Shape2River: a tool to generate river networks from vector data

Tiago Augusto Engel* Cesar Tadeu Pozzer

Universidade Federal de Santa Maria, Brazil

Figure 1: River network generated by our system.

ABSTRACT

This paper presents a tool to recreate hydrographic networks for 3D virtual scenarios using Geographic Information System (GIS) data. A vector description of the hydrography and a raster description of the terrain are used as input. The vector data is subdivided using a spatial hashing technique and the river data structure is modeled as a graph from where the water surface mesh is generated. The geometry is sent to GPU from where footprint textures and associated shaders are employed to enable terrain shape modification and texture mapping, producing various river feature elements. The solution is primarily targeted for simulation systems, but the concepts can be applied on 3D games or virtual environments in general. Results show a simple yet efficient process that provides quality visuals.

Keywords: river rendering, GIS, virtual scenarios, spatial hashing.

1 INTRODUCTION

Geographic Information Systems (GISs) have become increasingly popular nowadays due their wide application range[14]. Furthermore, the growing computing power of current computers has enabled the capture, storage and processing of large databases. GISs provide important means for understanding and dealing with nature events, thus allowing us to understand, reason and predict environmental phenomenon such as flooding and forest fires, or as a support tool for decision-making and public administration.

In this context, several initiatives emerged to provide free access for GIS data, as well as tools for processing them. However, these tools normally provide only a 2D view of the scenario or are very expensive. This has motivated a growing development area focusing on the generation of three dimension scenarios. Various efforts have been placed in order to automate the process and provide realistic aspects on the virtual world. However, it still imposes serious challenges nowadays, as it usually requires processing massive amounts of data [4].

Simulation scenarios require special attention to realism, where natural components such as rivers, vegetation and terrain play a crucial role on recreating an actual feel of presence. For instance, a river may represent obstacles that must be avoided or can be trespassed depending on the depth. Realism is also important when considering geographically accurate data to locate and reference entities in the virtual world, as if it was in the real one.

This work describes an automated process to recreate hydrographic networks using GIS data. By matching the vector data containing the river network information and a digital elevation model of the terrain, we aim to recreate a real world scenario. The solution is primarily targeted for simulation systems, but the concepts can be applied on 3D games and virtual environments in general.

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

2 RELATED WORKS

As our purpose is to add features into the terrain, we need to find ways of matching vector and elevation data together. Level of detail (LOD) has to be considered in order to produce the features, as using different LODs implies that the constituent triangles' positions may change at any frame, requiring the features to adapt accordingly. There are generally three approaches to render such features on the terrain [3]:

- **Overlay-geometry based:** this approach renders geometry on top of the terrain [15, 1, 11]. The method is susceptible to problems on LOD switching, as the objects have to be kept above the terrain. Furthermore, two coplanar surfaces are a common source of z-buffer artefacts (Z-fighting) [1].

- **Geometry based:** this approach consists on embedding the vector data into the terrain mesh itself [5]. This is achievable by evaluating the mesh locally and adding new triangles in order to represent the desired feature.

- **Texture based:** consists on rendering the vector features into textures, which are mapped onto the terrain [7, 12, 3]. Texture-based techniques have drawbacks related to long preprocessing time and the resolution cannot be changed without remapping the vector data [7]. Furthermore, the texture resolution is directly related to the occurrence of aliasing artefacts.

Our rendering approach is based on Bruneton's [3], which presents a texture based approach, where a data structure combines terrain elevation, appearance and vector data of different features (e.g. rivers and roads). The quadtree data structure holds the clipped vector data for each quad, and rasterize it when a new quad is visible. The output texture (called footprint) is sent to the

GPU where, among other operations, the texture is blended with the height map, thus creating a footprint effect.

In order to subdivide the vector data and speedup graph building time, we implemented a spatial hashing. The technique is an efficient approach to speed up proximity queries, commonly found on simulations composed by a large number of entities [9]. The hash function receives a 2D point and, based on the number of cells (or buckets) wanted, calculates the cell it belongs to. We use this approach as graph edges are determined by proximity, thus we can substantially reduce the time to access neighbours. We adapted the solution proposed by Pozzer [10], which allows us to build he hash table on $\mathcal{O}(n)$ complexity.

3 METHODOLOGY

We propose the following workflow for the river generation process (according to Figure 2): 1) load the vector data using an appropriate library, 2) apply a subdivision structure for the dataset in order to speed up proximity queries, 3) build the river network graph using the subdivided data, 4) generate the river surface mesh using the graph, and 5) render the geometry using the latest resources on the OpenGL pipeline.

Figure 2: Development workflow.

Our goal is to render two types of river features: the river bed and surface. The First represents the channel bottom which physically confines the water flow, while the latter is the actual surface water layer. Figure 3 illustrates the goal features.

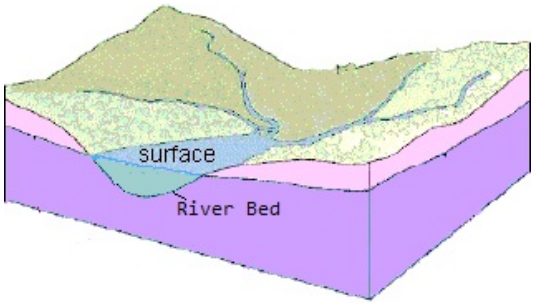

Figure 3: Goal summary. Figure adapted from [13]

In order to create the river bed, we use multi-resolution footprint textures, which are dynamically generated based on the camera position by rasterizing the river mesh at a proper resolution. They are generated on the fly directly on GPU, thus eliminating pre-processing steps. More details will be discussed on the next session. This approach is based on Bruneton's [3] and can also be applied for road generation.

The geometry is composed by the river surface mesh and the terrain. GPU also receives three textures: the elevation texture which is used to make the terrain height displacement, footprint texture to generate the riverbed feature and the appearance texture that represents the terrain surface color. Figure 4 shows the overall organization of our system.

Using this data, each stage of the rendering pipeline is employed in order to render the scene. The following sections discuss in detail the workflow implementation.

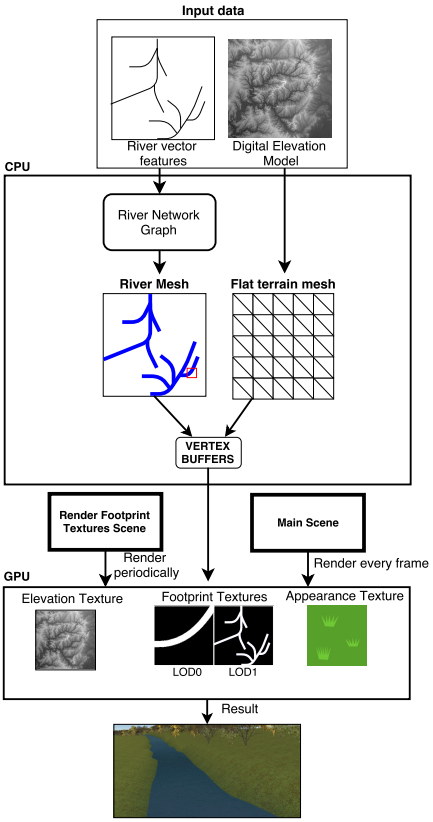

Figure 4: Overall System Organization. The workload is shared between CPU and GPU. We are working with two LOD levels for the footprint texture: one for close distance (LOD0) and another for far distance in relation to the camera (LOD1). Each stage has its inputs and outputs according to its role.

4 IMPLEMENTATION

This section describes the data structure adopted to represent the river network, and the methods used to render it.

4.1 River data structure

A river network can be represented as a set of polylines (also known as entities), which represent a set of connected vertices. Figure 5 shows an example of vertices and entities. We are using the ESRI ShapeFile[6] format, that is one of the most commonly used for vector data.

The river data structure must allow the representation of hierarchy and connections between different affluent and the main river course. Furthermore, it must be traversable to search within the dataset in order to find where the springs and river mouths are, as well as to understand the topography. Such features can be represented as a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where nodes are represented by polylines and edges the connections between them.

The polylines are stored in an unordered way, thus we can't assume any ordering or hierarchy between them. The only assumption we can make is that, within a polyline, the points are ordered. Another important feature is that when intersections occur, the point is replicated (one for each intersecting polyline). Therefore, in order to find adjacencies we need to loop through all points within the dataset (regardless polyline) and check whether each point intersects with other point(s) within the dataset. If so, we assign an edge from the the current polyline to the one we are testing with. Algorithm 1 shows the execution workflow.

Figure 5: ShapeFile representation. Line segments of the same colour represent polylines.

Input: vertices ← all vertices within the dataset
Input: n ← the number of vertices
for $i \leftarrow 0$ ***to*** n **do**
  for $j \leftarrow 0$ ***to*** n **do**
    if $distance(point[i], point[j]) \leq DIST\_TRESHOLD$) **then**
      $\mathscr{G} \leftarrow \{polylineId(i), polylineId(j)\}$
    end
  end
end

Algorithm 1: FINDEDGES finds all edges within the graph

The Algorithm 1 shows a naive approach to find the edges between polylines. It obviously is not suited for large datasets, thus we need to optimize the process, or performance becomes a major drawback. In order to speed up the algorithm, we propose the use of a spatial subdivision structure called spatial hashing. The algorithm was chosen due its simplicity and performance over traditional subdivision structures such as quadtrees. We implemented the spatial hashing algorithm proposed by Pozzer [10], which builds the hash table with $O(n)$ complexity. The data structure is built once, and kept in memory for queries.

4.2 Rendering process

The river mesh is generated from the graph by extruding the polyline. Additionally, we generate two footprint textures using a render-to-texture approach, where LOD0 is a high resolution texture around the camera as it moves in the space and LOD1 is a low resolution from the whole terrain. The first texture is periodically updated based on the camera position.

The footprint textures are used to carve the riverbed into the terrain mesh. The process consists on sampling the footprint texture and subtracting the terrain height using the Equation 1, applied for each triangle vertex. Once the river mesh and textures are generated, our approach relies on shader programs to generate river bed directly in GPU.

$$vertex.y -= evaluate(footpTex, texCoord) * riverDepth \quad (1)$$

Current public heightmap databases offer maps from the whole world for free. However, they have mostly 30-90m resolution, posing serious issues. Indeed, consider if we want to represent a 5m wide river on a 30m resolution terrain, the result would be a spiky silhouette with undesirable appearance. There are two ways to compensate this issue: the first is to refine the acquired data as a preprocessing step, and the second is to generate more triangles directly in GPU using the state-of-the-art tessellation shaders. We chose the last one as the first requires pre-processing and results on large databases. Additionally, tessellation can provide small scale details on the fly.

Once the riverbed is prepared in the terrain, we can render the river surface mesh using a water shader. Note that the river location correspondence with the terrain comes from the vector data, which is normally precise.

5 Results and Discussion

We evaluated our solution using a dataset with the features shown on Table 1. The vector data is at 1:50000 scale, while the elevation is made of a 90m resolution SRTM (Shuttle Radar Topography Mission) image and was acquired from an open database at Earth Explorer. The exact position on real world is not mentioned for secrecy reasons. The footprint textures have 4096x4096 pixels, while the LOD0 and LOD1 textures represent, respectively, 1x1 km and 27x27 km areas.

Even though performance wasn't a prior concern, it is an important tool to guide future developments, specially considering the role it plays in this subject area. We performed tests with the data described on table 1 and derived Table 2 from the average of 5 rounds. Our development suite is based on C++ and the OpenGL Shading Language (GLSL) [8] version 4.5 on Windows 10. The host machine has an Intel core i7 870, with 8GB RAM and a GEFORCE GTX 660Ti video card (with *v-synch* enabled).

Table 1: Terrain Settings.

Name	Entities	Vertices	Width	Height	Resolution
Alpha	1760	10308	27777m	27777m	2048

The dataset contains a medium size terrain, where the average edge length for triangles is 13m. However, with this resolution it is impossible to represent rivers narrower than 26m, otherwise the riverbed would be sharp as a 'V' form due vertex distance. Furthermore, since the riverbed would not be properly generated, terrain and river mesh can become close enough to be source of z-fighting. Tessellation plays a crucial role in this case, providing the small scale detail necessary for quality visualization. Figure 6 shows the contrast between the original triangles (left figure) and after the tessellation is applied (right figure). The river with all features implemented by our system is shown in Figure 1.

The texture update policy is distance-based, according to the camera offset, in this case we use 50m. Depending on the application, time-based approaches can be used as well. The approach allows fly-through as the texture update is simply a render-to-texture, there is no need to update any data structure.

Table 2 shows a drastic reduction in the execution times when using the spatial hashing approach. The cell size plays an important role on the query time, when using 100x100 subdivisions we have on average 1.0308 elements per cell, which provides on average $\mathscr{O}(1)$ access cost. Furthermore, the hash building time does not add any significant overhead on the computation. The performance

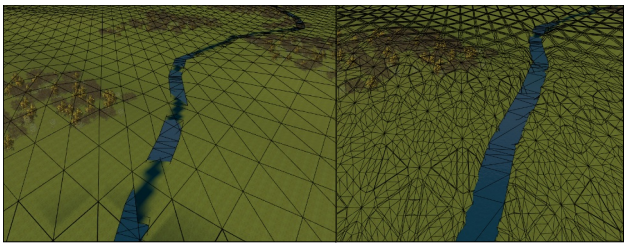

Figure 6: Terrain tessellation impact. The left figure shows original terrain, while right one uses edge-length based tessellation.

speed up over the naive approach is massive, demonstrating the importance and potential of this technique for neighborhood search.

Table 2: Performance statistics

Subject	**Time(Seconds)**
Graph build - naive approach	26.0733
Graph build - hash (10x10 subdivisions)	2.79302
Graph build - hash (100x100 subdivisions)	0.223884
Hash build time	0.00818755
Graphics performance	**FPS**
Average terrain resolution 2048x2048	42
Average terrain resolution 1024x1024	60

On the GPU side, it is noticeable that currently the system does not handle large terrains, specially because there is no LOD system, as it wasn't our focus. The terrain tessellation provides small scale detail, but it doesn't remove vertices. Furthermore, current solution does not provide culling for the river mesh, which may cause vertex data to consume large amount of GPU memory, a solution will be addressed in future works.

A brief performance evaluation indicates that there is an overhead on the geometry shader (GS), which has been reported as source of performance issues[2]. We perform texture fetching and Gaussian filtering to smooth river banks within this stage. Testing the GS without the filter provided an increase of 6 FPS over the previous solution. These evidences point against the GS, however in depth studies are necessary to determine the exact sources of performance issues.

6 Conclusion

We presented a process to generate a virtual scenario with river networks from vector and elevation data. The paper provides a full background that supports the approaches taken, providing detailed information for researches in the field.

The graph building procedure is a key part when working with polyline vector data, as it is originally presented in an unorganized way. In this sense, this paper shows a powerful method to quickly establishing neighborhood relations trough a graph. Even though we use this structure only to build the river mesh, the approach is applicable for road data, where the graph can be used for path finding tasks, for instance.

We took advantage of the latest resources on the OpenGL rendering pipeline to refine terrain mesh directly in GPU, avoiding preprocessing steps that incur in time and storage allocation. Furthermore, tessellation is used to provide small-scale detail and natural constructs in the riverbed.

Finally, we present a solid process that allows quality river rendering. Results show that the approach has potential, and future works are encouraged in order to create a more realistic and optimized solution.

7 Future Works

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

- **Quadtree for resource management.** Quadtree is a promising approach to provide vector data and texture management, allowing more resolution levels for the textures and culling distant rivers.
- **River mesh triangulation.** The study of triangulation techniques to provide quality mesh and robustness for river junctions.
- **River water simulation.** Realistic water simulation. May involve the river graph for flow direction and intensity.
- **Generalization for other vector data types.** The process can be generalized to accommodate polygons, which is how normally large bodies of water (e.g. lakes) are modelled.
- **Hydraulic erosion.** Hydraulic erosion models can be applied to enhance terrain features on river areas, making more natural landscapes.

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] A. Agrawal, M. Radhakrishna, and R. C. Joshi. Geometry-based mapping and rendering of vector data over LOD phototextured 3D terrain models. 2006.

[2] J. Barczak. Why Geometry Shaders Are Slow (Unless you're Intel), 2015.

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

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

[5] B. Deng and D. Xu. Visualization of Vector Data on Global Scale Terrain. (Iccsee):85–88, 2013.

[6] A. Esri and W. Paper. ESRI Shapefile Technical Description. *Computational Statistics*, 16(July):370–371, 1998.

[7] O. Kersting and J. Döllner. Interactive 3D visualization of vector data in GIS. *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, pages 107–112, 2002.

[8] Khronos Group Inc. OpenGL (Open Graphics Library), 2015.

[9] S. Lefebvre and H. Hoppe. Perfect Spatial Hashing. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 579–588, New York, NY, USA, 2006. ACM.

[10] C. T. Pozzer, C. A. de Lara Pahins, and I. Heldal. A Hash Table Construction Algorithm for Spatial Hashing Based on Linear Memory. In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*, ACE '14, pages 35:1—-35:4, New York, NY, USA, 2014. ACM.

[11] M. Schneider and R. Klein. Efficient and accurate rendering of vector data on virtual landscapes. 2007.

[12] A. Szofran. Global terrain technology for flight simulation, 2006.

[13] The USGS Water Science School. Earth's water: Rivers and streams, 2015.

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

[15] Z. Wartell, E. Kang, T. Wasilewski, W. Ribarsky, and N. Faust. Rendering Vector Data over Global , Multi-resolution 3D Terrain. *Proceedings of the symposium on Data visualisation*, (February 2002):213–222, 2003.