Optimizing tree distribution in virtual scenarios from vector data

Tiago Augusto Engel* Alex Frasson Cesar Tadeu Pozzer

Universidade Federal de Santa Maria, Brazil

Figure 1: A snapshot of the tree distribution generated by our system.

ABSTRACT

Tree distribution in virtual landscapes is a key feature on several application fields, where automated tools face challenging problems regarding performance and realism. It is common for this task to rely on designers to manually place elements, but when it comes to large scenarios, it becomes impracticable. Games and simulation systems are often based on real world scenarios generated using data from Geographic Information Systems (GIS). This paper describes an optimized algorithm to populate forest represented by polygon map in a virtual scenario. The solution is based on a quadtree built from the GIS data, providing fast location queries and enabling real time tree scattering.

Keywords: tree distribution, virtual scenarios, GIS, quadtree.

1 INTRODUCTION

Forests are fundamental part of natural landscapes. Methods to capture their features have been subject of study in several areas. In this context, Geographic Information Systems (GISs) have become a key tool, enabling the capture, storage and processing of large geographic databases. GISs provide important means for understanding and dealing with nature events, thus allowing us to understand, reason and predict environmental phenomenons such as flooding and forest fires, or as a support tool for decision-making and public administration.

The creation of virtual scenarios using GIS data has applications ranging from public administration, tourism, games, science and technology, to military simulation systems [12]. Various efforts have been placed in order to automate the process and provide realistic aspects on the virtual world. However, these aspects still pose serious challenges nowadays, as to do so usually requires processing massive amounts of data [5].

Recreating forests in virtual environments still poses several challenges regarding modeling, distribution and rendering. Simulation scenarios require special attention to realism, where natural components such as rivers, vegetation and terrain play crucial role on recreating an actual feel of presence. For instance, vegetation areas may present camouflage and sources of food. Realism is also important regarding geographically accurate representation, where real world features are properly placed.

Our primary goal is to allow real-time tree distribution and management. The tree management can be thought of as a polygon fill task, where areas are populated with trees, while the actual tree positions are defined by a distribution task. In this sense, defining realistic tree positions relies on forest dynamics, which are complex to simulate [1].

This paper describes an ongoing work towards providing realistic tree distribution and management in a virtual scenario. The input consists on a real world dataset containing the terrain heightmap and a polygon map delineating the forest areas. We propose a polygon fill algorithm based on a quadtree to distribute trees along the area. Finally, we show how to associate other features such as roads and rivers, preventing inconsistent cases. The paper is organized as follows: section 2 provides background and related works on the subject, in Sections 3 and 4 we propose an approach to the problem and describe its implementation, respectively. Finally, in section 5 we evaluate and discuss current results.

2 RELATED WORKS

Forests have been subject of many works in Computer Graphics, while tree distribution is a subject of forestry and environmental sciences. Our primary concern is not rendering nor level of detail for individual plants (see [2] for a survey on this subject). We also assume the trees are already modeled and our main focus is ground level visualization.

Polygon fill approaches. Populating a polygon with trees can be thought of as a rasterization task, where algorithms such as scanline and flood fill are used, with the difference that the samples are not equally spaced. However, rasterization approaches test each sample to determine whether it belongs to the primitive (usually triangles), and while this approach works fine on primitive forms, it doesn't scale well for large polygons.

Point in Polygon algorithms. Determining whether a point is inside a polygon is a classic computational geometry problem. When it comes to vegetation areas, there are degenerated cases that must be taken into account. The polygons may contain holes (represent openings) and have any shape (concave or convex). Huang[7] presents a full study on several point in polygon algorithms, from where we chose to implement the Crossing Number (also known as ray intersection) algorithm.

*e-mail: tengel@inf.ufsm.br

The crossing number algorithm consists on tracing a ray from the test point along an axis (x or y) and summing the number of times the ray crosses a polygon boundary. If the number of crossings is odd, then the point is inside the polygon, otherwise the point is outside. The Figure 2 illustrates of the two cases. We chose this approach for its simplicity, compliance with the aforementioned constraints, the needless of pre-processing and the $O(n)$ time complexity.

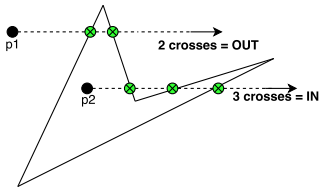

Figure 2: Crossing number algorithm.

Polygon Map (PM) Quadtree. Quadtree is a hierarchical spatial tree data structure that consists on the recursive decomposition of space [9]. It has been used in the literature to describe vector data[11, 10, 3] due its properties that are specially suitable for GIS data. Particularly, we are concerned on supporting polygon maps, as this is the common format for forest GIS data. Samet [11] presents an extensive study on PM quadtrees, where the authors discuss a full background on the matter and propose implementations for point location, dynamic line insertion and map overlay.

Tree distribution in nature. The automated generation of realistic tree distributions is a complex task [1]. To synthesize such level of dynamism, there is a number of factors to be taken into account such as: biome, species, soil features, weather conditions and age. Current studies are restricted to specific biomes or even locations, limiting world scale applicability. Botkin [1] presents a computer model to simulate forest growth in a mixed-species in north america. Crowther [6] provides a global scale study on tree density, showing biome specific statistics. Liu [13] presents a space occupancy-based method using an octree to simulate tree groups, enforcing relations such as: deviation, intersection, covering and shelter. Lane [8] provides a multilevel model focusing on plant distribution using multiset L-systems to simulate self-thinning, succession and clustering.

3 METHODOLOGY

The polygon fill approaches mentioned on Section 2 present a crucial issue: they need to test every sample to assure they belong to the polygon. This means that the performance is bound to the number of samples and the polygon vertex count. We can consider the polygon a static entity (no dynamic changes on shape), on the other hand, the sample density should be flexible. For instance, we can progressively increment the tree density according to the distance from the camera. However, when having high density samples, performance issues start to arise.

We aim to reduce the number of point in polygon operations, thus diminishing the performance impact caused by higher densities. In order to achieve that, we propose a PM quadtree structure to store the polygon. Once discretized, we can access the quadtree nodes and distribute the trees without needing point in polygon tests.

We propose the workflow presented on Figure 3. The solution consists on 1) loading the polygon map from the appropriate GIS vector format; 2) build the PM quadtree; 3) generate the forests by traversing the quadtree; 4) verify intersection with other features i.e. trees within rivers, roads; and 5) use the generated distribution to place the tree models in the 3D world.

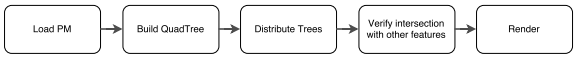

Figure 3: Solution workflow.

The forest polygon map may intersect with other vector features such as rivers and roads. The algorithm must foresee such cases in order to prevent inconsistent cases (e.g tree over a road or within a large river). These cases require special attention and can be solved either within the tree generator module or as a post-processing step once the tree positions are already generated. The following sections describe with detail the implementation.

4 IMPLEMENTATION

4.1 PM Quadtree

Our implementation is slightly different than the ones proposed by [11]. While their approach is based on a point space quadtree, focusing on the vertices, we are concerned with the edges as nodes shape should resemble the original polygon. Once proper level of detail is reached, we can use the quadtree to distribute trees instead of using the original polygon. In order to build the quadtree, there are three basic cases that need to be handled:

1. **Node encloses the polygon.** The node size is larger or equal the polygon bounding box, thus the polygon is entirely enclosed by the node.

2. **Polygon does not intersect the node.** The node is completely inside (`FULLYIN`) or outside (`FULLYOUT`);

3. **Node and polygon intersect.** The node has one or more edges intersecting the polygon.

The first case is the start setting, where the node size is the same as the polygon bounding box. This case is handled by subdividing the node in four. The second case is when the node is completely inside the polygon – in this case the algorithm that the node does not need to be split anymore, and the node can be tagged as `FULLYIN` or `FULLYOUT`. Finally, the last case is when there are intersections between the node edges and the polygon, handled by subdividing the node and recursively calling the build function in the new children. Figure 4 shows the three cases.

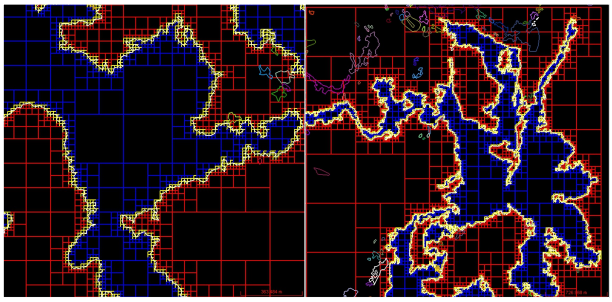

Figure 4: Quadtree generated from a large polygon. The red line segments represent `FULLYOUT` nodes, while the yellow and blue shows `PARTIAL` and `FULLYIN` ones, respectively.

The stop conditions are two, namely: 1) when a small node size is achieved (5x5 meters, for instance); and 2) the quadtree reached a depth limit. In these cases, if the node still has intersections, it is classified as partial (`PARTIAL`), which are those who have at least one edge intersecting the polygon. Otherwise, the nodes that intersect the polygon are further refined and classified.

The Algorithm 1 presents the `buildNode` algorithm. We use the crossing number algorithm to detect point in polygon while building the tree, and line to line intersection to determine if any polygon edge intersects quadtree nodes. While we still need to detect if node corners are inside the polygon, the number of tests is much smaller than we would have otherwise.

Input: node ← current quadtree node
Input: depth ← current tree depth
/* Test for stop conditions */
if
($depth > maxDepth$ ***or*** $getNodeSize(node) \leq minNodeSize$)
then
  if
  ($polyIntersectNode(node)$ ***or*** $nodeContainsPoly(node)$)
  then
    $node.location \leftarrow PARTIAL$;
  end
  else
    $node.location \leftarrow getNodeLoc(node)$;
  end
  return
end
/* Subdivide the node and recursively build child nodes */
if ($polyIntersectNode(node)$ ***or*** $nodeContainsPoly(node)$)
then
  $node.subdivide()$;
  $node.location \leftarrow PARTIAL$;
  $buildNode(node.nw, depth+1)$;
  $buildNode(node.ne, depth+1)$;
  $buildNode(node.sw, depth+1)$;
  $buildNode(node.se, depth+1)$;
end
/* Determine final location. */
$node.location \leftarrow getNodeLoc(node)$;
return;

Algorithm 1: BUILDNODE recursively builds the quadtree

4.2 Quadtree traversal and tree distribution

In this step, the tree positions are created in the virtual world. Once the quadtree is built, there is no need of point in polygon tests anymore. In fact we can simply traverse the quadtree and fill the leaf nodes that are inside the polygon with tree positions. This process can be done in real time, as long as the distribution does not offer major performance issues.

In order to test the distribution, we access each leaf and sample it regularly, adding a random offset for each point (see results in Figure 5). While this is a very simple solution, this was enough to test our approach. In future developments we will be focusing in the distribution itself. The algorithm to distribute trees is described in Algorithm 2. It consists on traversing the quadtree and filling the nodes that are fully inside the polygon. Depending on the accuracy needed, nodes partially intersecting can be filled as well, however point in polygon tests would be necessary in these cases. The method `distributeTrees` can be implemented using several approaches. For instance, Bruneton [4] presented an aperiodic tiling based on point distribution patterns.

The tree positions generated must not intersect with other features such as roads and rivers, therefore we need to remove trees in these locations. Such features are normally represented by polylines (sets of lines adjacent to each other). The algorithm to remove trees uses the generated positions (can also be done at generation time) and computes distance-to-line segment operations in order to identify close trees.

Input: node ← current quadtree node
Input: sampleSpacing ← distance from each sample
/* Stop condition */
if ($node == null$) **then**
  return
end
if ($node.location == FULLYIN$) **then**
  $distributeTrees(node.minBound, node.maxBound, sampleSpacing)$;
end
/* Call recursively to all children. */
$genTrees(node.nw)$;
$genTrees(node.ne)$;
$genTrees(node.sw)$;
$genTrees(node.se)$;
return;

Algorithm 2: GENTREES recursively distribute trees in the quadtree nodes

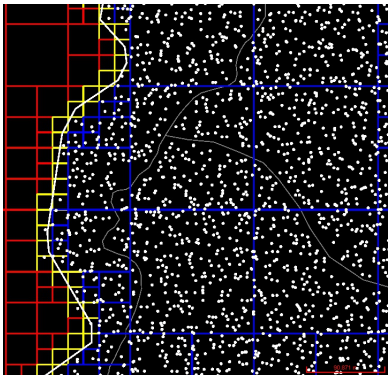

Figure 5: Tree distribution result for `minNodeSize=20` and `maxDepth=11`. White dots represent tree positions. Trees closer than 5m from the road (grey lines) were removed.

5 PRELIMINARY RESULTS AND DISCUSSION

Our quadtree nodes must be refined enough at the polygon edges in order to assure smooth forest borders. This separates our approach from the ones such as Samet's [11], and while our quadtree approach demands a higher node count to account for edge detail, it still offers good result in terms of performance.

We evaluated our solution using a medium size map with 3856 polygons covering a 25x55km area. Table 1 shows the details. The real world location is not mentioned for secrecy reasons. The dataset presents large polygons with, on average, 38.54 vertices per entity (polygon), being a good benchmark test for our algorithm.

Table 1: Terrain Settings.

Name	Entities	Vertices	Width(m)	Height(m)
Alpha	3856	148626	25860	55600

We evaluated the performance using an Intel i7 870 CPU with 4GB RAM. The running times were evaluated and the average of three runs was sampled. Results show that our approach is suitable for high density fill, while the point in polygon algorithm behaves better on sparse data. This is due the fact that the latter depends heavily on the sample count. On the other hand, the needless of any data structure makes the crossing number specially suitable for small polygons.

The Figure 6 shows the sampled running times. We can see that the quadtree approach maintains a stable running time – this

is caused by the quadtree construction, the most demanding processing task. On the other hand, once the tree is built, the density does not impose performance issues for the algorithm. Evaluation showed an overall of 1.126s to distribute trees across the whole terrain. It is also noticeable that the crossing number running times are bound to the sample density, as for each testing sample, a linear query is made in the polygon.

We observed a lower tree sum in the quadtree approach – fact associated with the distribution algorithm where only `FULLYIN` nodes are filled. The `PARTIAL` nodes belong to the border and were ignored. Considering that forest generation does not demand high precision placement, we can ignore the partial nodes. However, the crossing number can be used in the partial nodes in case precision is a requirement, allowing to have precise fill along the edges.

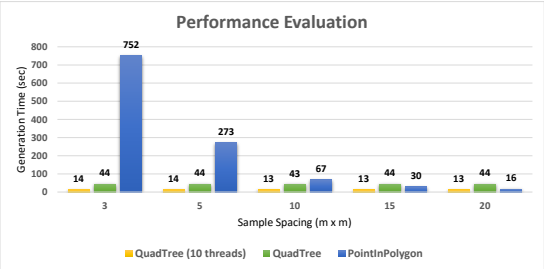

Figure 6: Performance comparison between the polygon fill algorithms when using different tree densities.

We identified that the performance is bound to the large polygons, which limited the overall running time. The 10 largest polygons have on average 3445 vertices each, while on the remaining polygons the average is 29. In the first case, the quadtree presents the most advantages as the large vertex count compensates the build time, specially when working with high densities.

Since every polygon has its own quadtree and there are no dependencies between them, we developed an alternative solution to compute the large polygons. The approach consists on a parallel version of the `buildNode` algorithm. The PM is sorted by size (in terms of vertex count) and the n largest polygons are assigned to a thread group, where each instance works on a polygon. While the main program processes the small polygons, the large ones are computed in separated threads. Figure 6 demonstrates the great speedup obtained with the mixed approach over the previous solution. Finally, the parallel removal of trees near river and roads presented great performance speedup, we split the generated trees across the threads so that there is no concurrency issues. Results show an average of 6.9 speedup when using 10 threads over the sequential version.

6 Conclusion

When building 3D scenarios, the task of distributing trees often is made manually by designers. However this approach does not scale for large terrains and world scale distributions. Furthermore, games and (specially) simulation applications frequently recreate real world scenarios, which are represented using GIS data.

We presented an optimized approach to generate forest distribution when its boundaries are described using polygon maps. The solution is based on a quadtree built from the vector data. The data structure is used to provide fast tree distribution across the polygon area. Performance analysis shows that the solution provided great speedup for high density areas, while the original point in polygon solution outperformed in low density and small polygons.

We also developed a mixed approach, where largest polygons are built in parallel. The alternative solution showed considerable speedup over the sequential approach. The results show that our approach is a viable solution to this problem, providing a solid base for the tree distribution algorithms, where further studies are encouraged to develop a more realistic solution.

7 Future Works

We presented an ongoing work that provides many work opportunities. To name a few:

- **Realistic tree distribution.** In-depth study towards realistic tree distribution in the terrain, taking into account nature patterns, as shown in Section 2.
- **Generalization for other vegetation types.** Polygon maps can describe other kinds of plants such as farm plantations, which require specific patterns for generation.

Acknowledgements

We thank the Brazilian Army for the financial support through the SIS-ASTROS project, developed in the context of the ASTROS 2020 strategic project.

References

[1] D. B. Botkin, J. F. Janak, and J. R. Wallis. Some ecological consequences of a computer model of forest growth. *Journal of Ecology*, 60(3):849–972, 1972.

[2] F. Boudon, A. Meyer, and C. Godin. Survey on Computer Representations of Trees for Realistic and Efficient Rendering. Research Report 2301, 2006.

[3] E. Bruneton and F. Neyret. Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum*, 27(2):311–320, 2008.

[4] E. Bruneton and F. Neyret. Real-time realistic rendering and lighting of forests. *Computer Graphics Forum*, 31(2):373–382, 2012.

[5] P. Cozzi and K. Ring. *3D engine design for virtual globes*. CRC Press, 2011.

[6] T. W. Crowther, H. B. Glick, K. R. Covey, C. Bettigole, D. S. Maynard, S. M. Thomas, J. R. Smith, G. Hintler, M. C. Duguid, G. Amatulli, M. N. Tuanmu, W. Jetz, C. Salas, C. Stam, D. Piotto, R. Tavani, S. Green, G. Bruce, S. J. Williams, S. K. Wiser, M. O. Huber, G. M. Hengeveld, G. J. Nabuurs, E. Tikhonova, P. Borchardt, C. F. Li, L. W. Powrie, M. Fischer, A. Hemp, J. Homeier, P. Cho, A. C. Vibrans, P. M. Umunay, S. L. Piao, C. W. Rowe, M. S. Ashton, P. R. Crane, and M. A. Bradford. Mapping tree density at a global scale. *Nature*, 525(7568):201–205, 2015.

[7] C.-W. Huang and T.-Y. Shih. On the complexity of point-in-polygon algorithms. *Computers & Geosciences*, 23(1):109–118, 1997.

[8] B. Lane and P. Prusinkiewicz. Generating Spatial Distributions for Multilevel Models of Plant Communities. *Interface*, 2002:69–80, 2002.

[9] D. P. Mehta and S. Sahni. *Handbook Of Data Structures And Applications (Chapman & Hall/Crc Computer and Information Science Series.)*. Chapman & Hall/CRC, 2004.

[10] R. C. Nelson and H. Samet. A Consistent Hierarchical Representation for Vector Data. *SIGGRAPH Comput. Graph.*, 20(4):197–206, aug 1986.

[11] H. Samet and R. E. Webber. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics*, 4(3):182–222, 1985.

[12] X. Wang. Integrating GIS, simulation models, and visualization in traffic impact analysis. *Computers, Environment and Urban Systems*, 29(4):471–496, 2005.

[13] Z. Wang and F. Liu. Three-dimensional Modeling of Trees Group with Spatial Characteristics and Its Space Subdivision Method. (2007):2–5, 2010.