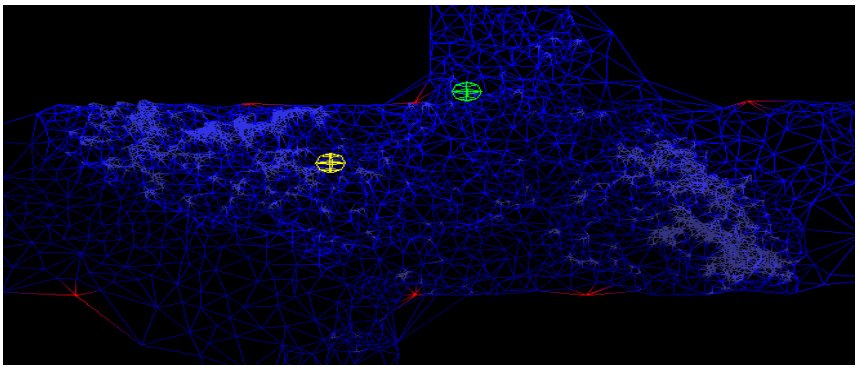
2D Fluid Simulation based on Voronoi Regions

Abhishek Venkatesh
venky@gatech.edu

Jeonggyu Lee
glaze@gatech.edu

CS7491 3D Complexity, Project 4
Advisor Prof Jarek Rossignac



Abstract: Animating fluid like motions is an important and challenging task in the field of Computer Graphics. In this paper, we represent a simple approach to animate fluids in closed boundaries based on Delaunay Triangulation. We construct Delaunay Triangles using the incremental scheme with our suggested improvement in [1]. We adjust the velocities of the particles based on their Voronoi region area. Unlike other fluid simulations based on the standard Navier-stoke equation, our goal is to render the fluid animation in real-time with more importance on visual appearance rather than physical simulation.

Introduction

Fluid simulations is a very interesting and active area of research in computer graphics. In scientific aspect, we want to simulate exact fluid dynamics. On the other hand, in Computer Animation, it is acceptable if it produces plausible result even though it has some inaccuracies. Most fluid dynamics are based on Navier-Stoke equation which represents movement of fluids in terms of vector fields. This approach usually assumes particles on the middle of the grids. Our fluid simulation is built on top of Delaunay triangulation of points. In [1] Venkatesh and Lee suggested a simple improvement for speeding up the incremental Delaunay triangulation algorithm. Instead of grids we compute the Voronoi region of the fluid particles from the Delaunay triangulation algorithm suggested in [1]. In each frame, a particle exerts a force on its neighbors whose magnitude is inversely proportional to the Voronoi region of the particle. We give the exact equations later in the paper.

Section I describes the representation of boundary and fluid, Section II provides the main idea and details of our proposed algorithm for fluid animation, Section III describes how we compute the Voronoi region from the Delaunay Triangulation, Section IV mentions the results

along with a screenshot and Section V gives the conclusion and future work.

Section I: Representation of boundary and fluid.

Representation of the boundary:

The boundary is represented as a set of points. The inside of the boundary is triangulated and stored. Note that the incremental Delaunay triangulation requires a triangle 'T1' which encompasses all the points to be triangulated. For our purpose this initial triangulation of the boundary points replaces that triangle 'T1'. The fluid particles are triangulated using the incremental scheme for Delaunay triangulation into this initial triangulation. An example of triangulated boundary is shown in figure 1.

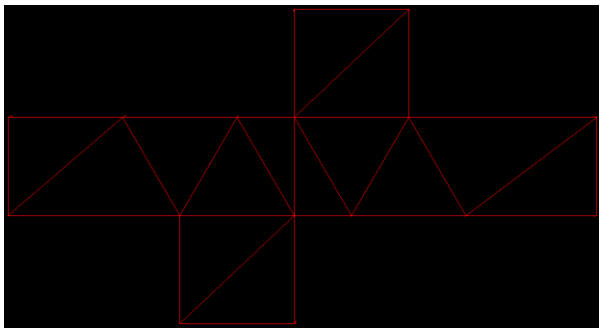


Figure 1: Boundary with interior triangulated

A point to note is that the boundary edges will not be flipped in the incremental Delaunay triangulation based on edge flips. This is because the corresponding corners of boundary edges have no opposite. For this purpose we may have to hand craft the initial triangulation. The other option is to perform a Delaunay triangulation of the boundary points and remove the unwanted triangles.

Representation of Fluid

The fluid is represented by a set of points (particles). Each particle is given an initial velocity which will then be updated every frame based on the surrounding particles. The initial position of the particles in our animation was generated randomly in selected parts of the boundary (See figure 2).

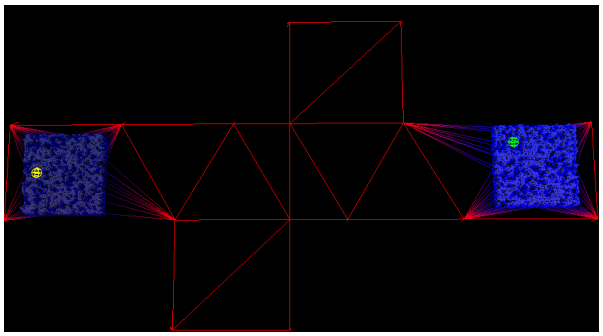


Figure 2: Initial position of fluid particles in blue

Section II: Computation and Animation of the fluid particles

The main steps involved in our fluid animation are as follows:

1) Perform a Delaunay triangulation of all the particles (including the boundary) as a set of points. Since the boundary is already triangulated we simply perform the triangulation by inserting the fluid particles one by one into the initial triangulated boundary.
2) Compute the Voronoi regions of the fluid particles from the dual of their Delaunay triangulation. Let the Voronoi region area of the i^{th} fluid particle be denoted as 'VA_i'.
3) The velocity vector of each particle is updated based on the following equation:

$$Velocity[i] = VoronoiVector[i] + \alpha * Velocity[i] + \beta \sum Velocity[neighbor(i)]$$

where α and β are constants which can be varied. For our simulation the value of α used is 0.9 and β is 0.1.
Let V_{ni} be the vector denoting the position of the 'i^{th}' particle minus the position of its neighbor 'ni'.

VoronoiVector[i] =
$\sum$Vni. Scaled(|Velocity[ni] |/(|V_{ni}|*VA_{ni}))

The above equation is based on the following assumptions

a. The force exerted by a particle on its neighbor is inversely proportional to its Voronoi Area.
b. The force exerted by a particle on its neighbor is inversely proportional to its distance from the neighbor.
c. The force exerted by a particle on its neighbor is directly proportional to its own velocity along the direction of the vector V_{ni} from the particle to the neighbor.

4) Assuming constant frame rate, the position of each particle is updated by simply adding its velocity vector to its current position.
5) If the position of the particle goes out of the boundary it is pushed back inside the boundary and its velocity direction is reversed with reduced magnitude.
6) Though our animation is 2D, we do give some height to some particles which have very small Voronoi area. This is based on the fact that if the Voronoi area is small then there are lots of particles cluttered around that area. Also these particles give an appearance of waves. A simple method to render these waves is by giving a height to these particles which is inversely proportional to the Voronoi Area. Also the white component of the color assigned to these particles is inversely proportional to their Voronoi Area. (figure 3)
7) Finally the fluid is rendered by displaying the triangulation. Though we could do just particles disks or do environment mapping. We plan to do these in future. For now the triangulation gives a little bit more insight on the actual working of the algorithm.

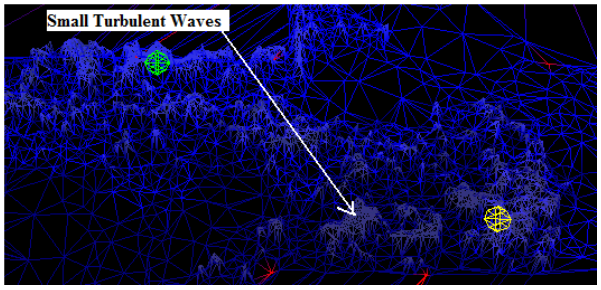


Figure 3: The particles with small Voronoi regions are given a height inversely proportional to their Voronoi Area

Section III: Computing the Voronoi Area of the particle

The Voronoi diagram for a point set S is the partition of the plane which associates a region V(p) with each point p in S in such a way that all points in V(p) are closer to p than to any other point in S. The Delaunay triangulation and the Voronoi diagram are duals of each other. To generate the Voronoi region for a point, simply join the circumcenters of the triangles which the vertex is part of in clockwise or anti-clockwise order as shown in figure 4.

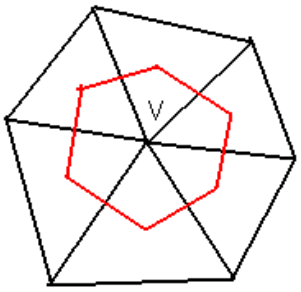


Figure 4: The red line joins the circumcenters of the triangles which is the vertex is part of. The enclosing region inside the red loop is the vornoi region for the vertex V.

To compute the Voronoi area of all the particles using the corner table we use the following piece of code

```
for(ssize_t i=0;i<number of corners;i++){
 if(O[i]==-1) {
  vec P = midpoint(G[V[n(i)]]+G[V[p(i)]])
```

```
if(G[V[i]].isLeftOf(G[V[n(i)]],G[V[p(i)]])==CC
[i/3].isLeftOf(G[V[n(i)]],G[V[p(i)]]))  {

VoronoiArea[V[p(i)]]+=(tri2DArea(G[V[p(i)]],CC
[i/3],P));
VoronoiArea[V[n(i)]]+=(tri2DArea(CC[i/3],G[V[n
(i)]],P));
 } }
 else if(i<O[i]) {
VoronoiArea[V[p(i)]]+=(tri2DArea(G[V[p(i)]],CC
[i/3],CC[O[i]/3]));
VoronoiArea[V[n(i)]]+=(tri2DArea(CC[i/3],G[V[n
(i)]],CC[O[i]/3])); }}
/* here CC[] stores the circumcenter of the
correspong triangle of the corner */
```

In this piece of code if the corner does not have opposite then the Voronoi area of the next and previous corner vertex is calculated (updated) only if the circumcenter of the triangle lies on the same side of the corner with respect to the previous and next corner. (figure 5). If the opposite is present the area is updated as shown in figure 5.

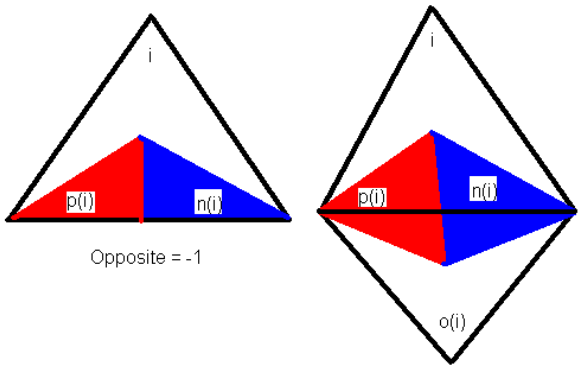


Figure 5: Calculating Voronoi area of the particles by traversing the triangulation using the corner table operations. The red triangle area is added to the Voronoi area of p(i) and the blue triangle area is added to n(i). The tip of the red and blue triangles are the circumcenters of the triangle (top) with corner 'i' and the triangle with corner o(i)(bottom).

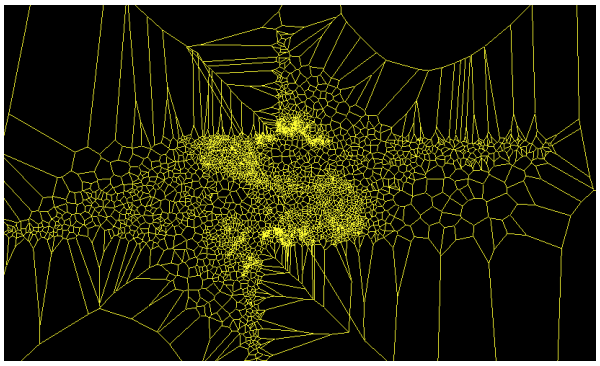


Figure 6: A screen shot of the voronoi diagram computed at each Frame for the fluid particles

Section IV: Results

For this project we used a machine with the following configuration:

Intel(R) Core (TM) 2 Duo CPU 1.6Ghz(2CPUs), CPU Memory: 2046 MB RAM, Nvidia GeForce 8400M GS with 128MB video memory. OpenGL was used as the graphics API library.

We used 2400 particles (a total of 4800 particles + the boundary particles) for each of the two blocks of fluid. We were able to generate an average frame rate of 13 frames per second.

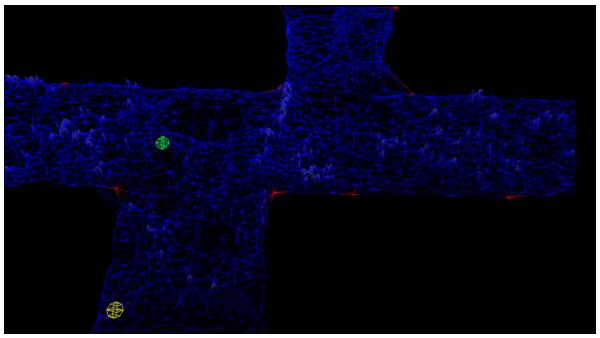


Figure 7: A screen shot of the fluid animation.

Conclusion and Future Work

In this paper we presented a simple fluid animation approach. Instead of the traditional grid based approaches our approach is based on fast Delaunay triangulation of the fluid particles. Our simple equation (in section II) for updating the velocity of the particles based on the Voronoi

Area gives visually satisfactory results and more importantly works real time. Though we implemented our fluid simulation in 2D we proposed a simple idea to give height to particles with small Voronoi areas. This gives an effect of waves and adds a volume effect to our fluid simulation. The whole approach is extensible to 3D though it might not be real-time because we do not have efficient triangulation algorithms in 3D.

For demonstration purpose we used 4800 particles and were able to get a descent frame rate. But as we scale up we might not be able to generate a descent frame rate as the triangulation time will increase. We plan to use hardware acceleration using GPU programming to achieve better frame rate and scale up our fluid animation for more particles.

Although our fluid simulator produced interesting fluid-like motions, we need to consider following aspects.

Accuracy – At each frame our fluid simulator merely move particles according its Voronoi region. After the movement, there is no further process to correct the position of the particle. We have to simulate fluid features like "incompressibility" by adding relaxation step after each frame to ensure its correctness.

Rendering – We are rendering particles using colored line segments. To achieve more visually appealing result, we will consider advanced rendering techniques such as volume rendering, environment mapping and alpha color blending as future works.

Acknowledgement

We would like to thank Prof Jarek Rossignac and Jason Williams for suggesting that our improved Delaunay Triangulation algorithm can be used to animate fluids in real-time.

References

[1] Venkatesh, Lee :An improved Incremental Delaunay Triangulation Algorithm

[2] Rossignac, Safonova, Szymczak 3D-Compression Made Simple: Edgebreaker on a Corner-Table

[3] P. Cignoniz, C. Montaniz, R. Scopigno, DeWall: A Fast Divide & Conquer Delaunay Triangulation Algorithm in E^d.

[4] Wikipedia, the free encyclopedia, Delaunay triangulation http://en.wikipedia.org/wiki/Delaunay_triangulation

[5] Lischinski, Incremental Delaunay Triangulation

[6] Shewchuk, Triangulation Algorithms and Data Structures. A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator (http://www.cs.cmu.edu/~quake/triangle.html)

[7] Rossignac, Wiliams, CS-7491-A: 3D Complexity Techniques for Graphics, Modeling, and Animation. http://www.gvu.gatech.edu/~jarek/courses/7491/

[9]R. Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, editor, New Trends in Discrete and Computational Geometry, volume 10 of Algorithms and Combinatorics, pages 37–68. Springer-Verlag, New York, 1993.

[10] Guibas and Stolfi, Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi.

[11] Jos Stam, Stable Fluids.
http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/ns.pdf

[12] Robert Bridson, Matthias M¨ uller-Fischer, FLUID SIMULATION. SIGGRAPH 2007 Course Notes