Bilkent University

Department Of Computer Engineering

CS565

Computational Geometry

Term Project

Real-Time Path Planning for Multi-Agents In Dynamic Environments

By

Ateş Akaydın

Supervisor: Assoc. Prof. Dr. Uğur Güdükbay

Progress Report II

May 20, 2008

Table of Contents

1. Introduction

Real-time path planning for multiple-agents in dynamic environments is a challenging issue that has many applications in the fields of computer graphics.

In computer graphics, this problem is important for simulating crowds and crowd behavior in a realistic manner in dynamic virtual environments. Virtual crowd simulation has applications in emergency evacuation, architecture design, urban planning, personnel training, education and entertainment. In robotics, the same problem occurs in multiple-robot coordination and planning in dynamically changing physical environments.

2. Related Work

In terms of computer graphics terminology, there are several approaches to model the problem which can be classified as: Social Forces Models, Cellular-Automata Models and Rule-Based Models.

Social Forces Models make use of attractive and repulsive forces to simulate interactions in between agents and obstacles. They are much like the potential field approaches used in robotics literature. An important, recent approach in this context was proposed by Treuille et al [2].

In Rule-based methods, contact in between obstacles and agents are avoided by ordered application of rules such as wait rules. Hence, collisions need not to be handled. Therefore, Rule-based methods fail to simulate natural behavior like pushing. Some examples of Rule-based methods are given in [3] [4].

On the other hand, in Cellular-Automata models, world space is discretized as a regular grid. Agents can only move into free grid cells and therefore collision behavior can't be simulated accurately. High-Level goals of the agents are also embedded in to these grid cells and each agent can use this information to accomplish goals. Cellular-Automata models are fast and easy to implement. Several approaches that make use of these models are given in [5] [6].

In addition there are some hybrid models which make use of several of these three types of models. An example of such an approach was proposed by Pelechano et al [1]. In their so called "HiDAC" model which adopts various aspects of the social forces model although it is heavily rule based.

In order to navigate in complex environments, these models require some high-level representation of the subject environment. Some popular representations include roadmaps, cell and portal graphs and potential fields. Information considering the properties of the environment and high-level goals can be pre-computed and integrated with the environment representation for real-time simulation purposes.

One of the most recent and novel approaches to the problem was made by Avneesh Sud and his colleagues in their paper, namely *"Real-time Path Planning in Dynamic Virtual Environments Using Multi-agent Navigation Graphs"* [8]. In their approach authors make use of 1st and 2nd order Voronoi diagrams to determine a collision free path(towards the gloabal goal) for each agent within the environment.

In addition a Social Forces model similar to Helbing's Model [7] was implemented in order to do local planning with respect to the nearby agents.

3. Objectives

Having taken *"Real-time Path Planning in Dynamic Virtual Environments Using Multi-agent Navigation Graphs"* [8] and "Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware" [9] as ground works, I've studied on finding ways to do navigation planning for large number of agents in dynamic environments in real-time and implemented the proposed approach of [9] in order to generate Voronoi Diagrams for path planning at interactive rates.

4. Proposed Approach

The approaches proposed in two noteworthy papers "Real Time Path Planning in Dynamic Virtual Environments Using Multi-Agent Navigation Maps" by Avneesh Sud et al [8] and "Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware" by Kenneth E. Hoff et al [9] will be used during the implementation of this project. Basically a Multi-Agent Navigation Graph (MaNG) will be constructed at each simulation step from the 1^{st} and 2^{nd} order Voronoi Graphs computed by the graphics hardware [9]. MaNG is just a unification of these two Voronoi Graphs and it gives information about the pairwise proximity of agents and obstacles within the system with respect to each other. In the proposed system, agents and obstacles both form the sites of the Voronoi Diagrams. Convex-Polygonal obstacles will be used in order to provide a connected Voronoi Graph. Detailed information of constructing the MaNG is given in section 6.

Each agent within the system is given a global goal. Local behavior and interactions with respect to other agents, objects and obstacles are possible as well. Path planning for the global goals are done on the MaNG by using an optimal graph search algorithm, like the A^*-Search.

Local behavior will be made possible through a social forces model like Helbing's Model [7]. Basically, a potential field will be derived which defines attractive and repulsive forces locally within the Voronoi Sites so that the agents can properly behave against emergent entities such as other agents and obstacles. The parameters of the force equations uniquely define different emotional characteristics of the agents. In addition if possible and necessary, agents will try to interact with entities within a variable distance through the abstract interaction routines. A unique interaction routine will be assigned to each agent so that it can interact only with a particular set of entities of interest. Ways of interaction is variable and they are precisely defined by the interaction routines. As an example if an agent is assigned a fireman routine, it may try to extinguish fire wherever it spots it. On the other hand if civilian routines are assigned, agents will try to escape from fire.

One important problem that occurs especially in large scale virtual environments (complete city models for instance) is the difficulty of simulating the agents within the

entire navigable area. Due to the scale of the environment, simulating and keeping track of the agents in such situations would require excessive amounts of computational power and storage. So the regions where the agents will be simulated should be restricted. In addition, to create a reasonable crowd simulation, agents can be generated (or picked from a pre-allocated pool of agents) at the borders of the restricted region of focus, having goals towards the borders across. Those agents which reach to their goals will be put back to the agent pool. This would provide a constant flow of agents within the virtual environment so that a live simulation can be created. MaNG Graph (and 1st,2nd order Voronoi Diagrams) will also be computed only for the agents and obstacles within the current region of focus. The region of focus should also be dynamic and it should change whenever the current position of the camera is changed. Agents which fall outside of this region will also be put back to the agent's pool.

Finally, extensive scenarios will be developed to simulate the behaviors large numbers of agents having different characteristics. For example, one particular scenario can be emergency evacuation. In this scenario, an explosion in the virtual city will occur and those agents, who are close to the event, will be panicked and they will try to evacuate the zone as fast as possible. At the same time, agents that are specialized in dealing with the event, such as fireman will enter the zone and try to extinguish the fire occurring due to the explosion.

5. Background & Notation

In this section, notations and equations regarding the theoretical background of the approach are given. Notations presented in this section are the most important of those given in [8]. Please refer to the paper for other aspects of the below formulations along with Lemmas depicting several important properties of Voronoi Regions, Diagrams and Graphs (and their proofs).

A geometric primitive, an agent or an obstacle in d-dimensions is called a site. Sites can be points, edges, triangle or more complex polygons. In this work, only 2-dimensonal sites are considered as planning is done for 2-dimensional floor of the environment. Here a particular site i is represented with the symbol p_i. And the center of mass of the site is represented with $\pi(p_i)$. The interior and boundary of a set of points S is given as $Int(S)$ and $\partial(S)$ respectively.

Given as site p_i, $d(q, p_i)$ denotes the Euclidian distance between a particular point $q \in \Re^d$ to the closest point in the site p_i.

Having defined the basic symbols we can begin with defining a k-th order Voronoi region as:

$$Vor^k(T \mid P) = \{q \in D \mid d(q, p_i) \leq d(q, p_j) \forall p_i \in T, p_j \in (P - T)\}$$

Where P is the set of all sites and T is as k-tuple subset of P where $|T| = k$. This basically defines all points in P that are closer to T than any other remaining sites in P. Hence the k-th order Voronoi Diagram VD^k can be defined as follows:

$$VD^k(P) = \bigcup_{p_i \in P} Vor^k(T, P), |T| = k.$$

Voronoi Diagram is hence the union of all points which are closer to a particular k-tuple site T -than any other sites-, for all possible combinations of such k-tuples.

We then define the k-th order governor set of a particular point q which is the set of closest k-tuple sites to q . This is shown as:

$$U = \{T_0, T_1, ..., T_m\}, |T_i| = k, q \in Vor^k(T_i \mid P)$$
$$Gov^k(q \mid P) = U$$

Finally the k-th order Voronoi Graph (VG^k) can be generalized as follows:

$$VG^k = (V, E)$$

where

$$V = \{v \in D \mid \left|Gov^k(v \mid P)\right| \geq 3\}$$

$$E = \{e \mid e = (v_{1,} v_2), v_{1,} \in V, v_{2,} \in V, \exists c \mid c = Vor^k(p_i \mid P) \cap Vor^k(p_j \mid P), v_{1,} \in c, v_{2,} \in c,$$

$$p_i \in Gov^k(v_1 \mid P),\ p_j \in Gov^k(v_2 \mid P)\}$$

Having defined the notation, we can now begin with the Multi-Agent Navigation Graph (MaNG). Local and global path planning on the MaNG graph is given in section 6 along with the ways of constructing MaNG. Constructing the MaNG data structure requires computation of the 1st and 2nd order Voronoi Diagrams. Computations of General Voronoi Graphs (GVDs) are done in Graphics Hardware and are explained in detail in section 7.

6. Path planning Using Multi-Agent Navigation Graphs (MaNG)

This section explains the MaNG data structure, its properties and usage in path planning in detail.

a. Multi-Agent planning using Hybrid Voronoi Structures

Each moving agent within the system is considered as a dynamic obstacle for the remaining agents. The task is to compute a global navigation data structure (which is the multi-agent navigation graph (MaNG)) which provides clearance and proximity information for each agent. For each agent the path to its global goal should be maximally clear of other obstacles and agents. A data structure that unifies 1st and 2nd order Voronoi Graphs can hence provide a pairwise maximally clear path for each agent within the data structure. The actual paths to the goal points can be searched on this data structure. Multi-Agent Navigation Graph is such a graph that has this property. Basically it is the union of the first order Voronoi Graph $VG^1(P)$ and a subset of the second order Voronoi Graph $VG^2(P)$. The subset includes intersection of $VG^2(P)$ and the Voronoi Regions ($Vor^k(p_i \mid P)$) of the agents within the system. Voronoi Regions of the static obstacles are disregarded since no pair-wise proximity information is necessary to direct an agent around an obstacle (due to the fact that the obstacle is static). The paths on the first order Voronoi Graph would be enough for this case.

Formally the MaNG can be defined as follows:

$$MG(P) = (V, E), where$$
$$V = \{v \mid v \in V^1 \cup (V^2 \cap Vor(p_i \mid P)) \forall p_i \in P_a\}$$
$$E = \{e \mid e \in E^1 \cup (E^2 \cap Vor(p_i \mid P)) \forall p_i \in P_a\}$$
$$VG^1(P) = (V^1, E^1)$$
$$VG^2(P) = (V^2, E^2)$$

Where P_a is the set of agent sites and $P_a \subseteq P$.

A color is assigned to each edge and vertex in $MG(P)$ with respect to their membership in $VG^1(P)$ and $VG^2(P)$. Edges and vertices from first order Voronoi Graph $VG^1(P)$ are colored **red** and edges from second order Voronoi Graph $VG^2(P)$ along with the vertices in $VG^2(P) - VG^1(P)$ are colored **black**. Finally each edge in $MG(P)$ is assigned a cost of travelling on this particular edge (distance for instance). It can be shown that $MG(P) \subseteq NG^2(P)$ where $NG^2(P)$ is known as the 2nd nearest neighbor graph(please see [8] for proof). In the below figures(1..5) examples of first(figure-1) and second(figure-2) order Voronoi Diagrams, 2nd order nearest neighbor diagram(figure-3), 2nd order nearest neighbor graph(figure-4) and the multi-agent agent navigation graph- MaNG -(figure-5) are given. In each figure black points represent agents and white points represent obstacles.

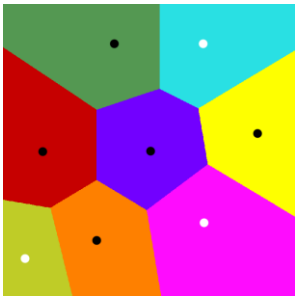

Figure-1

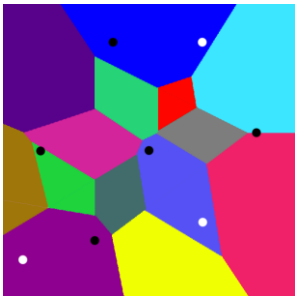

Figure-2

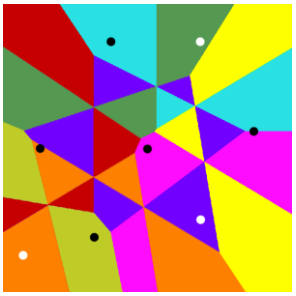

Figure-3

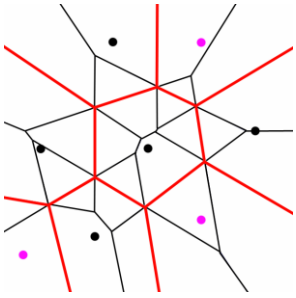

Figure-4

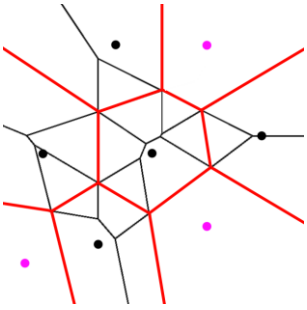

Figure-5

b. Multi-Agent Simulation

In this section, path planning with MaNG and local dynamics model is given.

To begin with, a local dynamics model, largely inspired from the Helbing's Social Force Model [7] is developed. In addition to the repulsive and attractive forces defined in the actual model an additional force F^r is applied to each agent, and it basically directs the agent to the next vertex on the path towards its global goal (computed on the MaNG). In addition to F^r, there are three primary forces namely F^{soc}(repulsive force applied from nearby agents), F^{att}(attractive force which forms up groups), F^{obs}(repulsive forces applied by the nearby obstacles).

Given a particular agent p_i, Voronoi Region $Vor(p_i)$ and some point p in this region, force field at this particular point p can be evaluated with the below formulae:

$$F(p) = \sum_j \left[F_j^{soc}(p) + F_j^{att}(p)\right] + F_i^r(p) + \sum_o F_o^{obs}(p)$$

$$p_i \in P,\ j \neq i, o \in O$$

where

$$F_j^{soc}(p) = A_i \exp^{(2r_a - \|p - x_j\|)/B_i} n_j(p) \times (\lambda_i + (1-\lambda_i)\frac{1+\cos(\phi_j(p))}{2},$$

$$F_j^{att}(p) = -C_j n_j(p),$$

$$F_o^{obs}(p) = A_i \exp^{(2r_a - d(p,o))/B_i} n_0(p) \times (\lambda_o + (1-\lambda_o)\frac{1+\cos(\phi_o(p))}{2},$$

$$F_i^r(p) = \frac{v_i^d(p) - v_i}{\tau_i},$$

$$v_k^d(p) = v_{\max} e_k(p)$$

where

$$e_k(p) = (v_i - \pi(p_i))/\|v_i - \pi(p_i)\|$$

Where A_i and B_i are constants denoting strength and range of repulsive interactions respectively. C_j is the constant used to determine the strength of attractive interactions. These constants play an important role on the individual behavior characteristics. λ_i is used to define an anisotropic character for the agent interaction. Since the obstacles may be dynamic, such a term is needed to bias the repulsive forces along the motion of the obstacles.

Force field $F_i^r(p)$ directs the agent along the shortest path computed by using the algorithm provided below:

Input: Agent p_i, Goal position $\mathbf{g_i}$, Set of sites P, MaNG MG(P)
Output: Path S_i from current position to goal position

$k \leftarrow$ LocatePoint($\mathbf{g_i}$)
if $k = i$ **then**
$S_i \leftarrow edge(\pi(p_i), \mathbf{g_i})$
return
Compute $V_i \leftarrow$ set of black vertices in Vor(p_i|P)
Compute $E_i \leftarrow$ set of black edges in Vor(p_i|P)
if V_i 6= /0 *and* $\pi(p_i) /\in V_i$ **then**
Augment MG(P) with green edges,
$e_j = (\pi(p_i), v_j) \forall v_j \in V_i$
Assign weight to e_j ,$w(e_j) \leftarrow d(\pi(p_i), v_j)$
else
foreach *edge* $e_j \in E_i$ **do**
Compute $v_j \leftarrow$ closest point on e_j to $\pi(p_i)$
Augment MG(P) with green edge $e_j = (\pi(p_i), v_j)$
Assign weight to e_j ,$w(e_j) \leftarrow d(\pi(p_i), v_j)$
end
Compute $V_k \leftarrow$ set of red vertices in Vor(p_k|P)
Augment MG(P) with green edges $e_j = (\mathbf{g_i}, v_j) \forall v_j \in V_k$
Assign weight to e_j ,$w(e_j) \leftarrow d(\mathbf{g_i}, v_j)$
Add green labels to each edge $e_j \in E_i$
$S_i \leftarrow$ ShortestPath (p_i,$\mathbf{g_i}$,MG(P))
Remove green labels from each edge $e_j \in E_i$
Remove all green edges from MG(P)

Basically $F_i^r(p)$ directs the agent p_i to the first vertex of the shortest path S_i. During the execution of the above algorithm, additional edges (colored green) connecting the position of the agent p_i and some closest black vertex are drawn on MaNG as well as an initialization step. These edges form the direct link from the position of the agent p_i to the entrance point of MaNG(which are the black vertices).

A possible path on the MaNG is given in the below figure-6, where the blue line represents the path. Green lines show the initial edges added as described above. Notice that green edges are also supported for the goal point(to connect goal to the graph) along with the initial starting point.

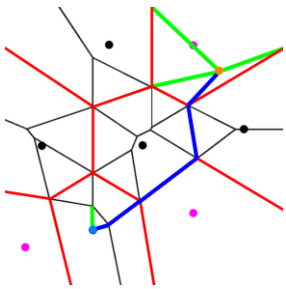

Figure-6

7. Computation of GVDs Using Graphics Hardware

The approach proposed in [8] requires generation of the GVDs at interactive rates. The authors referred to [9] for a solution to this problem.

First and second order Voronoi Diagrams are computed in the graphics hardware. This section briefly discusses the computation process and implementation details. The process is given with its entirety in [9].

a. Brute-Force Approach

The brute-force way to compute a discrete Voronoi Diagram would be to point sample the space containing the Voronoi sites. To do so we first generate a grid of sample points for a particular site. For each sample point in the grid, we find the distance to the closest point of the subject site and store in the grid. We do this for each available site.

Basically the algorithm iterates through the sites then for each site discrete approximation to the distance function is made and a current closest site and distance grid is updated with respect to the distance functions of the sites. The end result will resemble the Voronoi Diagram of the sites. This approach can be implemented with standard Z-buffered raster graphics hardware as well. Examples of sampled distance functions for two point sites are given in figure-7. In figure-8 the two distance functions are composited through the distance comparison operation. The second figure of figure-8 depicts the end result where each point is classified with a color ID, with respect to the site which it is closest to.

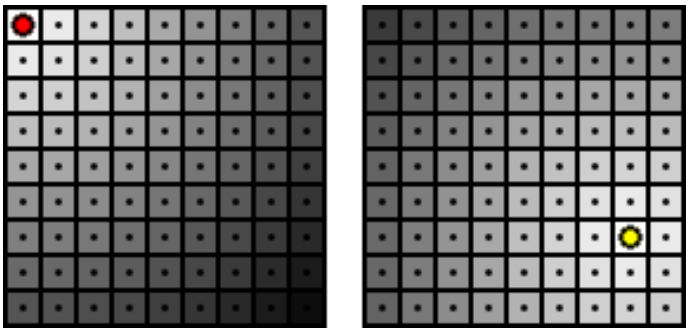

Figure-7

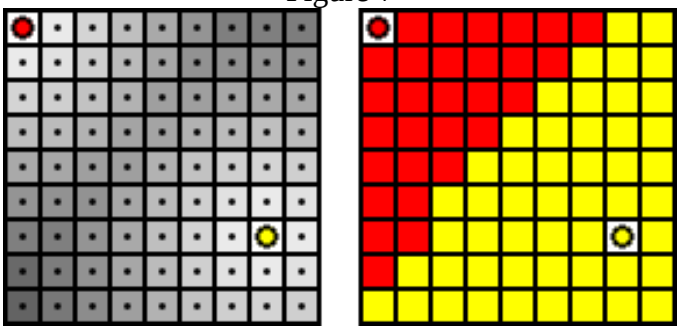

Figure-8

b. Proposed Approach

i. Cones Algorithm

One interesting observation regarding Ordinary Voronoi Diagrams came directly from Lejeune Dirichlet(1850) and Georgi Voronoi(1908). Both authors observed that a right circular cone is aligned with each of the point sites of the Voronoi Diagram at their apex, the minimum envelope of the intersections of these cones would correspond the Voronoi Diagram when projected to the plane. This observation is visualized in figures-9.

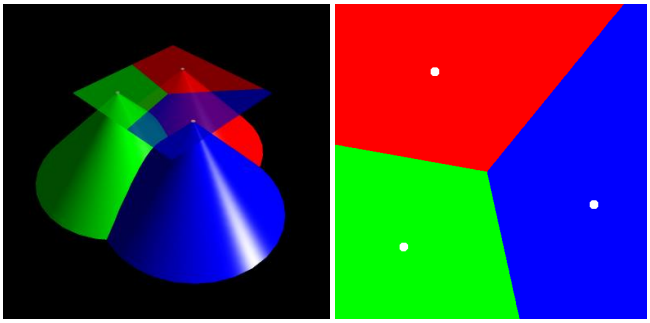

Figure-9: Dirichlet & Voronoi Observation

Hence the problem of generating an ordinary Voronoi diagram can be reduced to rasterizing right circular cones for each point site in the GPU. During the rasterization process Z-Buffer elimination is used to eliminate the invisible portions of the cones occluded by other cones. Each of the cones for different sites can be assigned a unique color id. As an end result the Z-Buffer when read back would store the distances towards the closest site at each pixel maximized at the Voronoi boundaries. Similarly the Color Buffer of the GPU would correspond to the actual Voronoi Diagram where each pixel is classified to the closest site with the help of the uniquely assigned colors. Figure-10 and figure-11 represent the end result (for both Z-Buffer and the Color-Buffer) for the example given in figure-9.

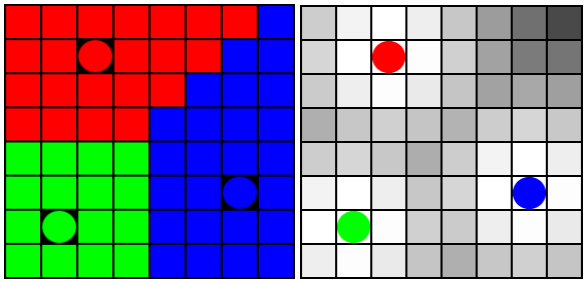

Figure-10: Color-Buffer(left) and Z-Buffer(right)

Authors ([9]) refer to the Cone function used for the point sites and functions used for higher order sites as distance functions. If we use the Brute-Force algorithm the Cones distance function for the point sites of the Ordinary Voronoi Diagram (OVD) should be evaluated at each sample point (or pixel) within the

scene. A better approach would be to reduce the number of samples and approximate the cone with a finite (and fewer) number of points while introducing some boundable error. Although per-pixel evaluation of the distance function is still done, this evaluation has now been passed to the GPU. Hence this approach greatly reduces the time to generate an OVD. Finer samplings of the Cone mesh would result in a better approximation and hence effectively reduce the error. Though, it is a trade-off because finer samplings will also cause an increase in the response time. The error bound is visualized in the Figure-11.

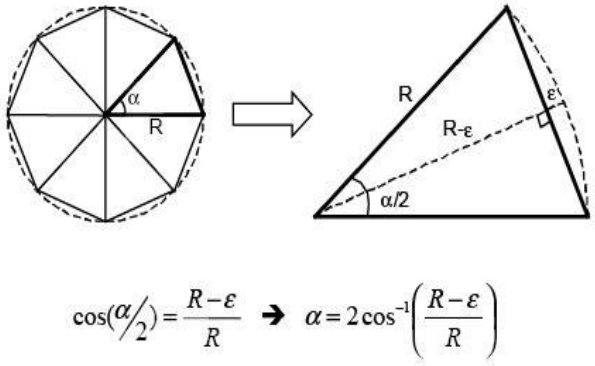

Figure-11: Bounding error for the cone mesh

The maximal error will be made at the perimeter of the base circle of the cone. Since we've used a finite approximation, the base of the cone meshes will correspond to triangles. As we go farther away from the center of the cone this error is increased. And it is maximized at the perimeter. In this case the error ε corresponds to the distance in between the radius of the cone's base circle and the length of the bisector of a particular triangle originated at the center of the cone and within the triangulation of the cone approximation. The polar angle of the cones can be bound with the equation given in Figure-11. In example, a 1024x1024 pixel grid would require cone approximations with no more than 85 triangles to make an error which is less than the width of a single pixel.

For an MxM grid the radius of the cones should be taken as M $\sqrt{2}$ so that each cone influences the entire grid and hence no samples on the grid are left open. This assumption is true for higher-order sites as well which are explained in detail in the upcoming sections.

ii. Higher Order Sites and GVD

Generalized Voronoi Diagrams with high-order sites such as lines, curves and polygons can as well be generated with the same approach. For higher-order sites, different distance functions (meshed) are used.

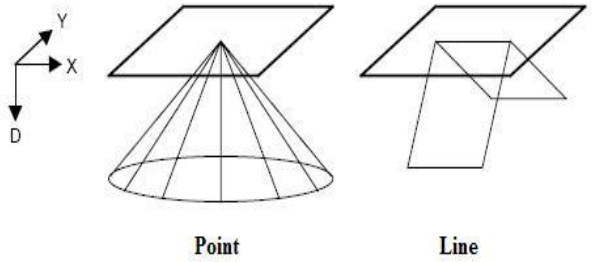

Figure-12: Distance Functions for point and line sites

For instance for higher-order sites such as a line segment we can use a Tent like structure which is composed of two plane segments, intersecting at the subject line segment. Both of these planes must make an angle of 45 degrees with the projection plane as the depth of the planes at some point would be same as the 2-d Euclidian distance from the line (on the projected plane). The tent distance function is exact and it does not introduce error. Cone meshes are again used for the end points of the line segment. An example tent mesh is given in figure-12.

The approach can easily be generalized for even higher order sites such as polygons. As each polygon is composed of vertices and edges we can simply use cone meshes for the vertices and tent meshes for the line segments. For the case of curves, we can tessellate the curves into a polyline and for each edge and vertex composing this polyline we can use the cone and tent distance mesh functions. Tesselation procedure as well, introduces some boundable error. Examples of higher order sites are given in figure-13

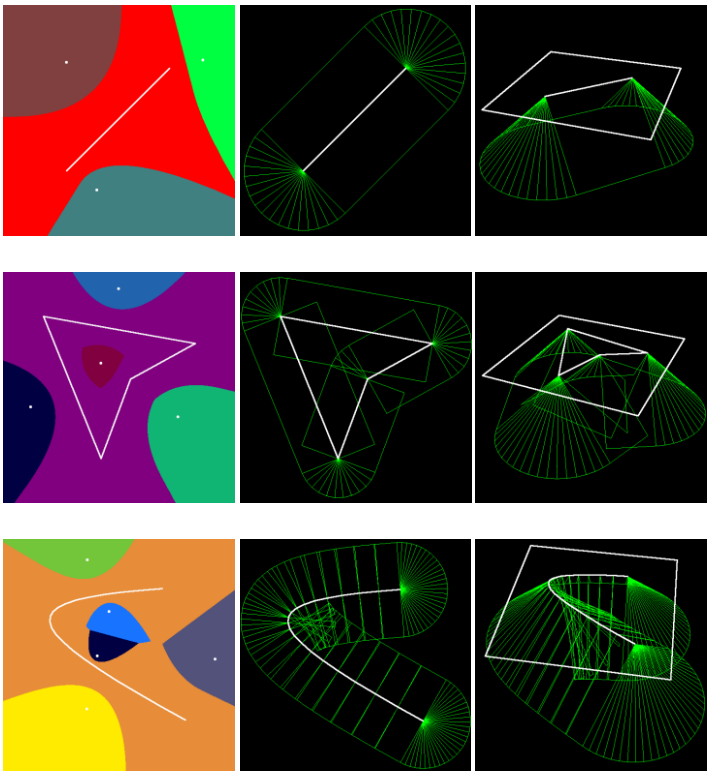

Figure-13: Distance functions for higher order sites

Line Segment (Top), Polygon (Middle), Curve (Bottom)

iii. Boundaries & Neighbors

The algorithm however generates only the Voronoi Diagram. We may need to explicitly identify the borders of the GVD to be used in other applications such as the generation of an approximate Voronoi Graph. Two algorithms for this purpose are identified in [9]. An informal version of the latter algorithm which is more efficient is as follows:

1. The boundaries of the acquired GVD are scanned until a proper seed point is found at where a difference in the color (site-ids) occurs.
2. Starting from this seed point we walk along the boundaries of the GVD by recursively iterating through the neighboring pixels where color differences occur.

Coarser approximations to the GVD would accelerate this process by reducing the pixels to be checked. However this would also introduce resolution error. Though since the boundaries are maximal due to the nature of the GVD. This resolution error can be omitted to some extend if it is certain that a site is not skipped in entirely due to error.

Neighbors of the sites can be found with the same algorithm where we keep an adjacency list of sites and fill it as we walk through the boundaries with respect to the algorithm given above. Each time we find a color difference, if it is not yet updated we decode the color to obtain the unique site-id and update the corresponding neighboring list of the site id in the adjacency list data structure.

iv. Sources of Error

Sources of error can be categorized into two:

1. Distance Error
    a. Meshing Error
    b. Tesselation Error
    c. Hardware Precision
2. Combinatorial Error
    a. Distance Error
    b. Pixel Resolution
    c. Z-Buffer Precision

- ***Distance Error***

Distance error is the error of miscalculating the distance of particular pixel to its closest site id. This error is affected by the three sub-types of error: meshing error, tessellation error, hardware precision error. Meshing

error is (as clarified in "7.b.i. Cones Algorithm" section) is the error occurred due the finite approximation of the distance functions. It can be bounded as stated before. Tesselation error (as mentioned in "7.b.ii. Higher Order Sites and GVD" section) is the error occurred due to tessellating a higher-order site such as a curve into a set of lower-order sites (lines and points). This error can again be bounded by increasing the amount of samples taken during the tessellation process. Hardware precision error is the floating point error common to all algorithms.

- ***Combinatorial Error***

Combinatorial error is the error of misclassifying a pixel with respect to its actual closest site. A pixel can be assigned wrong colors (or site-ids) due to three types of error. The first one is the Distance error as mentioned above. The second one is the major source of error which is the resolution error. The resolution error relates to the resolution of the pixel grid and can be reduced either by taking more samples (increasing the resolution of the grid) or adaptively zooming into the sections of the grid where an error is expected. Resolution error may cause Voronoi regions to be missed entirely. The final source of combinatorial error is the hardware precision of the Z-Buffer. It is already a standard for the GPU and is bounded.

8. Agent Generation and Restricting Workspace

In most scenarios, simulations of multi-agents are done in large-scale virtual environments (mostly virtual cities). Hence simulating the entire environment is extremely costly in terms of both storage and computational complexity. Therefore we need to restrict the area where we simulate the agents. This should be done in a reasonable way so we don't compromise the quality of the simulation. A possible approach to this problem is to simulate only the region around the current position of the camera. However, the region should be taken bigger than the camera's view-frustum so that we don't lose track of the agents when the camera rotates. In addition, in this specified region, agents should be simulated (but not drawn) even if they are not visible (occluded by obstacles, buildings).

If the range of the view-frustum can be taken sufficiently large, we could just simulate the rectangular region enclosing the view-frustum. Hopefully this would not cause agents to disappear before they get out of the observable range.

One another issue would be the way the agents are generated. A particular approach to this problem would be to randomly generate the agents at the free edge points of the rectangular area of focus. The agents can also be assigned random goals at the edges.

To improve efficiency in terms of computational time the free agents can be allocated in a pool structure (a stack for example). This would of course increase the memory requirements for large number of agents but it would not be a big issue. To generate agents we simply pop a free agent from the pool stack and initialize it at some random free edge point of the rectangular area of focus. During the initialization, a global goal at some random free edge point can be assigned for the particular agent. Similarly, any

agent that falls outside of this rectangular area at any time is pushed back to the pool stack. Figure-14 depicting agent generation is provided below:

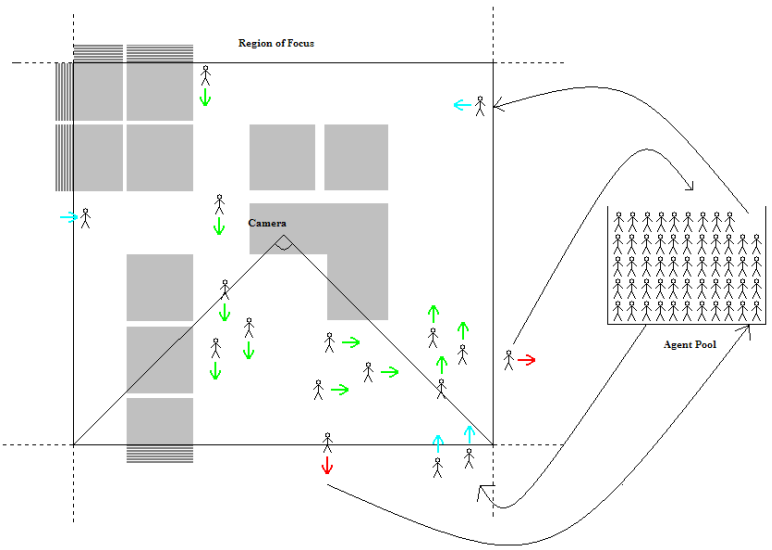

Figure-14 Agent Generation

Computation of the MaNG and other local force fields must also be done with respect to the region of focus. Namely at each frame of the simulation, obstacles, agents residing within the region of focus would be used for generating the voronoi diagrams. Since the region itself indexes a larger grid structure this phase is trivial.

9. Agent Interaction Routines and Local Decision Making

Local Force fields for short-term path planning and global (long-term) path planning with MaNG however modifies the agent behavior only in terms of navigation. In order to achieve more realism, agents must somehow be able to interact with their environment. To do so, I propose the notion of interaction routines. Interaction routines are generic routines of some sort that are uniquely defined and assigned to the agents. These routines take an instance of a close by, interact able entity, which can be another agent, a dynamic or a static object or it can even be an abstract object. After the execution of these routines, either the agent or the entity it is interacting with can be affected internally (for instance characteristic constants of the agent can be changed – in panic situations -). Or in another scenario, a fireman agent interacting with a fire entity may cause it to terminate the entity (by extinguishing it for instance).

Assuming that the interact able entity is considered as a site during Voronoi Diagram evaluation; it becomes efficient to find an entity to interact with. The MaNG data structure can be modified slightly to include the references to the actual entities in Voronoi Sites. Since the first order Voronoi diagram perfectly provides pair-wise proximity information we just need to check the neighboring sites to the subject agent's site in order to find a proper entity to interact with.

In terms of implementation, function pointers are used to refer to the interaction routines of the agents. In addition each agent can have more than a single interaction routine (a link list of pointers) to interact with different types of objects. Briefly, it can be said that interaction routines acts as a basic A.I. mechanism for the agents. It should however, be noted that complex interaction routines be a serious bottleneck in the system with respect to the numbers of the agents using this particular routine. So interaction routines should be made as simple as possible

10. Implementation Details

Implementation of the approach proposed by K.Hoff et al. [9] is done at the current stage of the project.

Implementation is done with C++ as the programming language using the Minimalist GNU (MinGW 4.3.2.) with gcc compilers on Windows environment. OpenGL 1.4 and GLut libraries are used to operate on the GPU. In addition a simple graphical user interface is added with the help of the GLui library.

Dev C++ 1.9.2 and Microsoft Visual C++ 5.0 Express Edition are both used as Integrated Development Environments (IDEs) throughout the project.

11. System Configuration

Implementation is done and results are obtained on a system having the following configuration:

Processor: Intel Centrino Duo T2500(2.0 Ghz) – 2Mb L2 Cache -
Ram: 2.0 Gb
GPU: Nvidia GeForce Go 7400
Harddisk: 100 Gb
Operating System: Windows XP with Service Pack 2

12. Results

Results are generated with different error thresholds(ε) and with different grid resolutions. In general increasing error threshold and reducing grid resolution substantially affects the response time of the system. In any case due to the maximal nature of the GVD boundaries large errors may be acceptable with respect to our content in order to not to sacrifice the interactive response times. In addition coarse resolution for the grid although increases error may be more beneficial both for the response time and the time that is required to generate the boundaries and the Generalized Voronoi Graph from the obtained diagram.

Several test results are provided in the below figures with different thresholds and resolutions. Response times for each test are also given. Tests are made for higher-order sites are also provided.

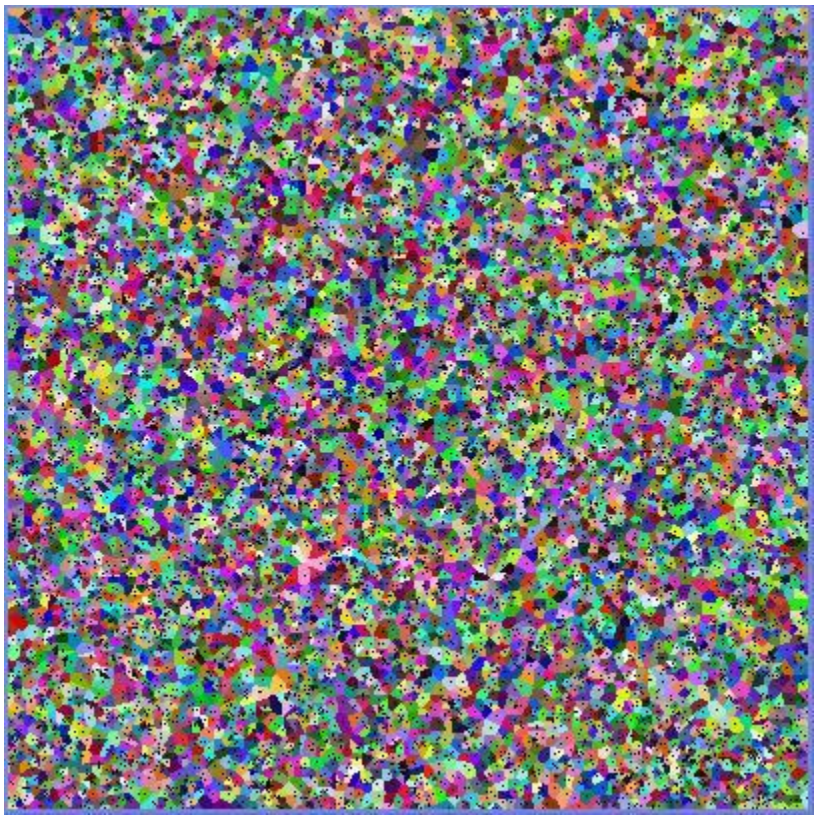

Figure-15: Ordinary Voronoi Diagram of 10000 randomized point sites
Error Threshold: 1
Resolution: 400x400
Response Time: 9.61 Seconds

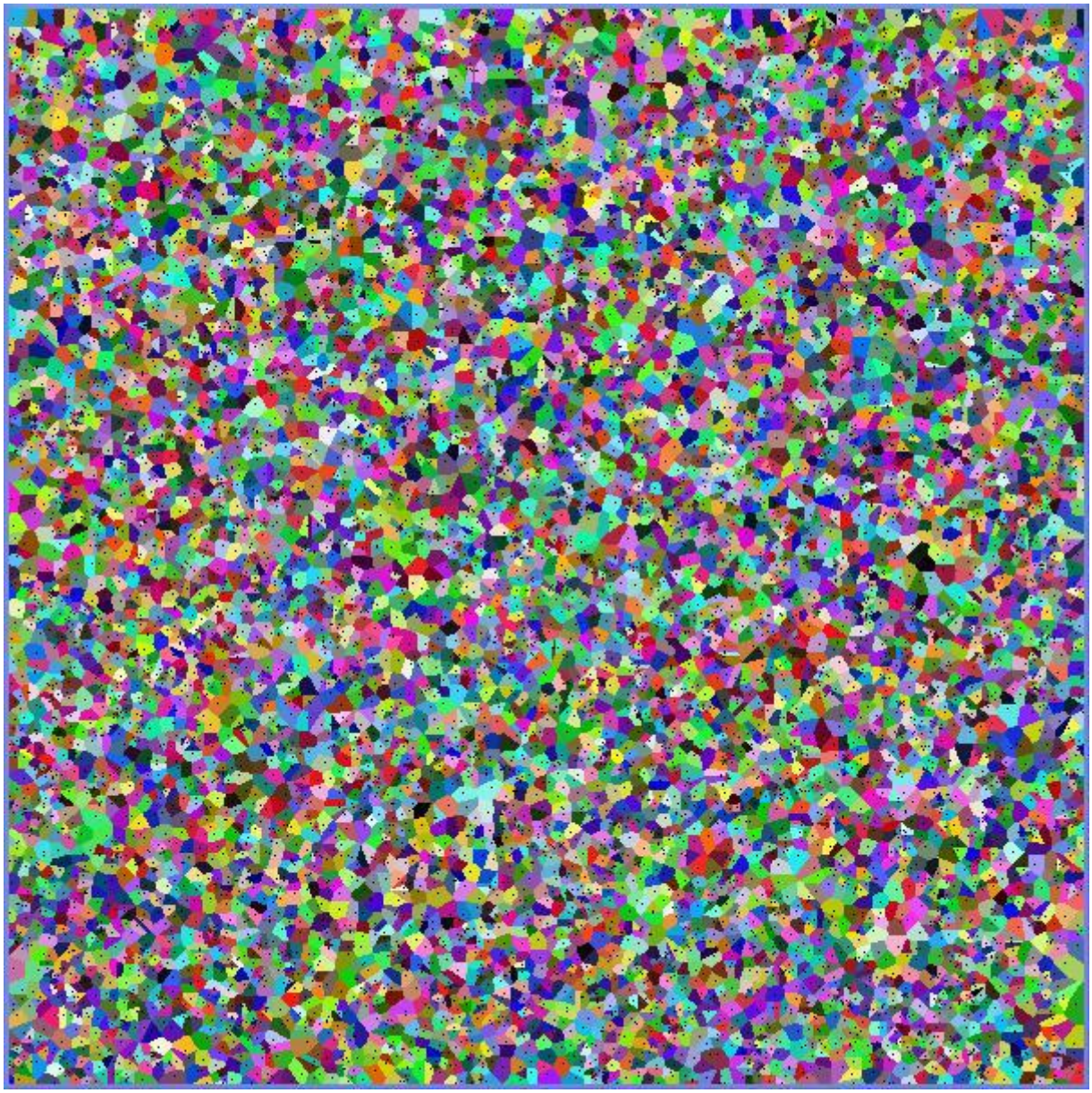

Figure-16: Ordinary Voronoi Diagram of 10000 randomized point sites
Error Threshold: 1
Resolution: 600x600
Response Time: 20.218 Seconds

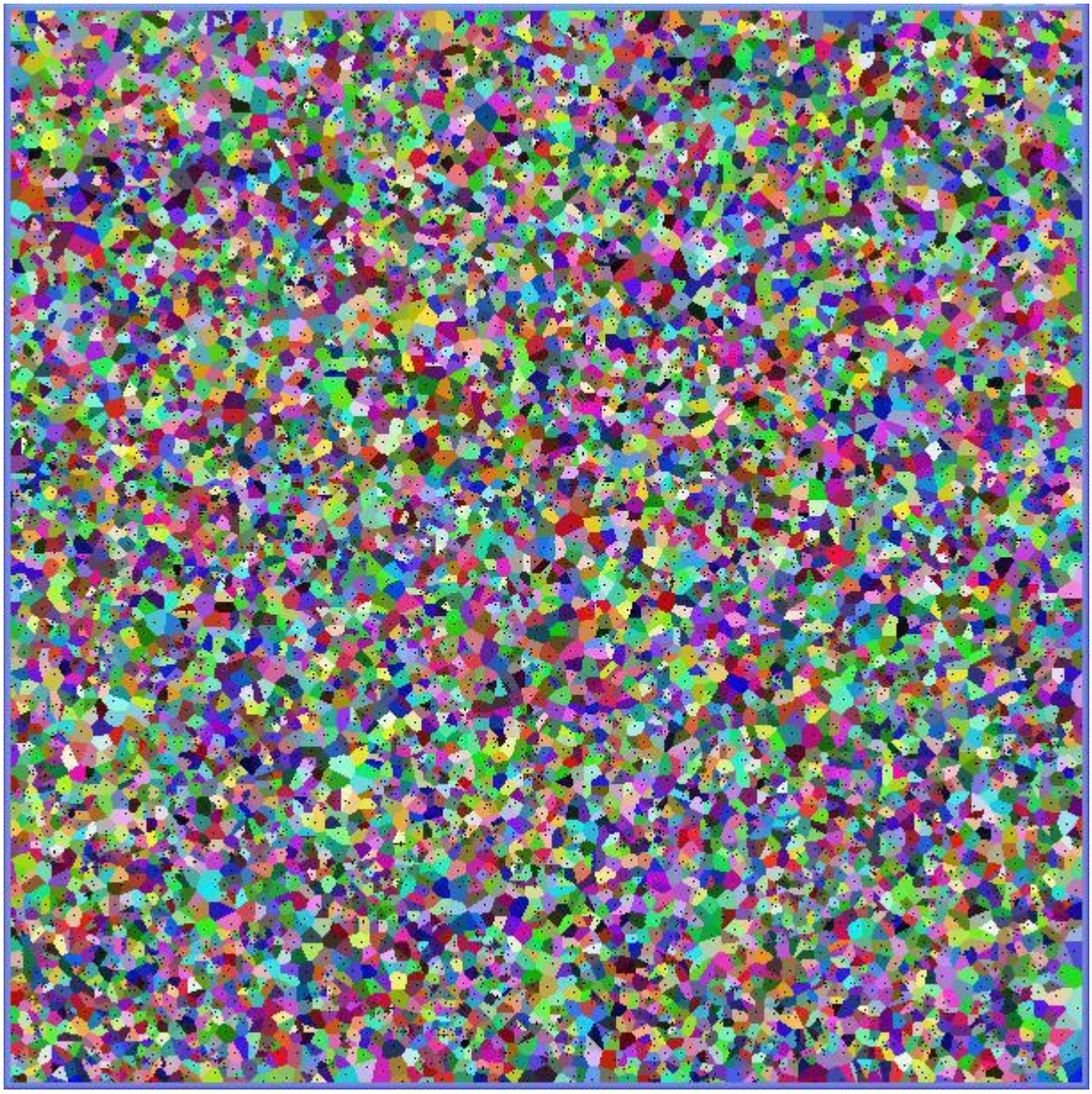

Figure-17: Ordinary Voronoi Diagram of 10000 randomized point sites
Error Threshold: 200
Resolution: 600x600
Response Time: 3.515 Seconds

Figure-18: Ordinary Voronoi Diagram of 800 randomized point sites
Error Threshold: 50
Resolution: 200x200
Response Time: 0.016 Seconds

Figure-19: Ordinary Voronoi Diagram of 100 randomized point sites
Error Threshold: 50
Resolution: 200x200
Response Time: 0.015 Seconds

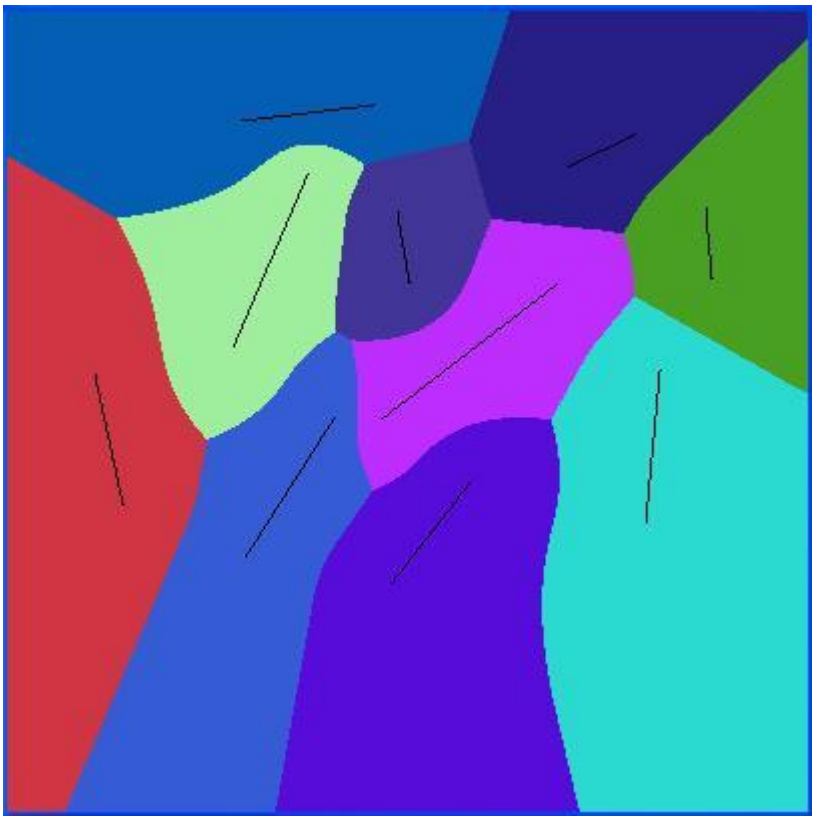

Figure-19: Generalized Voronoi Diagram of 10 line sites
Error Threshold: 1
Resolution: 400x400
Response Time: 0.047 Seconds

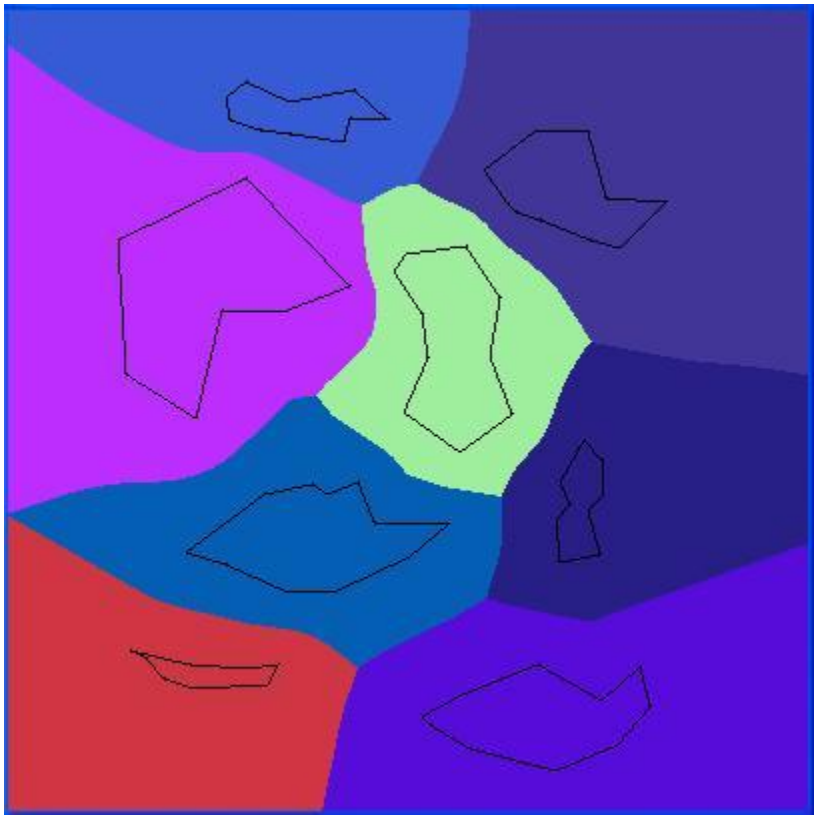

Figure-19: Generalized Voronoi Diagram of 8 polygon sites
Error Threshold: 1
Resolution: 400x400
Response Time: 0.125 Seconds

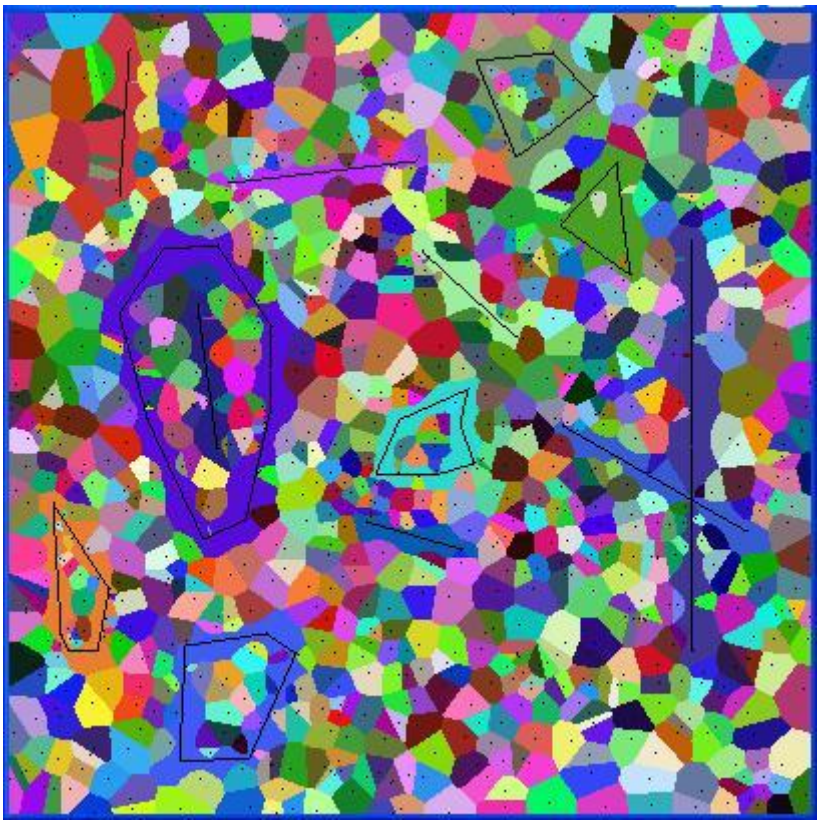

Figure-20: Generalized Voronoi Diagram of 6 polygon sites, 7 line sites and 1000 randomized point sites
Error Threshold: 1
Resolution: 400x400
Response Time: 1.063 Seconds

13. Conclusion

As mentioned in [8], the system has an overall complexity of $O(m^2 + n \log m)$ to generate the MaNG data structure and to compute the paths and behaviors of the agents within, at each frame (where m^2 is the resolution of the mxm grid and n is the number of agents). When compared with other recent approaches such as the approach proposed by Treuille et al.”Continuum Crowds” [2], it is seen that the approach presented in this report is a bit more efficient although they have similar time complexities.

First of all the approach of K.Hoff et al [9], solves the serious problem of proximity by effectively utilizing the graphics hardware. Such utilization is indeed a big improvement which lowers the work load of the CPU. In addition it is a known fact that recent GPUs outperform CPUs in terms of computational power.

Second, each agent takes a unique intermediate goal path on the second order Voronoi graph at each step of the simulation. Therefore it is not possible for several agents to have the same intermediate goal at any time instance. Unlike the approach presented in “Continuum Crowds” paper [2], agents will never get stuck in the local minima.

Finally, in this approach, agents are not grouped and hence each agent within the system can have a separate global goal without sacrificing the computational complexity.

14. References

[1]. Pelechano N., Allbeck JM. and Badler NI., Controlling Individual Agents in High Density Crowd Simulation. *Eurographics symposium on Computer animation, San Diego, California*, (2007), 99--108.

[2]. Treuille, A., Cooper, S. and Popivic, Z., Continuum Crowds. in *ACM Transactions on Graphics (SIGGRAPH 2006)*, (2006), 1160--1168.

[3]. Reynolds, C., Flocks, herds, and schools: A distributed behavior model. in *Proceedings of ACM SIGGRAPH*, (1987), 25-34.

[4]. Reynolds, C., Steering behaviors for autonomous characters. in *Game Developers Conference*, (1999), 763-782.

[5]. Chenney, S., Flow Tiles. in *ACM SIGGRAPH/ Eurographics Proceedings of Symposium on Computer Animation*, (2004), 233-242.

[6]. Kirchner, A., Namazi, A., Nishinari, K. and Schadschmeider, A., Role of Conflicts in the Floor Field Cellular Automaton Model for Pedestrian Dynamics. in *2nd International Conference on Pedestrians and Evacuation Dynamics*, (2003), 51-62.

[7]. Helbing, Buzna, Johansson and Wernrt Self-Organized Pedestrian Crowd Dynamics. *Transportation Science*, *39* (1), 2005. 1-24.

[8]. Avneesh, S. and Andersen, E. and Curtis, S. and Lin, M. and Manocha, D. *Real-time path planning for virtual agents in dynamic environments using Multi-Agent navigation graphs*. IEEE Virtual Reality, Charlotte, North Carolina, USA (2007)

[9]. K. Hoff, T. Culber, J. Keyser, M. Lin, and D. Manocha. *Fast computation of generalized voronoi diagrams using graphics hardware using graphics hardware*. *Proceedings of ACM SIGGRAPH 1999*, pages 277-286, 1999