PROJECT REPORT

PROJECT TITLE :

REAL TIME TERRAIN RENDERING USING DATA STRUCTURES

SUBMITTED BY:

Amol Samota
Mining Engineering
18155009

ACKNOWLEDGEMENT

It is our privilege to express our gratitude to several people who helped us directly or indirectly to conduct this research project work. We express our heartfelt indebtedness and owe a deep sense of gratitude to our teacher and faculty guide Prof. Rajeev Shrivastva for their guidance and knowledge to complete this project.

The study has indeed helped us to explore more knowledgeable avenues related to the topic and we are sure it will help us in our future.

INTRODUCTION:

ABOUT:

Terrain rendering is a technique or a set of methods to depict real world or imaginary world surfaces. It basically gives us a blueprint of the terrain, giving us information about how undulating, uneven and wavy a terrain is. Now lets see the basic structure of a terrain rendering model:

The basic structure of a terrain rendering application consist of 4 main components:

1. **Software Application:** It is actually the main assembly place where we run our program .In our case we have used Visual Studio 2019 for implementation. The main input file i.e the terrain data to be rendered is loaded through this application.
2. **CPU:** The software application uses the CPU to identify and load terrain data corresponding to initial location from the terrain database, then applies the required changes to build a mesh of points that can be rendered by the GPU.
3. **GPU:** completes geometrical transformations, creating screen space objects (such as polygons) that create a picture closely resembling the location of the real world.
4. **Display:** this component is responsible for displaying our result of the rendered terrain.

TERRAIN GENERATION:

For static terrain generation we simply have to look for all the data points (more the data points, more accurate rendering) but for moving terrain generation and rendering, we need to optimize the source data and that can be done by using level of detail (LOD) to manage data points processed by CPU and GPU.

LEVEL OF DETAIL:

LOD consists of decreasing the complexity of a terrain representation as it moves away from the viewer. Level of detail techniques increase the efficiency of rendering by decreasing the workload on GPU stages. LOD also depends on the relative speed and position of viewpoint.
Level of detail is mainly of two types:

1. Discrete level of detail (DLOD) : It is a traditional approach that creates LOD for each object separately while pre-processing. At runtime it picks each object according to a specific selection criteria.

Advantages:

- Requires simple programming
- Fits in modern graphics
- Easy implementation

Disadvantages:

- Not suitable for large data
- Not much accurate and large memory requirements

2. Continuous level of detail (CLOD) : It creates data structure from which a desired level of detail can be extracted at run time. It is a newer approach.

Advantages:

- Objects do not use more polygons than necessary. Therefore, it has a better memory utilisation.
- We can also locally vary the level of detail parameters.

Terrain-Rendering Algorithms can be classified into :

A number of techniques have been proposed for rendering terrains, we classify them into three categories: **Hierarchical algorithms**, that recursively subdivide the heightmap using a common data structure; **Triangular irregular meshes**, where the triangles can be of any shape and size to give the most faithful representation of the terrain; and **GPU-based approaches**, that cache vertices or triangles on the graphics card, so they can be rendered efficiently.

HIERARCHICAL ALGORITHMS:

Hierarchical terrain-rendering algorithms recursively subdivide the heightmap into a primitive shape, resulting in a hierarchy. Various shapes can be used; the only requirement being that one instance of the shape can be partitioned into s smaller copies of the same shape. Subdivisions may introduce cracks in the terrain, which can be prevented by adding or discarding vertices, or by subdividing further on an adjacent shape. Real-time Optimally Adapting Meshes (ROAM) and QuadTrees
are two common examples of hierarchical techniques.

Advantages -:

- They allow adaptive refinement, where a part of the terrain can be represented with more or fewer triangles as required, and this structure can change rapidly as the viewpoint moves
- Produce accurate approximations of the terrain, using considerably fewer triangles
- Optimal Triangulation not only makes rendering simple but once a triangulation is generated, it is also beneficial for view-frustum culling.

Disadvantages -:

- Adaptive refinement requires a number of CPU
- It uses a significant amount of memory to track the current state of the triangulation.

TRIANGULAR IRREGULAR NETWORKS:

TINs (Triangular Irregular Networks) represent the terrain as a number of triangles of different shapes and sizes. A TIN comprises a triangular network of vertices, known as mass points, with associated coordinates in three dimensions connected by edges to form a triangular tessellation.TIN are based on a Delaunay triangulation or constrained Delaunay.

Disadvantages -:

- Highly CPU intensive and require a considerable amount of memory to model the mesh
- Geometry caching is difficult, due to the irregular structure
- TIN restricts the possible triangulations, because discarding triangles having vertex inside them results in a less accurate representation.

GPU-BASED ALGORITHMS:

These techniques, sets of triangles, known as clusters, chunks or aggregate triangles are cached on the graphics card and rendered together. Examples of the algorithms that used this approach is Ulrich's Chunked LOD algorithm. These algorithms use the same process and data structures as the hierarchical techniques, but where a single primitive was rendered, a cluster would be displayed instead.

Advantages -:

- By caching regions in video memory rendering quality is improved.
- They render more triangles.

Disadvantages -:

- Care needs to be taken to prevent popping, a sudden change in the terrain geometry when a patch's level of detail changes.

METHODS IMPLEMENTED:

1. A simple LOD algorithm that we implemented is delaunay triangulation, here the technique was simple - keep on adding triangles till the required level of detail is reached. Since level of detail depends on the distance between the viewpoint and the terrain, lesser distance implies more LOD, thus it gives us a 2D blueprint of our terrain.

2. Another method we used is Real time optimally adapting meshes (ROAM) algorithm which is based on a binary triangle tree data structure. Here each patch is made up of an isosceles right triangle. The triangle is split recursively into children and the process is repeated until the required level of detail is reached.

Below is a literature review about the methods available and why we implemented above two methods for terrain rendering.

→ LITERATURE REVIEW :

Name of different approaches	Brief Concept	Advantages	Disadvantages
Delaunay Triangulation First Proposed by Boris Delaunay	**Delaunay triangulation** for a given set **P** of discrete points in a plane is a triangulation DT(**P**) such that no point in **P** is inside the circumcircle of any triangle in DT(**P**). Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid silver triangles. Looking at two triangles ABD and BCD with the common edge BD (see figures), if the sum of the angles α and γ is less than or equal to 180°, the triangles meet the Delaunay condition. If two triangles do not meet the Delaunay condition, switching the common edge BD for the common edge AC produces two triangles that do meet the Delaunay condition.	The triangles are as equi-angular as possible, thus reducing potential numerical precision problems Ensures that any point on the surface is as close as possible to a node The triangulation is independent of the order the points are processed	Delaunay triangulations impose a constraint on TINs , resulting in a less accurate representation. It do not utilise the GPU, requires extensive use of the CPU, and use a large amount of memory
Real-time Optimally Adapting Meshes (ROAM) First proposed by LindStrom	A ROAM triangulation contains only right-angled isosceles triangles. Each triangle can be split into two right-isosceles triangles of half the size, by dividing along a line from the apex to the midpoint of the hypotenuse. An inverse process exists where triangles are merged. The resulting binary tree (or BinTree) of triangles is stored in memory. Priority queues are used to manage which triangle is most in need of a split or merge operation.	Better performance can be obtained which can make them more desirable for games applications , as all detail levels, can all be uploaded to the graphics memory at initialisation time	Difficult to form large triangle strips, which can be rendered more quickly, since the triangulation is not uniform

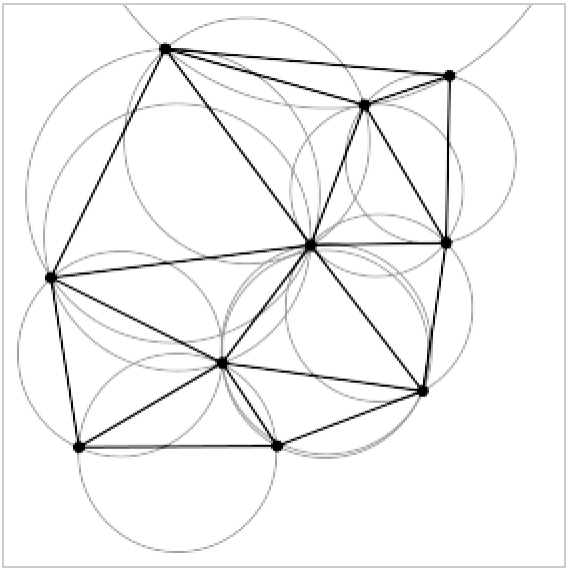
QuadTree First proposed by Raphael Finkel and J.L. Bentley	QuadTrees are based on a similar idea to ROAM, but rectangles, or preferably squares, are used. Each square can be split into four squares of quarter the size, resulting in a QuadTree of squares. To prevent T-junctions, we enforce the constraint that no two adjacent squares differ by more than one level of detail, and when adjacent squares do differ by a single level, a single vertex is discarded.	They also produce accurate approximations of the terrain, using considerably fewer objects. Squares or Rectangles are very easy to traverse. This makes rendering quite simple	Adaptive refinement requires a number of CPU computations to calculate where more or less detail is necessary, and a significant amount of memory to track the current state of the tessellation.
Geometrical mip-mapping Proposed by Losasso and Hoppe	Geometric mipmapping (geomipmapping), a geometry equivalent of texture mipmapping. With geomipmapping, the terrain is divided up into a regular grid, called a patch or tile. Each patch can be rendered at several levels of detail; the LOD is varied by changing the space between grid lines. The tiles can be cached in vertex buffers on the graphics card for fast access, and can be rendered efficiently using triangle strips. Cracks between adjacent tiles of different levels of detail can be avoided by adding or removing vertices.	It provides a steady rendering rate, even when the viewpoint is moving quickly. Each patch can be rendered at several levels of detail. Cracks between adjacent tiles of different levels of detail can be avoided by adding or removing vertices.	To prevent popping, a sudden change in the terrain geometry when a patch's level of detail changes. Using an appropriate metric can help, but is not usually enough for a high-quality rendering.

DELAUNAY TRIANGULATION

INTRODUCTION:

Delaunay triangulation for a given set of discrete points in a plane is a triangulation such that no point in the set is inside the circumcircle of any triangle in the triangulation. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation.

For a set of points on the same line there is no Delaunay triangulation (the notion of triangulation is degenerate for this case). For four or more points on the same circle (e.g., the vertices of a rectangle) the Delaunay triangulation is not unique: each of the two possible triangulations that split the quadrangle into two triangles satisfies the "Delaunay condition", i.e., the requirement that the circumcircles of all triangles have no points in them.



A Delaunay triangulation in the plane with circumcircles shown

SOFTWARE REQUIREMENTS:

- Microsoft Visual Studio 2019 for implementation

- Simple Fast Multimedia Library (SFML) for graphics and rendering purposes.

WORKING OF ALGORITHM:

Bowyer watson is a very effective implementation of the Delaunay triangulation.

The Bowyer–Watson algorithm is an incremental algorithm. It works by adding points, one at a time, to a valid Delaunay triangulation of a subset of the desired points. After every insertion, any triangles whose circumcircles contain the new point are deleted, leaving a polygonal hole which is then re-triangulated using the new point. By using the connectivity of the triangulation to efficiently locate triangles to remove, the algorithm can take ***O(N log N)*** operations to triangulate N points, although special degenerate cases exist where this goes up to $O(N^2)$

The following algorithm describes a basic implementation of the Bowyer-Watson algorithm. Its time complexity is **O(n^2)** Efficiency can be improved in a number of ways. For example, the triangle connectivity can be used to locate the triangles which contain the new point in their circumcircle, without having to check all of the triangles - by doing so we can decrease time complexity to **O(nlog n)**.Pre-computing the circumcircles can save time at the expense of additional memory usage.

- The algorithm starts by **finding a super triangle**, which is simply a triangle which contains all points in our dataset. We add this triangle to our current triangulation, it actually initializes our process.

- We **select each point** from our set of points one by one and add it to our triangulation by following the condition that the given point does not lie inside the circumcircle for each triangle in our triangulation.We create a set of bad triangles, a bad triangle is one whose circumcircle contains the given point.

- Now we **make a polygon set** which contains the set of edges to be finally rendered on the screen through triangulation. We traverse all edges of the triangulation and if the edge is not shared by any of the bad triangles we add it to the polygon set.

- Next step involves **removal of all bad triangles** from triangulation, then we finally connect all the edges in the polygon set to the given point and add these newly formed triangles to our triangulation.At last we also remove any vertex from super triangle if present, since it was only used for initialisation and was not a part of original dataset.

PSEUDOCODE -:

function BowyerWatson (pointList)

// pointList is a set of coordinates defining the points to be triangulated

triangulation := empty triangle mesh data structure

add **super**-triangle to triangulation *// must be large enough to completely contain all the points in pointList*

for each point **in** pointList **do** *// add all the points one at a time to the triangulation*

badTriangles := empty set

for each triangle **in** triangulation **do** *// first find all the triangles that are no longer valid due to the insertion*

if point is inside circumcircle **of** triangle

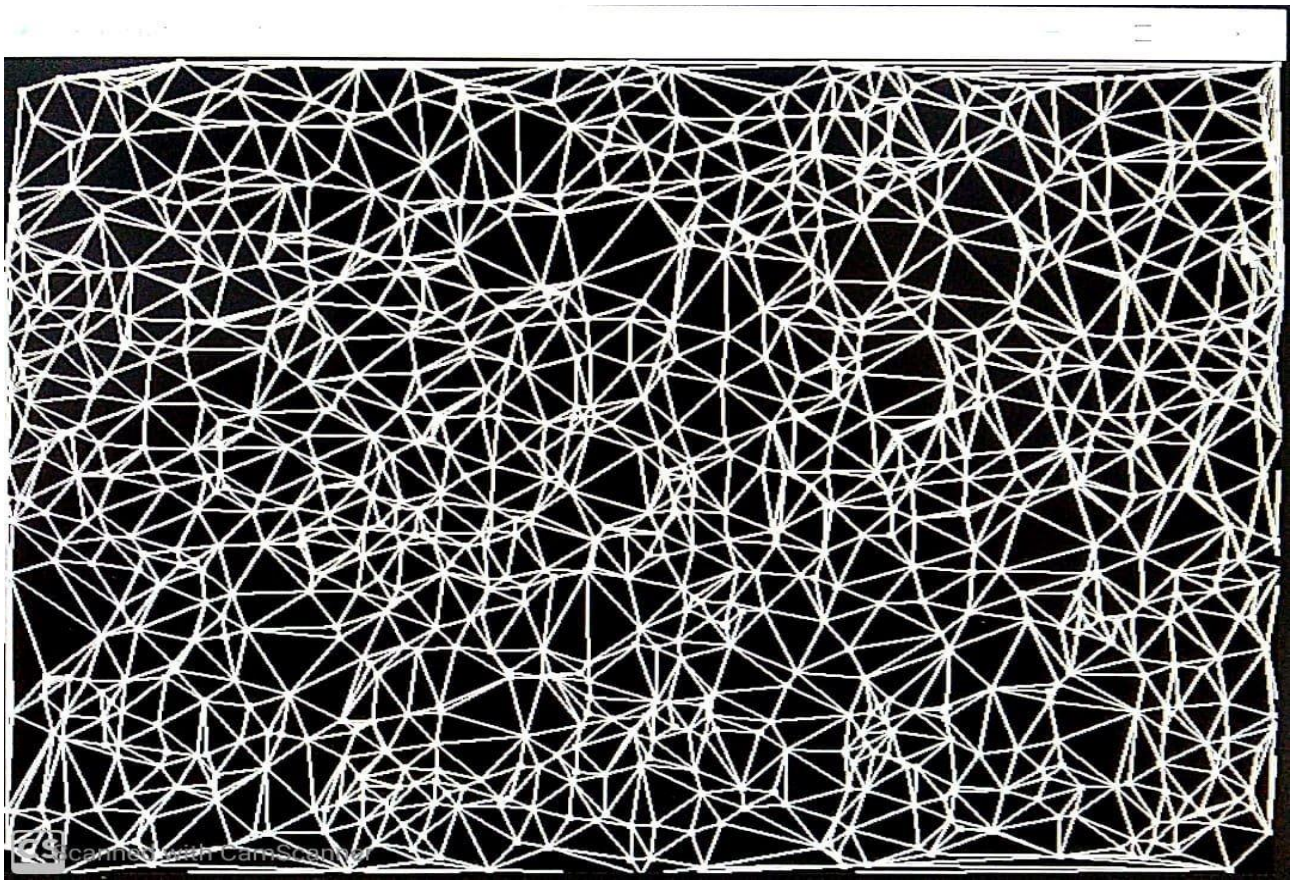
add triangle to badTriangles

polygon := empty set

for each triangle **in** badTriangles **do** *// find the boundary of the polygonal hole*

```
        for each edge in triangle do
            if edge is not shared by any other triangles in badTriangles
                add edge to polygon
        for each triangle in badTriangles do // remove them from the data structure
            remove triangle from triangulation
        for each edge in polygon do // re-triangulate the polygonal hole
            newTri := form a triangle from edge to point
            add newTri to triangulation
    for each triangle in triangulation // done inserting points, now clean up
        if triangle contains a vertex from original super-triangle
            remove triangle from triangulation
return triangulation
```

RESULT: Below is the screenshot of 10000 triangles which were generated using this algorithm.



CONCLUSION:

So we implemented the delaunay triangulation method which is very useful to form triangular mesh from a given set of points. Delaunay triangulation has proved to be very useful in understanding the level of detail to the simplest level and then advancing for more complex datasets and methods by introducing new data structures. This method was also of extensive use in computer networks and graphics although presently its use has now been limited to simpler graphics and smaller networks.

REAL TIME OPTIMALLY ADAPTING MESHES (ROAM) ALGORITHM

INTRODUCTION:

Real time optimally adapting mesh is a continuous level of detail algorithm that is used for terrain visualization by performing tessellation until required level of detail is achieved. The algorithm is very useful for creating a 3D dynamic model of a terrain which is very useful in modern day gaming technologies

This algorithm is based on a binary triangle tree data structure. Here each patch is made up of an isosceles right triangle. The triangle is split recursively into children and the process is repeated until the required level of detail is reached. Now the method to tessellate, taking input, working of algorithm and error calculations are discussed below.

SOFTWARE REQUIREMENTS:

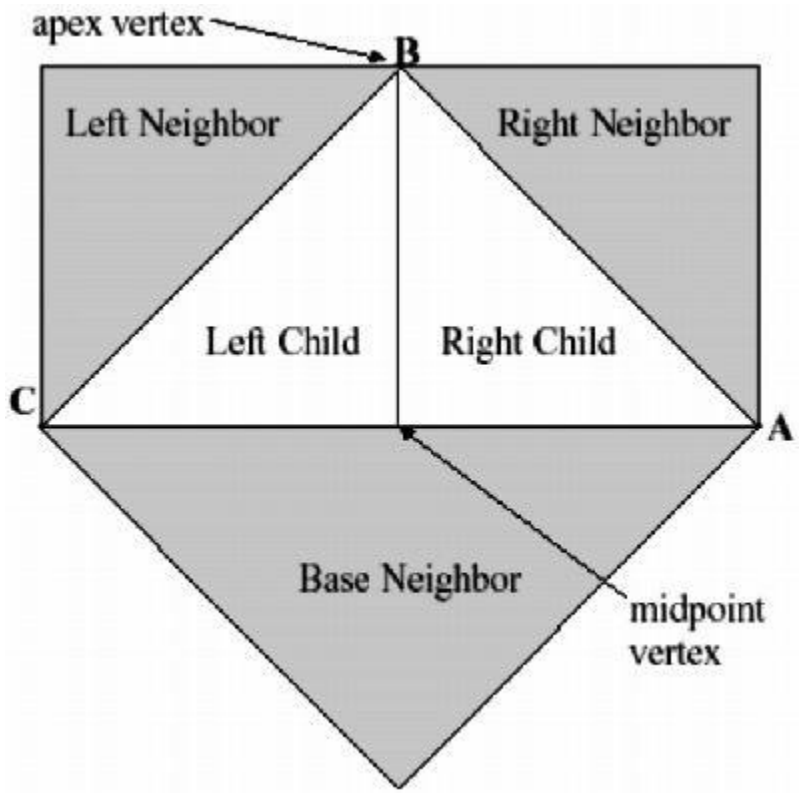
- A software application, here i have used Visual Studio 2019.
- OpenGL library and glut library installed in it.

WORKING OF ALGORITHM:

- We start by **uploading the heightmap** or the raw image into the program which is to be rendered, this is done with the help of the openGL library.

- Then we create a **landscape class** whose purpose is to pass the input heightmap in small sections or patches because for each section we have to create a binary triangle tree as well as the variance tree (for error calculations), now if we pass a large area patch, memory requirements will increase and construction of variance tree will also take a lot of time.

- Then we create a **binary triangle tree** in landscape class to store the new nodes being added to our mesh each time we render a patch. Landscape class is responsible for passing triangle coordinates (x,y,z) in the mesh to be rendered.



- The patch object receives data from landscape class. The first task of patch object is to calculate error in mesh before tessellation and this is done by calculating a quantity called **variance** which is basically an error parameter defined as the difference in height of interpolated hypotenuse midpoint and actual height field sample at that point. More the variance, more is the error, so more splitting of binary triangle trees at that point, thus achieving the appropriate level of detail.

- Since variance depends on the heightmap, in order to save memory consumption we create a **variance tree** alongside a binary triangle tree to get the variance data at each level of the tree.

- After resolving errors in mesh approximation, another problem that can occur is **cracks in our mesh** which occur due to uneven splitting of the binary triangle tree at a point. To resolve this the current node and the neighbour must point to each other, forming a diamond.So while splitting a node, if it forms a diamond or it lies on edge, we split it normally else we keep on force splitting from midpoint of hypotenuse until we reach edge or we get a diamond (two nodes pointing at each other)

- Now we finally perform **tessellation** (tiling the area using triangles) by splitting the nodes according to the variance to achieve desired LOD and doing forced splitting to avoid cracks in mesh. Tessellation also depends on the viewpoint of the camera, closer nodes have large variance so more splitting. The tessellation process keeps on repeating until all triangles are in their variance limit or we run out of nodes.

- We have the required information for **rendering**, we traverse all the triangle nodes once and if all the leaf nodes of the binary triangle tree are finally rendered on the screen with the help of openGL library.

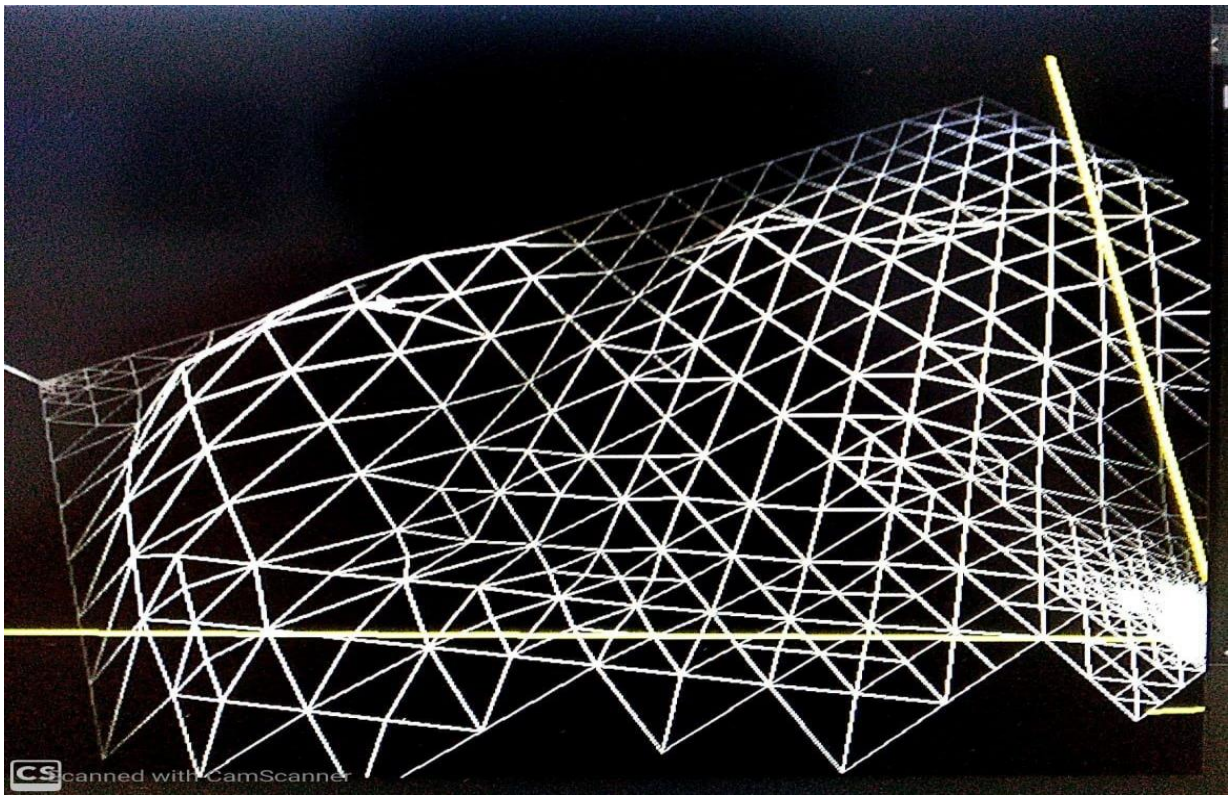
RESULT:

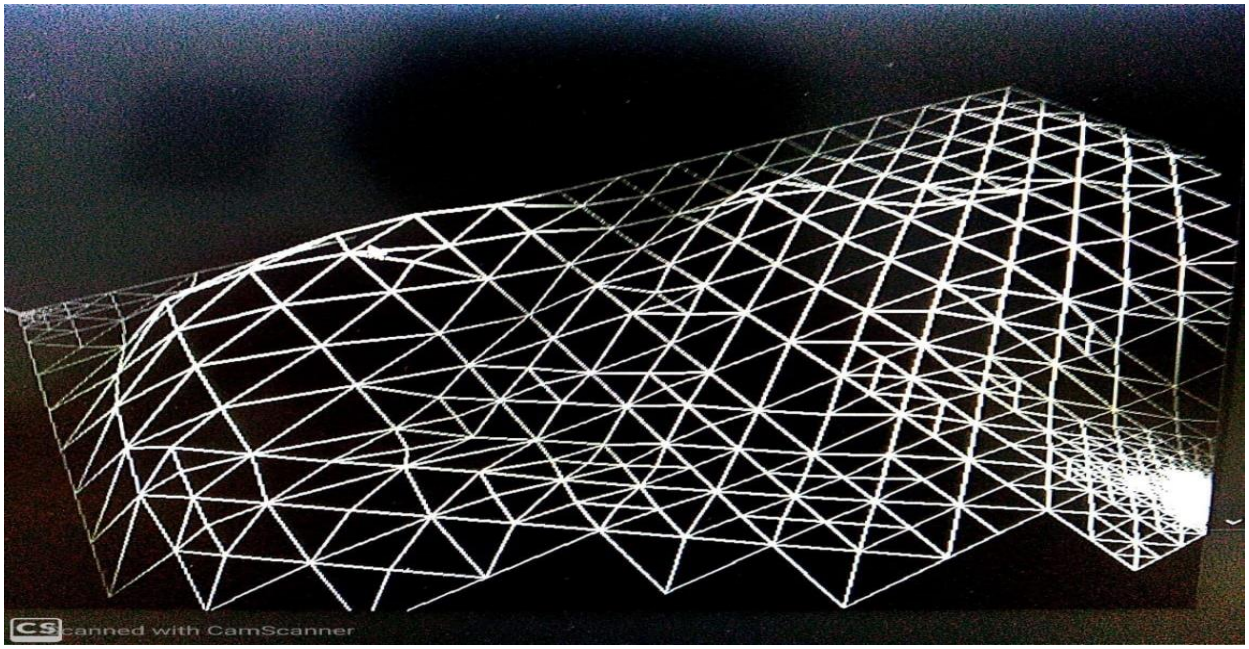
To test the final implementation of the system, a sample height map was used and frame rates were observed. A heightmap of size was chosen for these tests, and thus provided just over 100,000 potentially renderable triangles in the mesh. The system used for the tests was a typical desktop system; 2.0 GHz CPU, 4 GB RAM with a NVIDIA GeForce MX250 GPU.

.

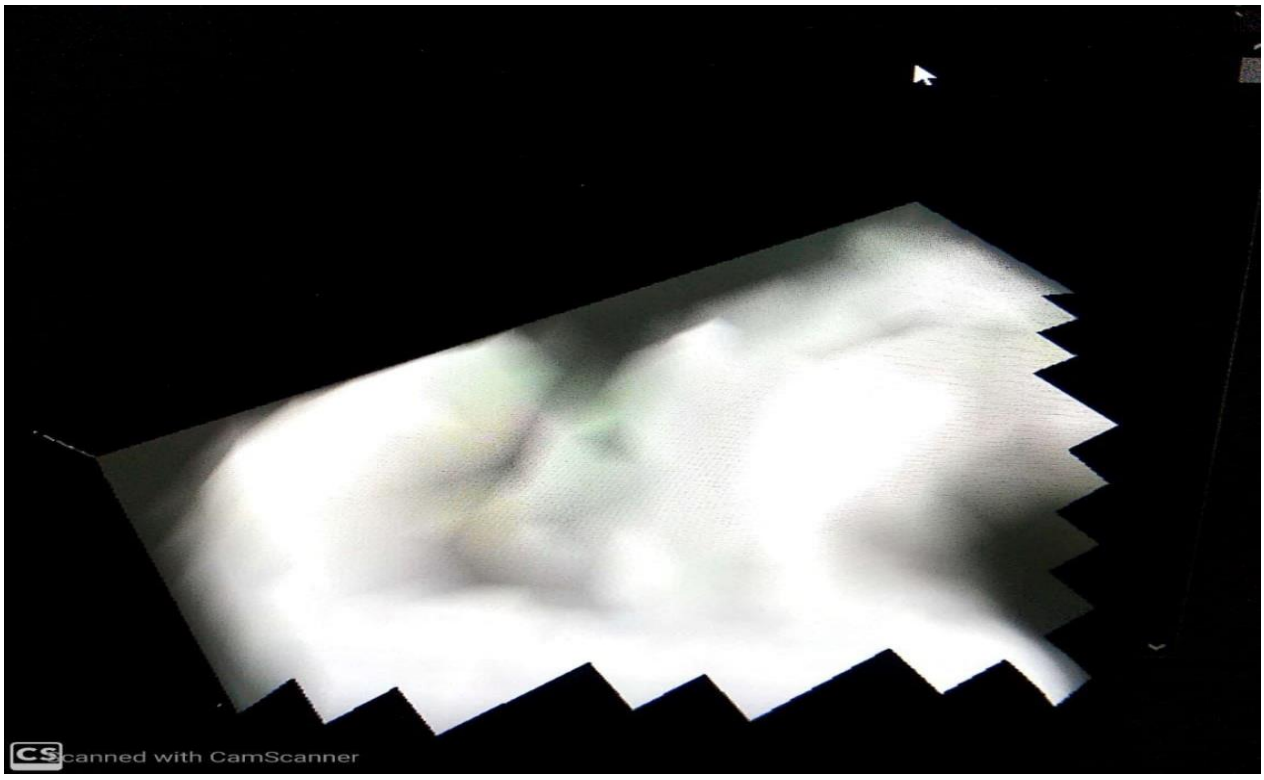
Below are few screenshots of our output, we used openGL library to fill colours in our mesh to give a different view.

- The yellow line below shows us the area covered by the viewpoint in the current frame. We can stop the rotation, manipulate it as required.

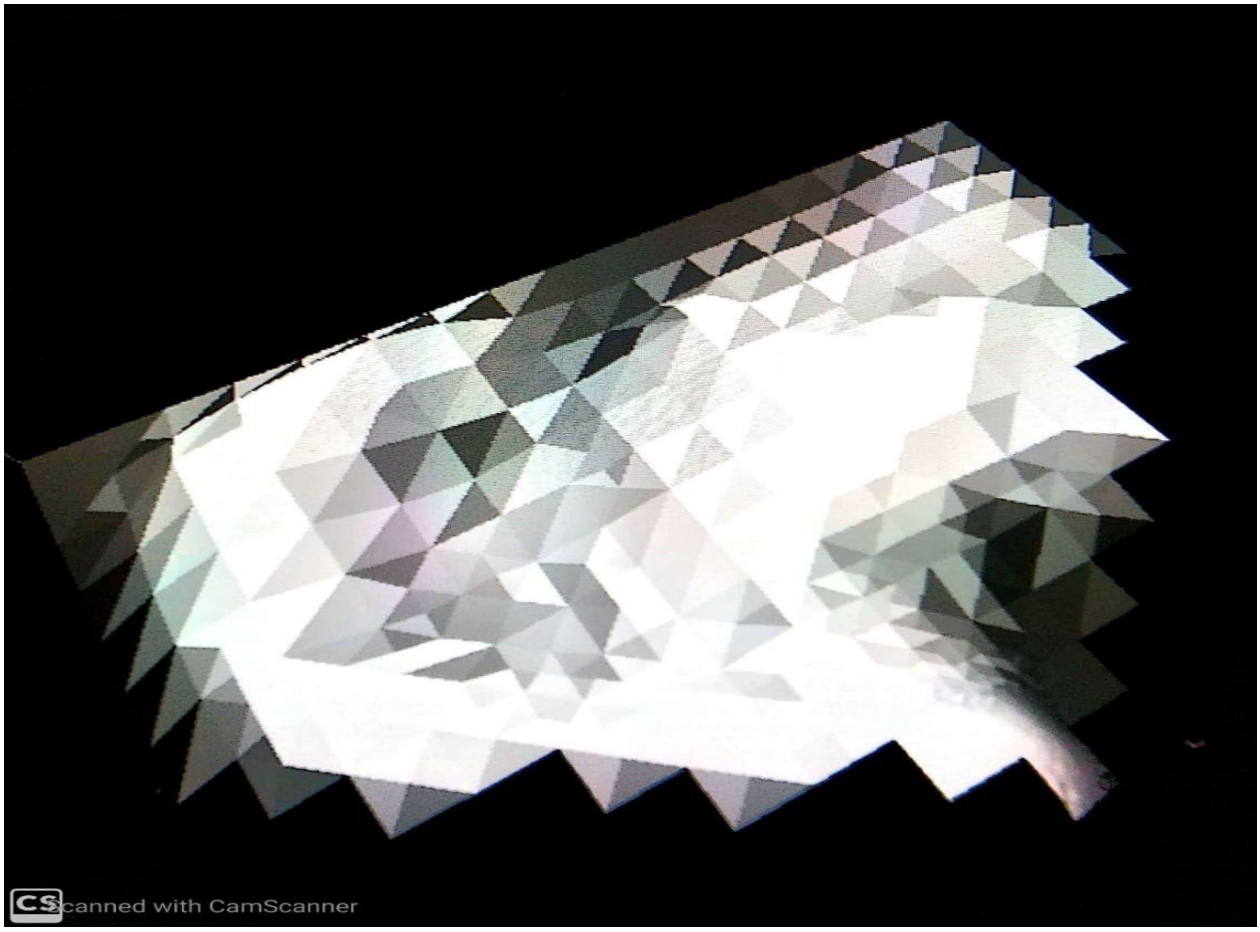




- On filling the mesh with a colour to give an idea about how bumpy or undulating the terrain is which was not clearly visible in the above representation.



- Here we filled the triangles created to give a neat representation of the triangles rendered.



CONCLUSION & FURTHER MODIFICATIONS:

So we performed terrain rendering using the ROAM algorithm**.** We can further do memory optimizations in our method especially under the rendering part which is performed using openGL. Moreover we can also derive any particular algorithms to pass patches of heightmap through landscape class which can save memory requirements.
Besides, if our viewpoint is static then we actually don’t need to recompute the variance tree again and again since the underlying heightmap doesn't change, thus saving time and memory.

Since we are using recursion we don't have to keep the previous record of the nodes of the tree thus saving a huge amount of RAM. Overall the method works fine with normal raw input files and many more optimizations can be done.

APPLICATIONS:

1. **GIS applications:** A geographic information system (GIS) is a conceptualized framework that provides the ability to capture and analyze spatial and geographic data.
2. **Computer Games:** Terrain rendering is widely used in computer games to represent both Earth's surface and imaginary worlds. Some games also have terrain deformation (or deformable terrain).

REFERENCES:

- https://en.wikipedia.org/wiki/Level_of_detail.
- https://en.wikipedia.org/wiki/Tessellation.
- A review on level of detail by Tan Kim Heok, Abdullah Bade, Daut Daman: https://pdfs.semanticscholar.org/0834/b51759461d503380e6229f487d233cad8fc3.pdf
- https://en.wikipedia.org/wiki/Delaunay_triangulation
- An implementation of Watson's algorithm for computing 2-dimensional Delaunay triangulations by:
  S. W. SLOAN
  Department of Civil Engineering, University of Newcastle, New South Wales 2308, Austraha

G. T. HOULSBY
Department of Engineering Science, Parks Road, University of Oxford, Oxford OX1 3PJ, UK

- https://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm#:~:text=In%20computational%20geometry%2C%20the%20Bowyer,graph%20of%20the%20Delaunay%20triangulation.

- https://www.gamasutra.com/view/feature/131596/realtime_dynamic_level_of_detail_.php?page=3

- https://www.gamedev.net/tutorials/programming/graphics/binary-triangle-trees-and-terrain-tessellation-r806/
