Multi-resolution modelling of terrains by using non restricted quadtree triangulation.

A.Aguilera[*], *J.C.Torres*[+], *F.Feito*[*]

[*] Dpt. de Informática. Universidad de Jaén.
angel, ffeito@ujaen.es
[+] Dpt. Lenguajes y Sistemas Informáticos. Univ. de Granada.
jctorres@ugr.es.

Abstract

The interactive visualization of terrains requires the processing of a high amount of data in real time, usually it is not possible to work with all of them on main in memory.
In this work we present, a method which solves the above problem by means of using spatial indices (quadtree). For that, the terrains are divided into square cells, each of them having a quadtree, which will only be refined up to a the detail level depending on the observer position. With this, allaus to reduce the time used for terrains visualization.

1. Introduction

Frequently, the terrain is shown as a regular mesh of heights, stored in a bi-dimensional array, so, the occupied space is reduced to a height value for each cell. In order to visualize the terrain, we must draw two triangles on each cell. The size of the cell must be small enough so that, when drawing the terrain, its splitting into triangles cannot be appreciated. This strategy forces us to store the whole surface with the same detail level and to draw all triangles in the mesh whenever an image is generated, which limits the possibility of interaction. Because of that, several multi-resolution representations for terrains have been proposed [1, 2, 3, 4, 5], which select and draw in highest resolution only those areas of the terrain which are can be distinguired by the observer in a given moment. Furthermore, Renato Pajarola present new algorithms of the restricted quadtree triangulation [6].

In this work, the use of a spatial index on a representation in a regular mesh is proposed in order to select the resolution level to be used on each area of terrain automatically, allowing the generation of triangles of strips, with the aim of accelerating the representation. The work is based on three basic ideas:
- The use of a 2D variant of Bono indices ("Branch on need octree"), introduced to accelerate the representation of volumetric models [2], so as to index the terrain.
- The storage of quadtrees in linear arrays in order to reduce the space.
- To develop the fact that the quadtree is always complete as to calculate topological relations among adjacent cells, which are not sisters in the tree.

2. Multi-resolution model for the representation of terrains.

Our model is based on the use of spatial indices for the multi-resolution representation of terrain. In order to be able to represent the terrain with spatial indices, firstly we have to divide it into cells. The representation with a square cell has a big limitation, as it only lets us represent square-shaped terrains. To solve this problem, we will use different smaller cells instead of using only a big one.

With this multiple representation of cells, we may represent not only terrains sized $2^n * 2^n$, but also with terrains sized $2^n * i * 2^n * j$, where n means the maximum number of subdivisions a cell can reach, and i and j are the number of cells throughout the length and breadth of the terrain respectively. With this last representation, the fitting to real terrains is better than the previous one, as the part of terrain to be rejected would be $2^n - 1$ points at worst, either for X or Y coordinate.

2.1. Spatial indices

Once the division model of cells has been selected, we have to study how it will be represented so that it takes all the necessary information. It can be carried out in different ways, the most appropriate one being the quadtree representation [2], so that each node of the tree corresponds to a cell of terrain. As we have seen in the above section, the terrain will be formed by i x j cells and a quadtree for each of them. Graphically, the model would be:(See figure 1, 2, 3):

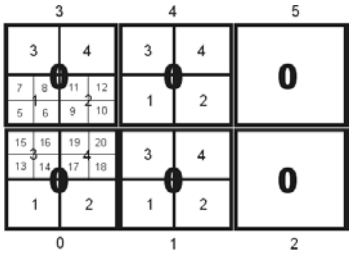

Figure 1: *Terrain obtained after applying division criteria.*

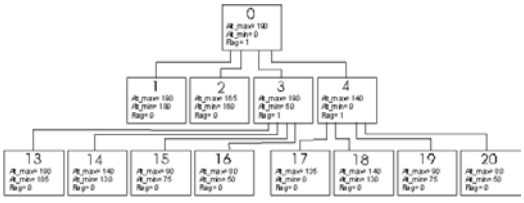

Figure 2: *quadtree corresponding to cell number 0.*

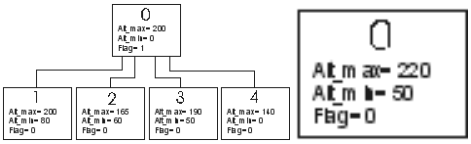

Figure 3: *quadtrees corresponding to* 1^{st} *and* 5^{th} *cells.*

Each of these nodes of the tree represents one of the cells or subcells of terrain, storing in it information of coordinates it occupies, a flag, the number of brother node if it exists (-1 if the node is on the border of terrain), and the highest and lowest height of all points the cell includes. Due to the facility to represent each of these quadtrees, we have used a vector.In order to get it, we have stored the tree in a vector, so that the position of a node is calculated from its position in the tree. So, with a given node, we would calculate where the son and father nodes are by means of the following formulae:

1) $Son[i] = father * 4 + i + 1, i \in [0,3]$,
2) $Father = (Son - 1)/4$.

So as to calculate the position of adjacent nodes, we must distinguish between two cases. Firstly, if the node is on the border of the cell, its brother node will belong to the other cell; secondly, if the node is inside the cell, the direction of node in binary will be taken; bits which represent X and Y coordinates will be separated and an arithmetical operation with the corresponding coordinate will be carried out. The figure 4 shows this process.

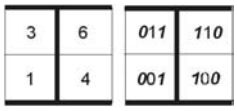

Figure 4: *Obtaining adjacent nodes.*

When calculating the brother node belonging to other cell, we must take into account that this saves a constant number of nodes, so the following formulae can be applied:
-Right node=Node-$N^o of 1^{st}$ node from present level.
-Left node=Node+$N^o of 1^{st}$ node from present level.
-Upwards node=Node+($N^o of 1^{st}$ node from present level*2).
-Downwards node=Node-($N^o of 1^{st}$ node from present level* 2).

2.2. Definition of structures used

2.2.1. Heights array of terrain points

As heights of terrain points have been obtained following a uniform distribution, we can use a bi-dimensional array in order to save such information, so that this array would have a $2^n * i * 2^n * j$ size.

This array will be used to get the height of each of the points we will draw on the screen, in such a way that from the coordinates of a point in the space, we will obtain the height of it.

2.2.2. Cells array

In order to be able to represent the model described before, we need a quadtree pointers array. The size of this array will be $i * j$, being "i" and "j" the values calculated previously for a terrain sized $2^n * i * 2^n * j$ puntos.

In addition to the quadtree pointer, each cell in this array will store the coordinate it takes up in the terrain, the maximum and minimum height of all points composing that cell, a flag to let us know if the cell is divided and the number of right-hand sister cell. If this cell does not exist for being on the border of terrain, we will give it the value -2.

To represent this array a vector has been used, numbering each of the nodes in an orderly way, so that having the cell number, we can know which row and column it occupies in our terrain.

2.2.3. The quadtrees

As we have explained above, in order to represent a terrain, we will use a spatial indices model, particularly quadtrees. The quadtree root node represents a square cell which is not divided, in such a way that as we go down the tree, the nodes would represent a part of the cell, the cell shape and the sub-cells.

In each node of the tree we will gather information on X and Y coordinates which that cell occupies, the right-hand sister cell and the maximum and minimum heights of all points composing that cell, and a flag showing whether that node is divided or not.

2.3. Definition of criterion for cells division

Both in the array of cells and in each node of quadtree, we store a flag. The flag is used for knowing how our terrain is drawn every at moment, since with it we can obtain the final nodes of either all quadtrees or array of cells.

In order to mark the final nodes of all quadtrees, that is, to carry out a pruning so as to eliminate the non-relevant parts for the drawing on screen, we have defined an independent function which goes over each cell of the initial array and sees whether they comply with the division criterion. If it does, it enables a flag, and then we observe the four son nodes so as to see whether they comply with the division criterion. This process is done again repeatedly up to the node does not comply with the criterion or we come to a leaf node, marking these nodes with a 0 value in flag.

The criterion used for marking the nodes is based on proximity and terrain roughness, that is, so as to know whether a cell must be divided or not, we will calculate the distance of that cell from the observer (who is placed in a point inside or outside the terrain), and the cell roughness (noting the difference of heights between its highest and lowest points) These two values are compared with a threshold so that, if it excels its value, then the cell is divided, not happening this when it does not.

2.4. Drawing the terrain

In order to draw the terrain in an efficient way, we obtain the nodes strips composing it. For that, we make use of the marked run of nodes to see whether every last node is a starting node of a strip or not. If the node is the initial one of a strip, then we store its number in a list. Later, this list will be used for drawing the terrain, taking each of its nodes and going over the quadtree so as to see whether the brother node on the right has the same level of subdivision. The drawing of the strip will go on untilit arrives to a node which does not have the same level as the initial one.

The aim of using strips for drawing the terrain is the good use of the optimization that OpenGL has for drawing them.

Drawing the terrain with different strips sized, the discontinuity or gaps which appear when having two strips together formed by nodes with different level inside the quadtree, taking form the so-called Crack (See figure 5).
One way of avoiding Cracks is using the fan-shaped drawing. In order to draw a terrain according to our model, we will use strips and fans, so that, when all nodes surrounding the node to be drawn have higher or same resolution level, then, we will draw that node as a strip, and if not, as a fan. In this way we avoid discontinuities among different strips. The algorithm used for marking and obtaining the last nodes, which are the initial nodes of a strip, is(See fig. 6 and 7).

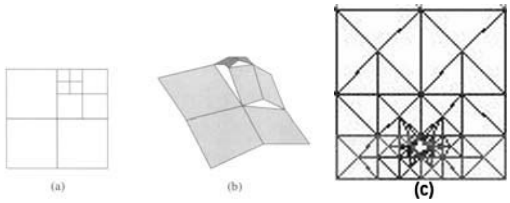

Figure 5: *A and b terrain drawn with crack. C terrain drawn without crack.*

```
Procedure obtain_terrain(array_cells terrain)
  Begin
    For num_cell=0 to num. of cells
      If(divide_cell(terrain, num_cell)== True )
        then
          terrain[num_cell].flag=1
          obtain_tree(terrain[num_cell].tree, 0)
        else
          terrain[num_cell].flag=0
          If (terrain[obt_cell_brot_left(num_cell)].flag==1)
            then
              Add num_cell to list of initval nodes of a strip
          end_if
        end_if
      end_for
    end
```

Figure 6: *Algorithm to obtain the terrain.*

2.5. Drawing the visible area of terrain

In order to optimize the drawing process, frustrum culling is made, drawing only the cells which are visible by the observer in a given moment.

For that, a new flag for each terrain of cell is added, in such a way that when the flag is enabled, that cell will be near the observer and visible for him. In this way we do not have to draw the whole terrain, if it is not visible for the observer. With this culling, the number of frames per second we can draw is increased.

```
Procedure obtain_tree(type_tree tree, int node)
  begin
    if(divide_node(tree, node)==True) AND (not_s_leaf(node))
      then
        obtain_sons(node, son1, son2, son3, son4)
        obtain_tree(tree, son1)
        obtain_tree(tree, son2)
        obtain_tree(tree, son3)
        obtain_tree(tree, son4)
        tree[node].flag=1
      else
        tree[nodo].flag=0
        if (tree[obt_node_brot_left(node)].flag==1)
          then
            Add num_cell a list of initval nodes of a strip
        end_if
    end_if
  end
```

Figure 7: *Algorithm to obtain a cell.*

The algorithm for the representation of terrain only go over the cells visible by the observer and generating the number of cells implied in the visualization. For that, two variables which will influence on the number of cells generated for visualization, must be taken into account: on the one hand, the direction on which the observer moves, and on the other one, the terrain depth to be drawn. As the terrain is stored in a cells vector, we to apply the following formulae in order to obtain the numbers of cells implied:

- The observer is moving upwards:
  n=((x-j)+num_cells_x*(y+i)+i)
- The observer is moving to the left:
  n=(x+num_cells_x*(y+i-j)-i-1)
- The observer is moving downwards:
  n=((x-j)+num_cells_x*(y-i-1)+i)
- The observer is moving to the right:
  n=(x+num_cells_x*(y+i-j)+i)

where X and Y are the coordinates where the observer is, "num_cells_x" is the number of cells in a row, and "j" and "i" are the indices of the following nested loops:

```
for (i=0;i<DEPTH;i++)
    for(j=0;j<(i*2+2);j++)
```

In the figures 8 and 9 we can see how the culling would affect the terrain drawing, placing the camera from the observer point of view and from upwards.

3. Conclusion and future works

A method for the representation of terrains has been developed by using quadtrees, which have let us increase the number of frames per second.

As future works, will study the criterion of cells division in order to see which of them is more suitable for obtaining a better representation of terrain.

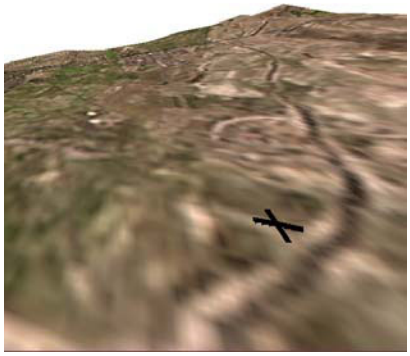

Figure 8: *Last terrain with culling.*

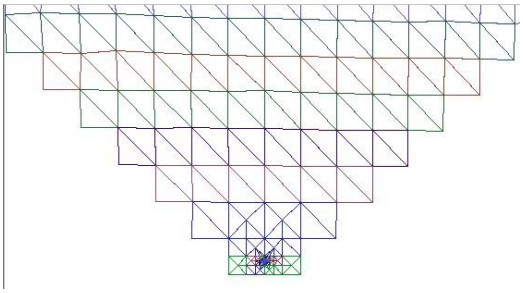

Figure 9: *Wireframe model with culling.*

References

1. V.Scheib, J.Haber, M.C.Lin, H.Seidel “Efficient Fitting and Rendering of Large Scattered Data Sets Using Subdivision Surfaces.”, *EG’02 Volume 21, Number 3*. 1
2. J. Wilhelms, A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, **11**(3):201–227, July 1992. 1, 2
3. P.Cignoni, E.Puppo, R.Scopigno, 1997. Representation and Visualization of Terrain Surfaces at Variable Resolution. *The Visual Computer*, **(13)**, 5, July 1997. 1
4. L. De Floriani, P. Marzano, E. Puppo. Multiresolution Models for Topographic Surface Description. *The Visual Computer*, **(12)**, 7:317–345. 1
5. E. Puppo. Variable Resolution Terrain Surfaces. *Genova University, tech.rep DISI.TR96-6* 1
6. Renato Pajarola. Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation. Vis 98. 1