Making Trees

Martin Muchembled marmuc@kth.se

I. Introduction

The goal of this project is to implement some aspect of the paper Procedural Tree Modeling with Guiding Vectors (Xu, Mould, 2015) [1], to create tree structures in 3D. The method they describe consists in generating a tree by using shortest path algorithms to create branches following a path between the root of the tree and a given endpoint (end of a branch). Constraints on branch placement are then expressed using different edge cost functions. If this method had already been exposed in different papers, this one presents a new way to compute edge costs, introducing guiding vectors for each node in the computation of the cost of a given edge. The paper is only describing how the branches should be placed, i.e. how to make a wire tree, and there is no mention on how to create a volume from it. Hence my project also includes a basic homemade algorithm to create volume. Moreover, this project includes a homemade, very simple rasterizer.

II. Preparing the environment

A. Homemade Rasterizer

In order to be able to test the actual tree generator, I had to choose among several options of environment. When I could have used something like unity since the purpose of my project is not to implement some rendering technique, I chose to stay in an environment that I already knew, and used the skeleton given in the labs as a starting point. A full tree consisting of a few thousands polygons, I had to implement back face culling in my rasterizer for performance reasons. For a given polygon, define its position $\vec{p}$ as the position of one of its vertex, let $\vec{p}_{cam}$ be the position of the camera, and $\vec{n}$ the normal vector of the triangle then if

$$c = -\frac{\vec{p} - \vec{p}_{cam}}{||\vec{p} - \vec{p}_{cam}||} \cdot \vec{n} < 0$$

the polygon is not drown. This simple rasterizer also includes a clipping algorithm, so that polygons that are not visible on screen are not drown, and the ones that are only partly visible are cut. It is made to only draw triangles, that are sometimes transforms to other polygons when clipped. The space is divided in 9 zones according to the position of the projection of a point on the screen as follow :

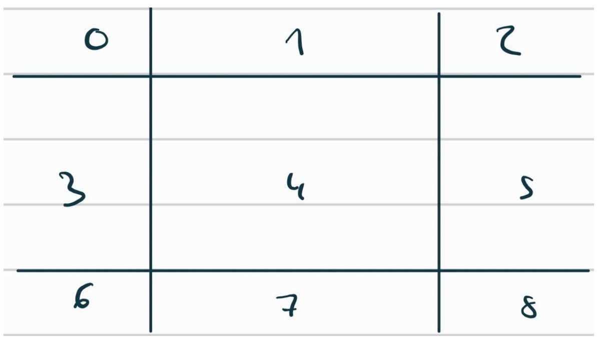

FIG 1 : The 9 zones in space. 4 being the screen by itself.

Then according to the position of each vertex of the triangle, the algorithm computes the polygon that needs to be drawn. If a vertex appears out of the screen, it is projected on the border according to the position of the two other vertices.

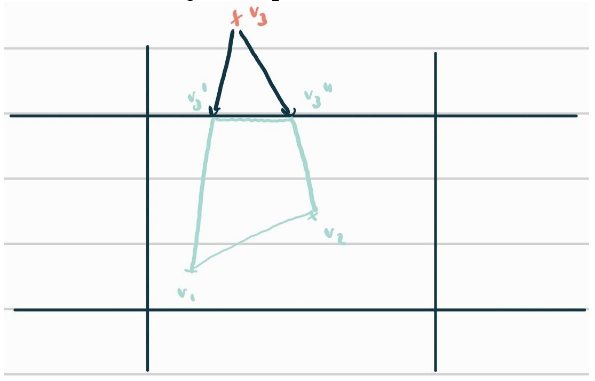

FIG 2 : New polygon from a triangle with one vertex out.

B. Homemade Edge Inflator

The paper is only presenting an algorithm to create a wire like tree shape, i.e. it is not dealing with how to create the actual volume. To address this problem, I created a simple homemade algorithm. Create a cylinder around the edges and a cube between each two consecutive edges. For each $\vec{e} = \vec{v2} - \vec{v2}$, let's create a circle-like polygon B_1 as follow for the base of the cylinder. $\vec{b}_{1,i} = \vec{v}_1 + R \cdot r(0, cos(\frac{2\pi i}{n}), sin(\frac{2\pi i}{n}))$ where n is the number of vertices in the base, R is the rotation such that $R \cdot \dot{x} = \frac{\vec{e}}{||\vec{e}||}$. Using these points, we can then create triangles to complete the base of the cylinder. B_2, the second cylinder base can be created similarly. Then, by joining the two bases we create the faces of the cylinder size. The points of the triangles must be ordered clockwise to be visible with backface culling. The sides of the cylinder are defined as $(\vec{b}_{1,i+1}, \vec{b}_{2,i}, \vec{b}_{1,i})$ and $(\vec{b}_{2,i+1}, \vec{b}_{2,i}, \vec{b}_{1,i+1})$.

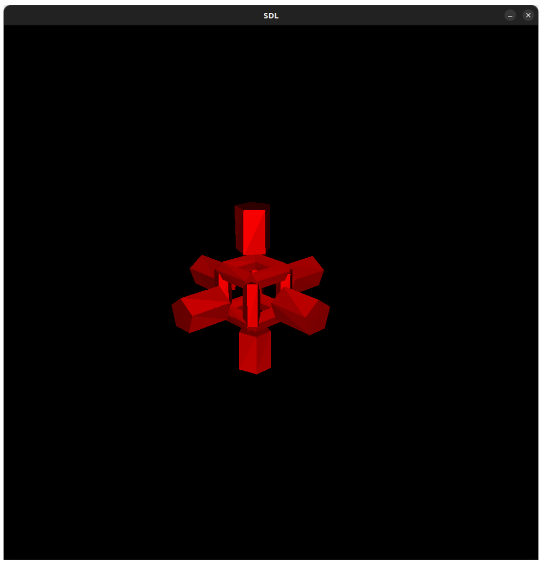

FIG 3 : A few inflated edges.

III. Tree Modeling

Now that we have a suitable environment to work in, we can move to the main part of this project, tree modeling. A tree is built using a graph, according to the following algorithm.

```
GENERATE-TREE(n)
1. Generate the graph $G = (V, E)$
2. Let $\vec{r} \in V$ be the root vertex of the tree.
3. Let $S \leftarrow \{\vec{r}\}$ be the set of sources.
4. Let $D$ be the set of destinations.
5. For $i \leftarrow 1, ..., n$ do
6.     Choose end point as destinations
7.     $P$ the set of vertices in the shortest paths between the sources any source in $S$ and any destinations from $D$
8.     $S \leftarrow P$
9. end for
```

Where n is the number of step of generation. The following subsections will describe the steps of the algorithm in more details.

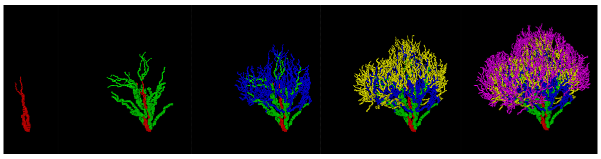

FIG 4 : 5 steps of generation.

A. Graph generation

Since a tree is generating using shortest path algorithm, we need a graph to work with. The vertices of the graph are placed using a Poisson disc distribution, i.e. $\vec{v}_1, ..., \vec{v}_n$ are placed randomly in the 3d space with $d(\vec{v}_i, \vec{v}_j) > r, \forall\, i \neq j$. In order to generate this set of vertices more efficiently, a grid is used to check only nearby points. To ensure that no points are in the same cell we set an upper bound on the cell size: $c \leq \frac{r}{\sqrt{3}}$. From that we populate the grid with 1 if the cell is occupied and 0 otherwise, then to add a new vertex in a cell, we just have to check is the adjacent cells are occupied, and if they are, if the vertex they contain is sufficiently far away.

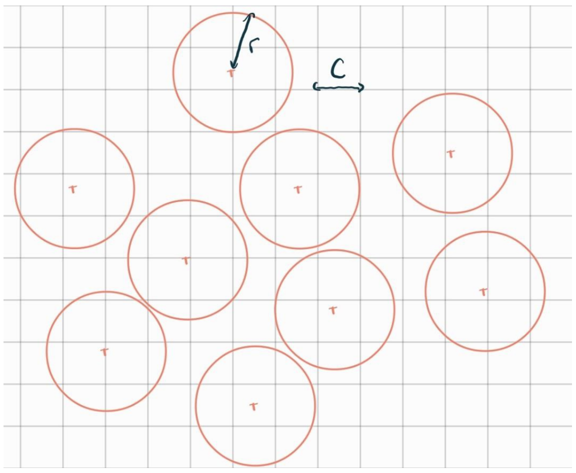

FIG 5 : 2D example of Poisson disc distribution and its grid.

To populate the grid we use the following algorithm :

```
POPULATE-GRID
1. Let $Q$ be a queue
2. Add a vertex v in the grid and to $Q$
3. While $Q$ is not empty do
4.     $v = pop(Q)$
5.     $C = (v_1, ..., v_k)$ k random vertices such that $r \leq d(v, v_i) \leq 2r$
6.     for $v'$ in C do
7.         if CHECK-NEIGHBORHOOD(v') then add $v'$ to the grid and to $Q$
8.     end for
9. end while
```

The algorithm stops when it is not possible to add a point anymore from those already added. Note that this is not perfect, some space might still be available for a new vertex, choosing higher value of k lead to less and less holes in the vertices.

For each vertex, we split the space in 8 parts, according to the three dimension in space. For a vertex $\vec{v} = (v_x, v_y, v_z)$ $S_{\vec{v}} = (\{\vec{u} \in \mathbb{R}^3 : u_x \leq v_x, u_y \leq v_y, u_z \leq v_z\}, \{\vec{u} \in \mathbb{R}^3 : u_x > v_x, u_y \leq v_y, u_z \leq v_z\}, ..., \{\vec{u} \in \mathbb{R}^3 : u_x > v_x, u_y > v_y, u_z > v_z\})$ the partition of this space. Then we create a directed edge in the graph from $\vec{v}$ to the closest vertex in each of these parts, if it exists, so each vertex has at most 8 incident edges. This is called Yao-8 [2] graph. In this project, the edges are found as the shortest path algorithm reach the incident vertices to save memory space.

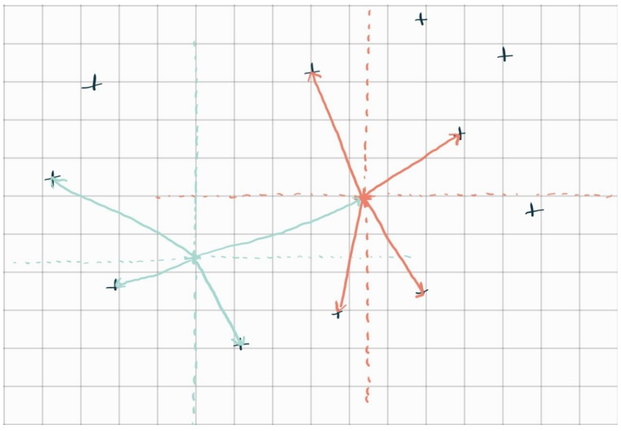

FIG 6 : 2D Example of edges construction around two vertices.

B. Shortest Path computations

The computation of the shortest path between to vertices is done using Dijkstra algorithm, the particularity of this implementation is the lazy evaluation of the edge and their cost, i.e. the edges and their costs are found as the computation of the shortest path go. When a node is reached, we find its incident outgoing edges, compute its guiding vector and with that the cost of each edges. Details about guiding vectors computation are given in the next subsection. For a given edge $e = (\vec{u}, \vec{v})$ and $\vec{u}$'s guiding vector $\vec{g}_u$, we set the cost of e as :

$$c_e = ||\vec{v} - \vec{u}||(1 - \frac{\vec{v} - \vec{u}}{||\vec{v} - \vec{u}||} \cdot \vec{g}_u)$$

$\vec{g}_u$ is a unit vector so the second factor is never negative. The cost of an edge of length l is then 0 when the vertex's guiding vector and the edge share the same direction, is l when they are orthogonal, and $2l$ if they are in opposite directions.

C. Guiding vectors

For a given vertex $\vec{v}$ guiding vector are computed, by default, as a rotation of constant angle t of the parent's guiding vector around the rotation axis defined as the cross product of the unit vector pointing up and the parent's guiding vector, i.e. if the path P leading to $\vec{v}$ is $P = ... - \vec{p} - \vec{v} - ...$ then $\vec{g}_v$ is the rotation of t of $\vec{g}_u$ around $\vec{r} = (0, 1, 0) \times \vec{g}_p$. If in the default version of the algorithm, t is a constant, we can also use a function $t(s)$ that gives a rotation angle as a function of the steps from the source, or $t(\vec{v})$ that gives a rotation angle as a function of the position of the vertex. Here is an interesting example of such function

$$t_1(\vec{v} = (v_x, v_y, v_z)) = \begin{cases} t & \text{if } \sqrt{v_x^2 + v_z^2} < T \\ -t & \text{otherwise} \end{cases}$$

where the rotation is inverted after a given distance T from center of the tree.

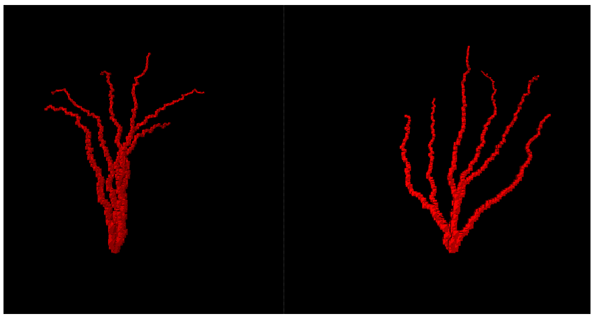

FIG 7 : The first generation step of two trees generated on the same graph with constant t (left) and $t_1(\vec{v})$ (right).

D. End point sampling

We still have to discuss how the endpoints are chosen.

1) Volume sampling: For the first step, we define a volume that we want to sample from. We take every vertex from this volume and keep some of them to be the first destinations. Different volume will give different tree shape. For example, defining a 3D annulus with a large inner radius will give a wider tree than a sphere.

2) Border sampling: For the next iterations, we sample the vertices from the set of vertices at h hopes from the existing structure. Let $\mathcal{T}$ be the set of vertices of the existing tree, $\vec{u} \in E$, and B_h the set mentioned above

$$\vec{u} \in B_h \iff \min_{\vec{v} \in \mathcal{T}}(|P_{v \to u}|) = h + 1$$

where $P_{v \to u}$ is the shortest path in terms of edge count from $\vec{v}$ to $\vec{u}$. h is a parameter that can be different for each step. A larger h will result in longer branches.

IV. Conclusion - What to do next ?

The generation is done using a graph where nodes are placed using a Poisson disc distribution and are linked with directed edges in a Yao-8 graph. It is an iterative process. In the first step we choose a node as the root/source of the tree and one or several points as end points destination from a given volume. Then we connect the connect the source to the destinations using a shortest path algorithm, here Dijkstra. The cost of each edge is computed using the edge length, the edge direction and the guiding vector of the node at the begining of the edge. The guiding vector represent the prefenrential direction of the branch going out of a node. Its computation is part of the parameters of the program. For the next steps, we choose all or a part of the nodes in the existing path as source nodes and some points at a given number of edges from the path as destinations and repeat the shortest path computation.

The number of endpoint at each step, the volume to sample the first destinations from, the number of jumps from the existing path, and the guiding vector computation function are some of the parameters that we can tweak to obtain different tree shapes using the same algorithm.

This project was challenging by its size, so I sometime had to sacrifice some quality in my code / algorithms, and to give up on some features to be able to complete it, however the code base (even if it is not always clean) is sufficiently modular to implement more variants of the algorithm quickly. Spending more time to explore different volume to sample from and different guiding vector computation could be a good way to push this project further. Also, the choice I made trying to create my own clipping algorithm and my own algorithm to give volume to the tree was, even if it allowed me discover concepts by myself, no the best thing to do. Implementing better techniques for these two points could be a huge improvement.

References

[1] 1. Ling Xu, David Mould. Procedural Tree Modeling with Guiding Vectors. Carleton University, Ottawa, Canada, 2015.

[2] 2. YAO A.: On constructing minimum spanning trees in k-dimensional spaces and relatedproblems. Tech. rep., Stanford University, 1977.