Managing and Rendering Large Terrains

Terrain is widely used in many 3D application, especially outdoor graphics environment, such as 3D games, virtual reality application, and simulation (e.g. flight simulator). In such scenes, the terrain may contain millions of triangles. Although the current graphic card has the ability to render millions of triangles per frame, many application keep on pushing the limits for the graphic card, as the demand for the better visual quality of the application increases. As a result, the gap between the graphic card performance and the necessity to display better visual quality is likely to become wider in the future.

To construct and render large terrain in the real time, I decide to use GeoMipMap algorithm since it gives best result when used together with 3d hardware rendering. The characteristics of this algorithm:

- De-emphasizing global optimization of the LOD procedure.

- Maximizing the use of GPU.

- Using triangle patches of clusters as basic elements.

The project is divided into several parts:

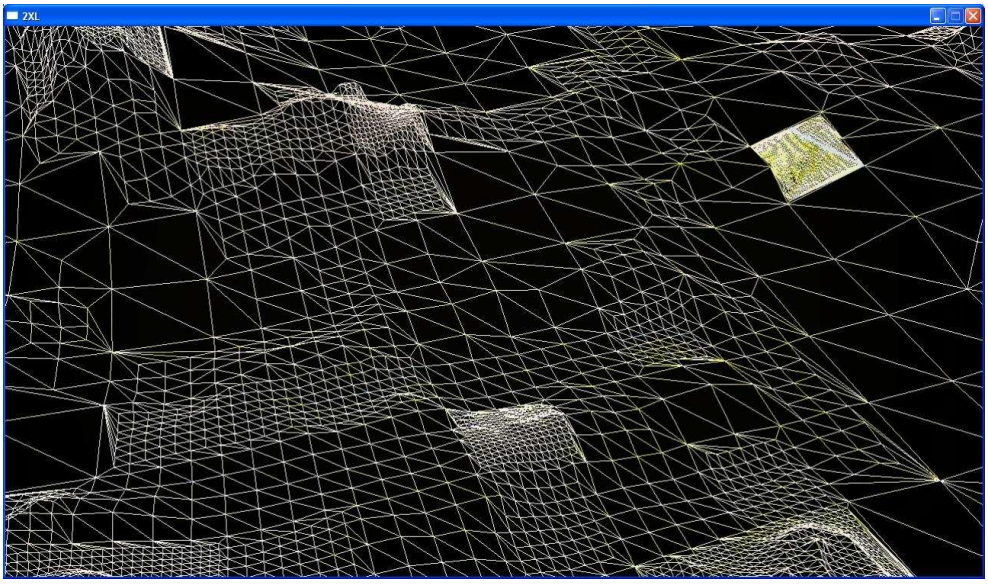
1. Scene Graph

In the current implementation, we may have > 10k terrain patch to be rendered. We used scene graph to speed up the rendering process. Scene graph is used in the frustum culling process to remove hidden (unseen) patch.

2. Terrain Rendering

GeoMipMap algorithm is used to render terrain in Real Time. GeoMipMap can be defined as texture mipmap technique in geometry which uses tiles of different resolution to match the projection terrain. The essence of the algorithm is to utilize the graphics hardware by delivering as many triangles as possible, and to use triangle strip (or fans) to speed up rendering.

Each tile in the terrain of GeoMipMap has different LOD.



Terrain rendering using GeoMipMap (viewed in Wireframe)

The computation in CPU is minimized since we only need to calculate the distance from the camera and center of tiles / patches. This distance is used to choose LOD level for each tile.

The pseudocode is described below:

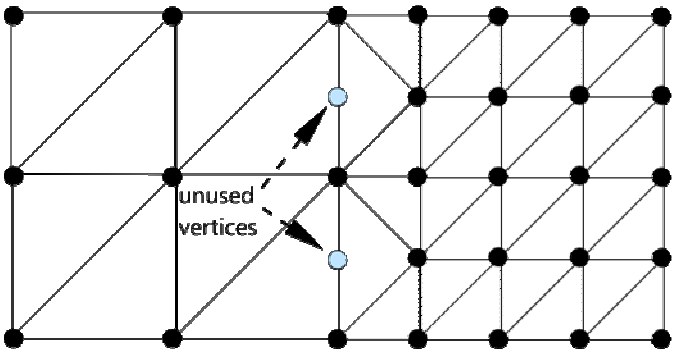
```
For i = 0 to (n – 1)
     If Dn > viewer distance
          return i
End For
Return n-1
```

Dn = minimum distance at which level of details n can be used

3. T-junction and popping effect

LOD terrain block introduce two new problems: T-junction and 'popping effect'. T-junctions can occur at the shared borders if the adjacent blocks have different

level of details. A crack may appear out of this T-junctions because higher level block uses fewer vertices on the edge. A solution for this is described below:

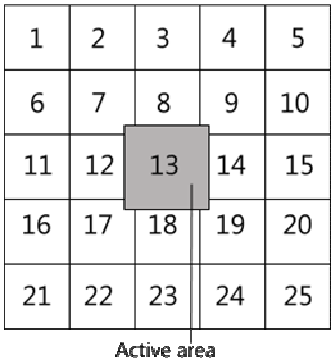


Terrain stitching

'Popping effect' happens during the switching of level of details. It can be solved by implementing Geomorphing using Trilinear Filtering. It interpolates the elevation data value of the vertices to be removed at the next level. However, this process adds workload to the CPU. To reduce the CPU workload, Wagner proposed the implementation of Geomorphing using vertex shader.

4. Streaming Height Map data

If the terrain is huge, we can not apply the GeoMipMap algorithm directly, since the height map data can not fit into memory. So, one solution is to render the terrain on demand. We load the terrain patch if it is within certain radius to the camera. And we unload the terrain patch if it is outside the boundary.



Block areas in terrain streaming

5. Achieving Low Memory

Based on observation in the real application, the number of terrain triangle rendered is only about 2% of the total triangle. To save the memory space in CPU and GPU, there is no vertices information stored. Instead, heightmap is stored in CPU memory. The engine generates vertices only when it is to be rendered. As result, small number of vertices is continuously sent instead of one big chunk once. Typical value is about 1000 vertices for each data transfer. There are two advantages of this method: higher frame rate and low memory space.

6. Terrain Collision

Terrain collision is quite easy to implement in this algorithm. The task is to find the height value, given coordinate x and y. The first step is to find the 'correct' terrain block and patch. Once found, the algorithm needs to find the 'correct' triangle for this coordinate. As the final step, Barycentric coordinate is used to find the height value.