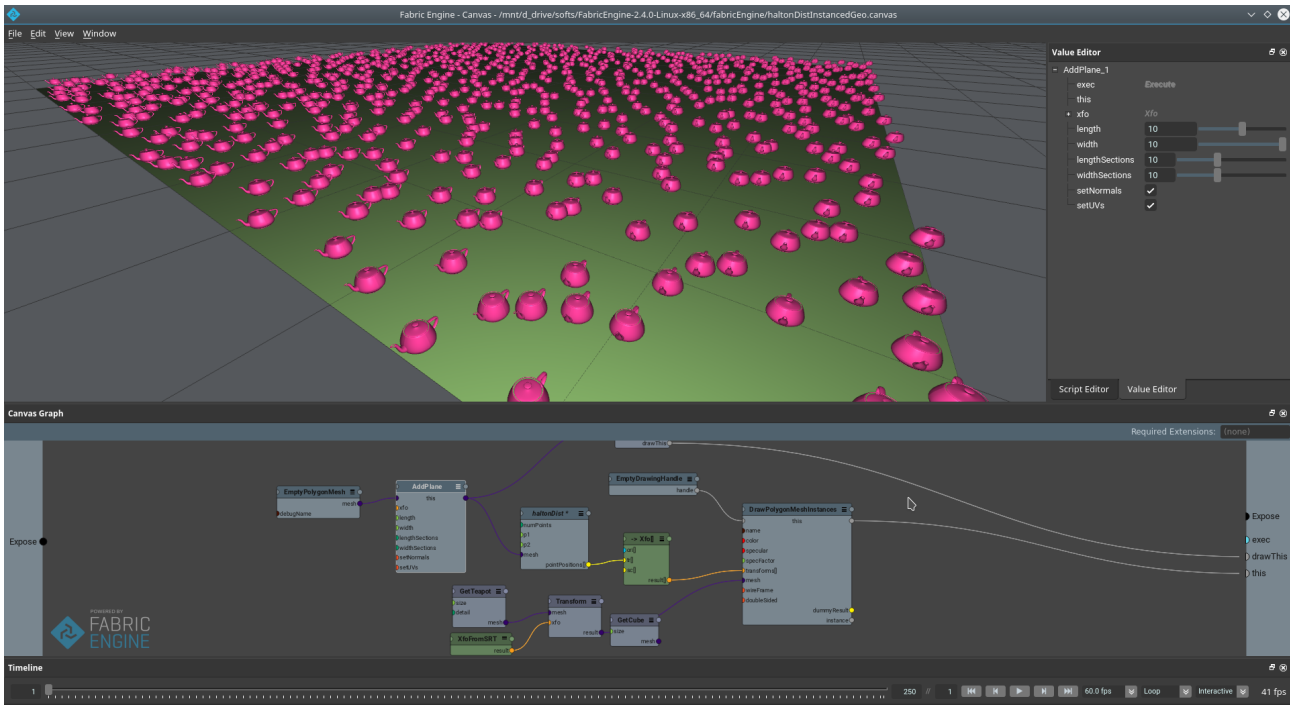

Fabric Engine - Canvas - /mnt/d_drive/softs/FabricEngine-2.4.0-Linux-x86_64/fabricEngine/hammersleyDistPoints.canvas
File Edit View Window
Value Editor
hammersleyDist
exec
Execute
p1
3
numPoints
10000
mesh
- Out -
Script Editor
Value Editor
Canvas Graph
Required Extensions:
(none)
DrawMesh
mesh
color
drawThis
EmptyPolygonMesh
mesh
debugName
AddPlane
this
xfo
length
width
lengthSections
widthSections
setNormals
setUVs
hammersleyDist *
p1
numPoints
mesh
pointPositions
DrawPoints
name
color
size
pos[]
debugShape
Expose
exec
drawThis
debugShape
FABRIC ENGINE
Timeline
60.0 fps
Loop
Interactive
0.12 fps


Fabric Engine - Canvas - /mnt/d_drive/softs/FabricEngine-2.4.0-Linux-x86_64/fabricEngine/hammersleyDistPoints.canvas
File Edit View Window
Value Editor
hammersleyDist
exec
Execute
p1
3
numPoints
100
mesh
- Out -
Script Editor
Value Editor
Canvas Graph
Required Extensions:
(none)
DrawMesh
mesh
color
drawThis
EmptyPolygonMesh
mesh
debugName
AddPlane
this
xfo
length
width
lengthSections
widthSections
setNormals
setUVs
hammersleyDist *
p1
numPoints
mesh
pointPositions
DrawPoints
name
color
size
pos[]
debugShape
Expose
exec
drawThis
debugShape
FABRIC ENGINE
Timeline
60.0 fps
Loop
Interactive
0.72 fps

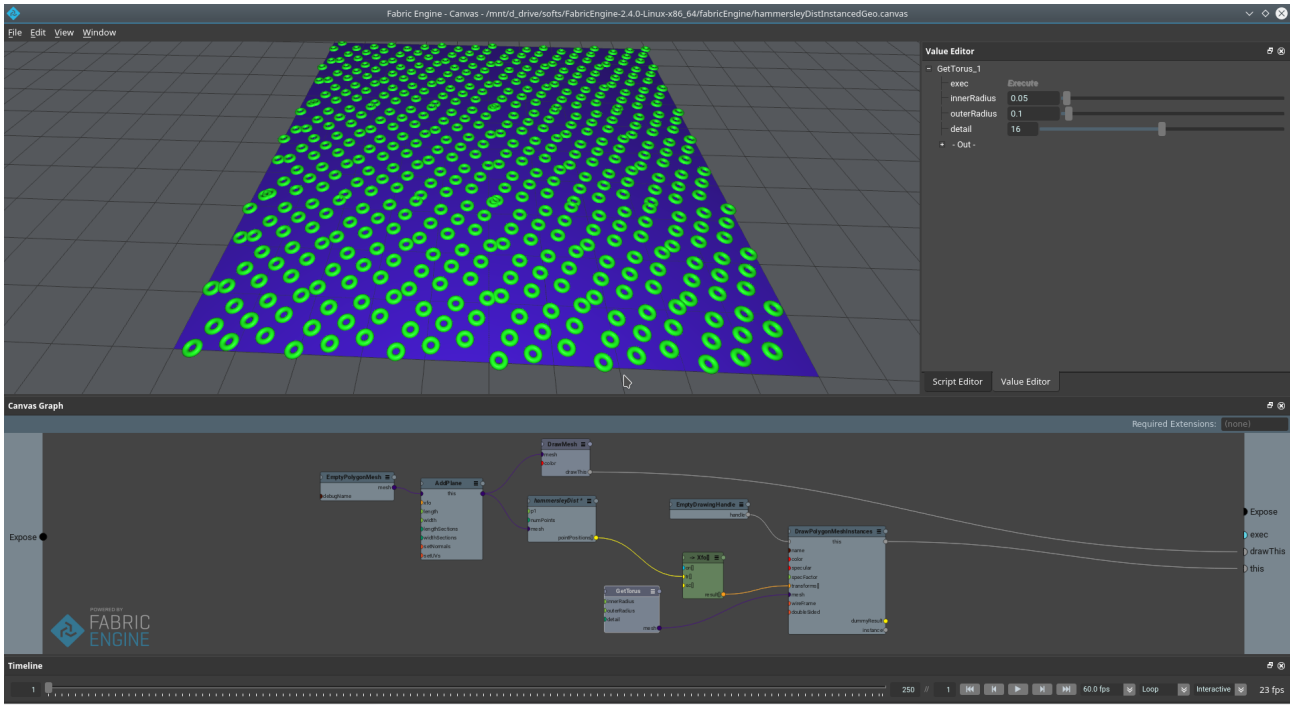
- Experimented with distributing the geometry instances on to the distributed points using *DrawPolygonMeshInstances* node and following are few results

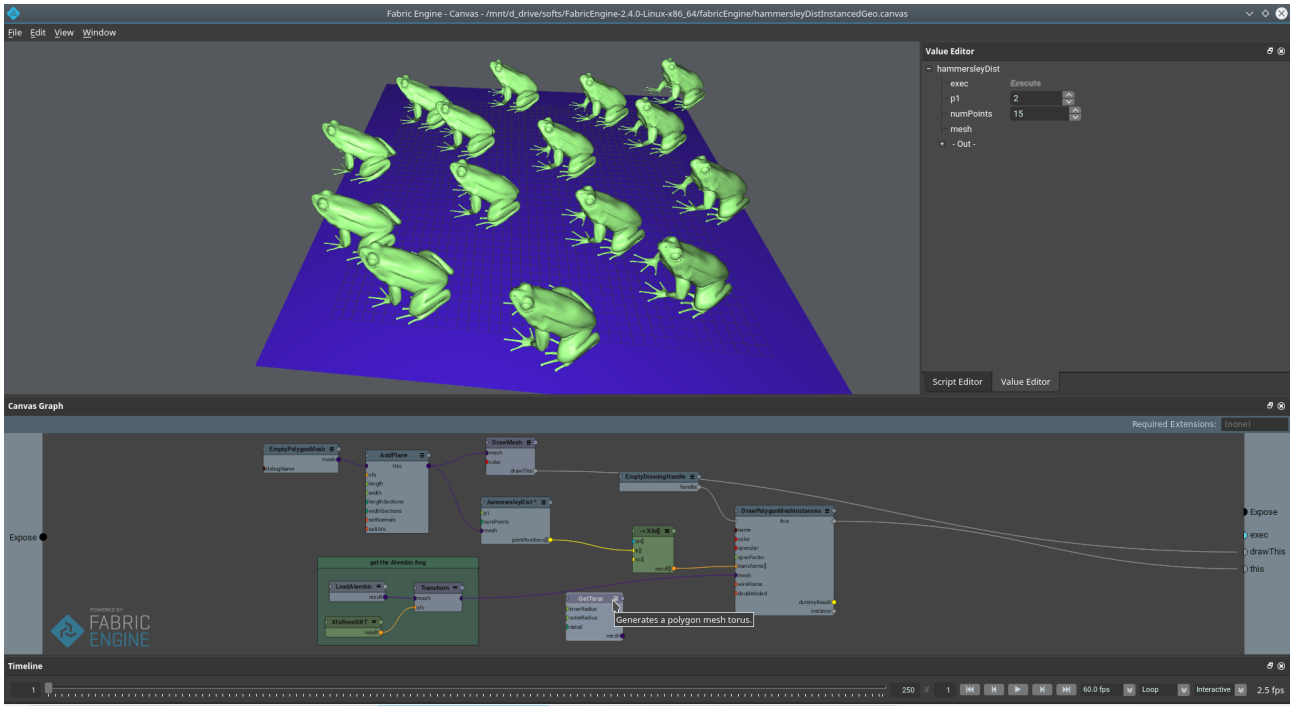
Halton Points Distribution with Geo Instances





Hammersley Points Distribution with Geo Instances





Day 5: *25 Feb 2017*

- Early morning was the time to prepare this report and send it across. Kept aside an hour to prepare the report along with the captured screenshots from yesterday

Advantages at the current state

- Points Distribution can fit on to any size of the Plane given as input

Disadvantages at the current state

- Considering only the positions on the plane without the orientation would not orient the geometry on a deformed mesh

Code: Halton Point Distribution KL Function Node

```
/*
    numPoints       In   Integer
    p1              In   Integer
    p2              In   Integer
    mesh            In   PolygonMesh
    pointPositions  Out  Vec3[]
*/

operator calcPositions<<<index>>>(io Vec3 pointPositions[], Scalar p1, Scalar p2,
PolygonMesh mesh){
    Scalar p, x, y, inverseP1, inverseP2;
    Integer i, a;

    // Get the Mesh Length, Width, BboxMin and BBoxMax
    Vec3 bBoxMax = mesh.getBoundingVolume().bBoxGetMax();
    Vec3 bBoxMin = mesh.getBoundingVolume().bBoxGetMin();
    Scalar bBoxLen = bBoxMax.x - bBoxMin.x;
    Scalar bBoxWid = bBoxMax.z - bBoxMin.z;

    x = 0;
    inverseP1=1.0/p1;
    // calculating the halton Seq for p1
    for(p=inverseP1, i=index ; i ; p*=inverseP1, i*=inverseP1)
        if(a = i % p1)
            x += a * p;
    x = bBoxMin.x + bBoxLen * x;

    y = 0;
    inverseP2 = 1.0/p2;
    // calculating the halton Seq for p2
    for(p=inverseP2, i=index; i ; p*=inverseP2, i*=inverseP2)
        if (a = i % p2)
            y += a * p;
   // map it to the plane dimensions in y
    y = bBoxMin.z + bBoxWid * y;

    // populate the x, y
    pointPositions[index] = Vec3(x, 0, y);
}

dfgEntry {
    //sanity Checks
    if (   numPoints <= 0
        || mesh == null
        || p1 <= 0
        || p2 <= 0)
      {
        pointPositions.resize(0);
        return;
      }

    pointPositions.resize(numPoints);
    calcPositions<<<numPoints>>>(pointPositions, p1, p2, mesh);
}
```

Code: Hammersley Point Distribution KL Function Node

```
/*
    p1              In   Integer
    numPoints       In   Integer
    mesh            In   PolygonMesh
    pointPositions  Out  Vec3[]
*/

operator calcPositions<<<index>>>(io Vec3 pointPositions[], Scalar p1, PolygonMesh mesh,
Integer numPoints) {
    Scalar p, x, y, inverseP;
    Integer i, a;

    // Get the Mesh Length, Width, BboxMin and BBoxMax
    Vec3 bBoxMax = mesh.getBoundingVolume().bBoxGetMax();
    Vec3 bBoxMin = mesh.getBoundingVolume().bBoxGetMin();
    Scalar bBoxLen = bBoxMax.x - bBoxMin.x;
    Scalar bBoxWid = bBoxMax.z - bBoxMin.z;

    x = 0;
    inverseP = 1.0/p1;
    // calculating the hammersley Seq
    for (p=inverseP, i=index ; i ; p*=inverseP, i*=inverseP)
       if (a = i % p1)
           x += a * p;
    // map it to the plane dimensions in x
    x = bBoxMin.x + bBoxLen * x ;

    y = (index + inverseP) / numPoints;
   // map it to the plane dimensions in y
    y = bBoxMin.z + bBoxWid * y;

    // populate the x, y
    pointPositions[index] = Vec3(x, 0, y);
}

dfgEntry {
    //sanity Checks
    if (numPoints <= 0
        || mesh == null
        || p1 <= 0)
      {
        pointPositions.resize(0);
        return;
      }

    pointPositions.resize(numPoints);
    calcPositions<<<numPoints>>>(pointPositions, p1, mesh, numPoints);
}
```

Future Plans

- To extend the code further
  - to consider the normals after using *turbulize* node in Canvas and calculate the orientations accordingly
  - to use *DrawTexturedPolygonInstances* if there was any texture to be used for specifying the density of the points distribution

Final Thoughts

- Fell in love with this wonderful application called Fabric Engine which triggered me to learn more interms of the simulations and procedural stuff which can be done.
- Three days before I have first opened up the website about Fabric Engine and by the time I am finishing up this report, quite confident of implementing anything given considerable amount of time

-Narendra Kumar Vadapalli