DOI: 10.1111/cgf.12735

COMPUTER GRAPHICS *forum*
Volume 35 (2016), number 6 pp. 120–132

Coordinated Crowd Simulation With Topological Scene Analysis

Adam Barnett[1], Hubert P. H. Shum[2,*] and Taku Komura[1]

[1]School of Informatics, University of Edinburgh, Edinburgh, United Kingdom
A.Barnett-1@sms.ed.ac.uk, tkomura@inf.ed.ac.uk
[2]Faculty of Engineering and Environment, Northumbria University, Newcastle upon Tyne, United Kingdom
hubert.shum@northumbria.ac.uk

Abstract

This paper proposes a new algorithm to produce globally coordinated crowds in an environment with multiple paths and obstacles. Simple greedy crowd control methods easily lead to congestion at bottlenecks within scenes, as the characters do not cooperate with one another. In computer animation, this problem degrades crowd quality especially when ordered behaviour is needed, such as soldiers marching towards a castle. Similarly, in applications such as real-time strategy games, this often causes player frustration, as the crowd will not move as efficiently as it should. Also, planning of building would usually require visualization of ordered evacuation to maximize the flow. Planning such globally coordinated crowd movement is usually labour intensive. Here, we propose a simple solution that is easy to use and efficient in computation. First, we compute the harmonic field of the environment, taking into account the starting points, goals and obstacles. Based on the field, we represent the topology of the environment using a Reeb Graph, and calculate the maximum capacity for each path in the graph. With the harmonic field and the Reeb Graph, path planning of crowd can be performed using a lightweight algorithm, such that any blocking of one another's paths is minimized. Comparing to previous methods, our system can synthesize globally coordinated crowd with smooth and efficient movement. It also enables control of the crowd with high-level parameters such as the degree of cooperation and congestion. Finally, the method is scalable to thousands of characters with minimal impact to computation time. It is best applied in interactive crowd synthesis systems such as animation designs and real-time strategy games.

Keywords: motion planning

ACM CCS: I.3.7[Computer Graphics]: Three-Dimensional Graphics and Realism–Animation

1. Introduction

Synthesizing globally coordinated crowd behaviour is a challenging but highly demanded problem in computer animation, computer games and evacuation planning. It is popular nowadays in movies to include scenes of large crowd such as soldiers marching towards a castle. However, creating characters that move cooperatively is difficult due to congestion and crossing groups of characters. Similarly, it is a regular complaint from players of real-time strategy games that characters cannot move as quickly as they want due to blocking in the bottlenecks of the terrain. Finally, when designing buildings, it is beneficial to simulate and visualize how walls and doors affect the evacuation process. With existing methods, users often have to divide a crowd into smaller units and manually allocate them into different pathways to achieve the desired effect [KLLT08], which is inefficient even for skilled people. Ideally, one would wish to have an easy-to-use yet effective system to generate coordinated crowd.

Previous work in crowd simulation cannot efficiently create fluent crowd movement in complex environments. On one hand, while traditional agent-based path planning algorithms such as [PAB07] can implement individual character intelligence to avoid congestion, they are computational costly for large crowds since the cost depends on the total number of characters. On the other hand, field-based approaches such as [TCP06] are scalable to large crowd since the majority part of the computational cost depends only on the complexity of the environment. However, simply using a field cannot easily avoid character congestion in the bottlenecks of the scene.

*Corresponding author: Hubert P. H. Shum (hubert.shum@northumbria.ac.uk)

Once congestion forms, movement behaviours such as stopping and backtracking to find alternative routes will occur, which are not desirable in systems that require smooth crowd movement.

In this paper, we propose a field-based crowd control method that coordinates characters globally such that congestion, queueing and slowly moving characters are kept at a minimum. Based on the harmonic field [FS97] calculated from the environment, we represent the topology of the scene using a Reeb Graph [Ree46], which is a graph structure that defines possible pathways in the scene. In contrast to previous approaches, our method focuses on finding the capacity of the scene, that is, the maximum number of characters that can move through each path without causing congestion at any bottlenecks. With such information, the trajectories of the characters can be easily synthesized using a lightweight algorithm. As a result, our method can produce coordinated crowds with smooth crowd movement. It is scalable and can be applied to a crowd of large size.

Due to the use of Reeb Graphs to represent the capacity of the scene, our method can efficiently synthesize different crowd behaviours based on the degree of crowd cooperation and congestion. We define the cooperation in terms of how selflessly the crowd uses the paths found through the scene, and the congestion in terms of the targeted capacity of the scene. For example, a crowd of less intelligent characters like orcs in computer games might be less cooperative. Using the cooperation parameter, we can create a crowd that prefers shorter pathways to crowd efficiency, thereby causing blocking effects at bottlenecks. In addition, pathways in a city would usually have higher targeted capacity comparing to those of the same size in a village, which can be adjusted through the congestion parameter. These high-level control parameters provide an efficient interface for animators to alter the behaviour of the crowd radically with minimal manual overhead.

With our proposed method, it becomes possible to synthesize coordinated crowd behaviour under a field-based framework. We demonstrate coordinated crowd movements such as an army invading a castle and people evacuating in a complex environment with multiple start and goal points. We also show how we can adjust the degree of cooperation and congestion to synthesize different crowd behaviours. Being a field-based approach, our system is computationally efficient and is scalable to large crowds. It can simulate the movement of thousands of characters with minimal computational cost.

Preliminary results of this research can be found in [BCK13], in which we presented a basic system to calculate the Reeb Graph and evaluate the maximum flow of a scene. In this paper, we complete the algorithm with a detailed system design, a new character path planning mechanism, as well as high-level control parameters to synthesize different crowd behaviour. There are three major contributions in this research:

1. We propose a new algorithm to produce globally coordinated crowds based on a given environment, such that crowd congestion is minimized. We represent the topology of the scene using a Reeb Graph with estimated path capacity, and plan the characters movement using a lightweight algorithm. The method is scalable to large crowds in complex environments.
2. We propose a new path planning mechanism for characters on top of the framework in [BCK13]. In particular, we introduce the concept of logical route to avoid path crossing and enhance path quality, which is important especially when synthesizing crowd in a complicated scene with a large number of obstacles.
3. We design a set of new control parameters in our algorithm to synthesize a wide variety of crowd behaviours that cannot be achieved in [BCK13]. The cooperation value determines if characters will take longer paths to facilitate overall crowd efficiency, while the congestion value defines the path capacity that is acceptable to the characters.

2. Related Work

In this section, we review the related work in simulating large crowds of characters. We first discuss the computational cost problem of agent-based crowd synthesis. Then, we explain different type of field-based methods and evaluate their performance in avoiding congestion. We further review work related to topology analysis and point out how they can be used to solve the problem.

2.1. Agent-based crowd synthesis

Agent-based crowd synthesis methods control the behaviour of individual characters using pre-defined objective functions. Traditional flocking algorithms can model the movement of the characters using rules based on other characters in their immediate vicinity [Rey87a]. More dedicated simulation rules can be defined to simulate more realistic behaviour involving perception and cognition [ST05].

In order to avoid character collision, each character can consider the rest as dynamic objects and select a path that does not lead to collision [LK06]. Velocity of the characters is considered to enhance collision avoidance accuracy [PPD07]. The concept of composite agents is also proposed to model collision avoidance behaviour in bottleneck such as the doors of a train [YCP*08]. Coordinating multiple characters can be done by inter-character communication, such that the characters can exchange information and select a path that is not blocked [KG14]. The general problem of agent-based methods is the high computational cost for larger crowd, as the cost is proportional to the number of characters in the scene. Tuning objective function of individual character to achieve an overall crowd behaviour is also not trivial.

2.2. Field-based crowd synthesis

Field-based methods control the movement of characters using a scalar field over the space of a scene. Geometric distance fields are used to provide plans for crowds on arbitrary surfaces [TSO*10]. Although these plans are globally constructed, they do not take into account the positions and plans of other characters, and hence are still prone to congestion. Fluid dynamics is introduced for evacuation simulation, where the crowd is treated as a fluid [Hen74]. This approach is extended such that characters are given more intelligence to determine their movement based on their desired goal [TCP06]. The characters follow the field with a view of avoiding areas of high density, which allows them to plan around congestion. However, it does not prevent congestion from occurring in the first

place, especially when many characters all decide to use a currently empty bottleneck at once.

Harmonic fields are widely applied in robotics [FS97, RI05], which define the desired flow of the characters in the scene. They have also been used in computer graphics for deformation transfer [ZRKS05], remeshing [DKG05] and morphing [KSK97]. For crowd simulation, it is possible to provide simple global guidance for characters by setting the potential at the edges of obstacles to be 1 and at the goal to be 0, such that the characters can descend the gradient to reach the goal [SFF*10]. We also use harmonic field in this research, but we integrate flow control information by analysing the topology of the scene.

2.3. Crowd control interfaces

It is possible to implement user interfaces for controlling the crowd manually and thereby avoid collisions. Field-based control approaches allow the user to manipulate the crowd around obstacles using a control field [PvdBC*11, Par10]. By using a mesh to model a crowd, the user can define the crowd movement and formation with simple gestures [HSK12, HSK14]. Different editing tools and interfaces are proposed to place and manipulate the path of a crowd [KLLT08, KSKL14, JPCC14]. While these methods are capable of producing scenes comparable to ours, the user has to bear the responsibility to design the crowd movement such that congestion is minimized. The manual labour becomes more intensive in scene of complex topology and large number of characters. We prefer a method that is automatic, scalable and easily controllable.

2.4. Topology analysis

Topology-based methods represent the scene using simpler structure for more efficient planning. There are a number of topology represent suitable for scene representation. The navigation mesh breaks the scene up into different navigable segments in a mesh [Sno00]. The probabilistic roadmap explores the scene through connecting a series of randomly generated points [KSLO96]. The Voronoi diagram constructs a graph that has the maximum clearance from all obstacles [GO07]. The Reeb Graph provides a topology that originates and terminates at the start and goal points while defining the valid topological routes between them [Ree46]. We use the Reeb Graph as its representation suits our problem of crowd navigation, and can easily be integrated with the harmonic field.

The density-based method in [vTCG12] and the capacity-based method in [PCM*06] also represent the topology of the scene as a series of edges and nodes. However, both approaches only take the current congestion into account. Unlike our system, they do not have the framework to avoid the occurrence of congestion and bottlenecks as a pre-process. In [VDAGHP10, KGvdS13], medial axis is used to represent the scene, and paths planning of the characters is considered as a dynamic flow problem. Similar to our approach, these methods aim at minimizing the travelling time of the characters. However, since they do not focus on continual motion within the scene, they offer no solid solution to the problem of crossing groups of characters. Under our framework, this problem is solved using a lightweight path planner.

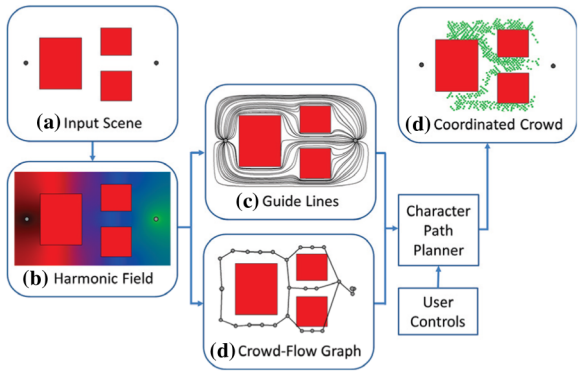

Figure 1: *The overview of our framework.*

3. Methodology Overview

The overview of our method is shown in Figure 1. (a) The input data are a scene consisting of a number of obstacles, an available space, as well as a set of start and end points. (b) We first compute a harmonic field over the available space. Using the field, we compute (c) a set of paths called *guide lines* through the space and (d) the *crowd-flow graph* that defines the topology of the scene. With the optional user controls, we then plan the path of the characters to synthesize the coordinated crowd. (e) Finally, the characters follow their assigned paths using a local controller for collision avoidance.

4. Topological Scene Analysis

In this section, we describe how the scene is analysed to provide a topological representation of the paths. We first describe how the harmonic fields are computed. Then, we explain how the field is used to construct the topological representation of the scene.

4.1. Harmonic fields

The harmonic scalar field is central to our method. It is used to compute the topology of the environment, the capacity of the open area and the trajectories of the characters. Harmonic field is ideal for our purpose because it provides a smooth gradient from the start points to the end points without any local extrema.

We follow [DKG05] to compute the harmonic fields over the environment. First, the available area is filled with a grid of h_{total} vertices in the resolution of h_{row} by h_{col}, where $h_{total} = h_{row} \times h_{col}$. In our experiments, h_{row} and h_{col} are set to 120 and 200, respectively. These values are chosen to provide a grid with a fine enough resolution such that all of the obstacles are adequately represented. The objective here is to compute a scalar field $\mathbf{h}$ that varies continuously from the value 0 at the start points to 1 at the goal points. We represent $\mathbf{h}$ by a column vector of the values at all h_{total} vertices in the grid, and solve following linear equation:

$$\mathbf{Ah} = \mathbf{b}, \tag{1}$$

where $\mathbf{A}$ is a h_{total} by h_{total} matrix of connections, in which the value of the entry $A_{i,j}$ is 1 when the vertex i and j are connected

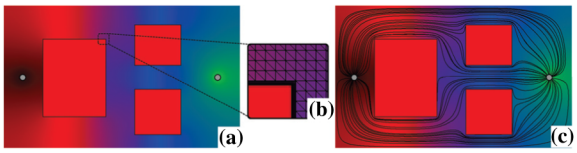

Figure 2: *(a) An example of the harmonic field generated for a scene, (b) zoomed-in view showing the right-angled triangles sampled in the space and (c) the guide lines computed over that harmonic field. The value of the harmonic field varies from 0 to 1, which is visualized by the varying colour from red to green.*

in an open area, 0 when they are not connected, and 1 if $i = j$. The column vector $\mathbf{b}$ has a size of h_{total} and is used to store the constrained values of the vertices, with 0 for the start points and 1 for the goals. The equation is a sparse linear problems and can be solved with sparse linear solvers to produce a smooth scalar field that has no local extrema. We use the solver CSparse [Dav06] in our implementation. An example of a harmonic field computed from an environment is shown in Figure 2(a).

4.2. Guide lines

Once the harmonic field is created, a set of *guide lines* are built, which is a series of paths following the gradient of the harmonic field from a start point through to a goal point. They provide smooth paths for the characters to follow through the space and can be used by the system to determine which direction characters should move at any given point within the space.

To calculate the guide lines, first, we first apply triangulation on the vertices of the harmonic field, and create a set of right angled triangles, which is illustrated in Figure 2(b) as a zoomed-in view. Then, we solve the following equation for each of them:

$$\begin{bmatrix} \mathbf{v_{t1}} - \mathbf{v_{t2}} \\ \mathbf{v}_{t3} - \mathbf{v}_{t1} \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix} = \begin{bmatrix} h_{t1} - h_{t2} \\ h_{t3} - h_{t1} \end{bmatrix}, \tag{2}$$

where $t1$, $t2$ and $t3$ are the three vertices forming the triangle, $\mathbf{v}_{t1}$ is the vector position of the $t1$ vertex, h_{t1} is the harmonic field value at the $t1$ vertex. $\mathbf{v}_{t2}$, $\mathbf{v}_{t3}$, h_{t2} and h_{t3} are defined accordingly. The equation can then be solved analytically for g_0 and g_1, which are the gradient of the harmonic field within the triangle in the horizontal and vertical direction, respectively.

Once the gradient is found, the guide lines are generated by sampling g_{total} seed points that are spaced evenly throughout the entire scene. From each seed point, we extend the guide line by connecting the next vertex in the increasing gradient direction until the goal is reached. The same process is performed in the decreasing gradient direction to reach the starting point. As a result, a guide line connecting a series of vertices is produced. In our implementation, g_{total} is set as either 100 or 1600, depending on the quality and computational speed requirements. An example set of guide lines is shown in Figure 2(c).

The guide lines and harmonic field have similarities to the streamlines and tensor fields commonly used for polygon meshing such as [ACSD*03], respectively. The harmonic field is chosen as a representation because it has the intrinsic property of containing no local extrema, whereas tensor fields may do in the form of umbilical points. Having no local extrema in the field is important in our problem, because it means that any path following its gradient is guaranteed to terminate at one of the end points in the scene.

4.3. Crowd-flow graph

The *crowd-flow graph* uses the harmonic field to provide a succinct representation of the available space in the scene such that the motion of the entire crowd can be planned and coordinated at once. It defines the topology of the environment based on the Reeb Graph structure. It also provides the maximal number of people who can pass through each Reeb edge. With the information, we compute the maximum flow of the scene, which indicates how many characters should move to a location to fill the scene.

4.3.1. *Creating the Reeb Graph*

To build the base structure of the crowd-flow graph, we first compute a Reeb Graph [Ree46] based on the given environment. Each node of the Reeb Graph corresponds to an area in the environment, and each edge corresponds to a connection between two adjacent areas. The Reeb Graph suits our problem because it provides a topology that originates and terminates at the start points and goal points while defining the valid topological routes between them.

We follow the process of [HSKK01] to compute the Reeb Graph with a newly proposed Morse function. Based on the concept of Morse theory [SKK91], a Morse function is a smooth real-valued function with no degenerate critical points, and is used to represent a manifold so as to study its topology. Unlike [HSKK01] in which geodesic distance is used as the Morse function, we use harmonic scalar field instead. This is because the scalar values on the harmonic field change smoothly depending on the start points, the goals and the geometry of the environments. Even if there are multiple start and goal points, smooth non-intersecting paths through the space can be obtained.

More precisely, we first decide the number of levels of harmonic value, R_{total} , which essentially represents the resolution of the Reeb graph. We generate $R_{total} + 1$ boundary values for the harmonic field, which are $\frac{r}{R_{total}}$ with integer values of $r \in [0, R_{total}]$. Vertices of the harmonic field are then grouped using the boundary values as the dividers. The connectivity of the vertices within each group are examined, as vertices in the same group may not be connected spatially due to obstacles. The average position of each disconnected part of vertices is calculated as a Reeb node. Two Reeb nodes are connected and form a Reeb edge if (1) the two corresponding sets of vertices are connected and (2) the corresponding harmonic field values are in adjacent groups. In our experiments, we set R_{total} to either 32 or 64 depending on the required simulation quality and the complexity of the obstacles. Figure 3 shows two examples of Reeb Graphs, with different colours representing the different vertex groups.

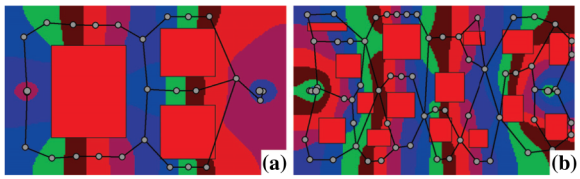

Figure 3: *Examples of the Reeb Graph generated (a) on a simple scene with a Reeb resolution of 16 and (b) on a more complex scene with a Reeb resolution of 32.*

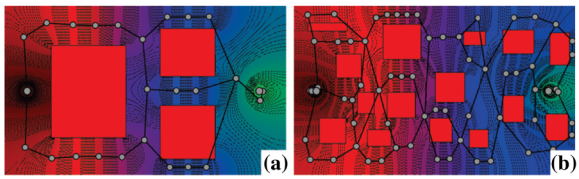

Figure 4: *Examples of the iso-flowlines (shown as dotted lines) in two scenes.*

4.3.2. *Computing the capacity of the graph*

In order to compute the max-flow on the Reeb Graph, we need to know the number of characters that can fit through the open space associated with each Reeb edge. With the capacity of each edge, we can evaluate the number of characters that can fit through the entire scene, as well as the way to fill it.

We compute the capacity of a Reeb edge by finding the shortest *iso-flowline*, which is a width-line running perpendicular to the natural gradient of the harmonic field, within the area covered by the Reeb edge. Figure 4 shows two examples of the iso-flowlines in the environment. Assuming h_{min} and h_{max} are the centre harmonic values of the two Reeb nodes connected to a Reeb edge, we can compute iso-flowlines for sampled harmonic values between them:

$$h_\alpha = (1-\alpha)h_{min} + \alpha h_{max}, \tag{3}$$

where $0 \leq \alpha \leq 1$ and α is a series of i_{total} evenly quantized values sampled across the range. For each h_α, we connect the vertices that have the harmonic value just above h_α (i.e. at least one neighbour vertex has a harmonic value below h_α) and form one iso-flowline. In our system, i_{total} is set as either 1 or 8 depending on the quality and computational speed requirements.

The capacity of each Reeb edge is determined by the length of the shortest associated iso-flowline. This is because it represents the width of the space perpendicular to the gradient of the harmonic field, which is the expected direction of flow of the characters. This process is then repeated for each edge of the Reeb Graph.

4.3.3. *Computing max-flow of the graph*

Once the crowd-flow graph of the environment is obtained, we can evaluate how to maximize the flow from start to goal with the capacity of the Reeb edges. While the max-flow concept has been widely used for the purposes of evaluating buildings through evacuation simulation [CFS82, KF85, CJ03], it is less investigated in the area of congestion minimization. Here, we explain the problem of applying max-flow in computer graphics, and explain our strategy to solve it.

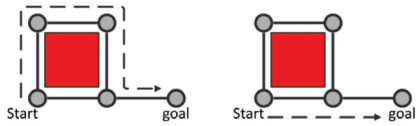

Figure 5: *The problem of Max-flow solutions. Both images represent valid solutions to fill this graph to the Max-flow, but the right one uses more efficient routes.*

Although max-flow algorithms can maximize the number of people moving from the start to the goal, we have found that some of the solutions produced may contain inefficient routes that are unnecessary long. This is because max-flow is an under-constrained problem. While there are many correct solutions, some of them involve convoluted and awkward routes. To illustrate, consider the example in Figure 5. In the left image, the graph has been filled to capacity. However, in some max-flow systems, the longer, less efficient upper route may been filled, instead of the shorter route below, as shown in the right image.

To solve this problem, we define a customized two-step strategy when computing the max-flow of the crowd-flow graph. Here, we define a route as a series of Reeb edges from the start point to the goal, and filling a route means assigning the capacity of the route as part of the max-flow solution. The maximum capacity of a route is constrained by the Reeb edge with the minimum capacity in the series. The objective here is to fill all the routes favouring the shorter ones.

In the first step, we fill the crowd-flow graph using the guide lines computed in Section 4.2 as a heuristic. The guide lines represent a subset of routes in the crowd-flow graph, and each guide line corresponds to a route represented by a series of Reeb edges. Starting from the shortest guide line, we fill the corresponding route to its maximum capacity. We then repeat the process for the next shortest guide line until every guide line is examined. By this stage, the crowd-flow graph is partly filled, as only those routes corresponding with guide lines are filled.

In the second step, we apply traditional max-flow algorithm on the remaining capacity of the crowd-flow graph. We use the method proposed in [BK04] as it is computationally efficient. It involves growing two search trees from both the starting point and the goal until they meet and form a route. Such a route is considered to be the shortest route and is filled. The tree is further grown until another route is formed. This process is repeated until all routes are filled. In our system, since all routes pass through the same number of Reeb nodes, they have the same length. As a result, expanding the tree is computationally efficient because we only need to build it once. Even for a scene with over 100 obstacles, it takes less than 5 ms to complete the calculation. By this stage, we have a complete max-flow solution for the crowd-flow graph.

Since in our customized strategy we fill the graph using the guide lines first, we essentially force the algorithm to use the set of more

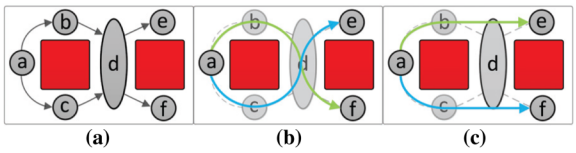

Figure 6: *(a) The Reeb graph of a two-obstacle scene, with one start point **a** and two end points **e** and **f**. (b) A sub-optimal solution of logical route that results in route crossing. (c) A preferable solution that can prevent congestion.*

elegant routes given by the guide lines. As a result, the computed max-flow of the crowd-flow graph may not be the optimal one. In practice, however, this does not heavily effect the max-flow value discovered. The reason is that the flow in the crowd-flow graph follows the same direction as the guide lines in general, since they are both created from the same harmonic field. In the experiments we have conducted, the customized strategy can maintain more than 99% of the original flow values, while shorting 2% of the paths that are of poor quality to avoid visual artifacts.

5. Character Path Planning

In this section, we describe how the topological information computed is converted into paths through the space. First, we explained how logical routes are created to avoid characters obstructing each other. Then, we explain how they are converted into geometric paths for characters to follow.

5.1. Logical route creation

The maximum flow computed in Section 4.3.3 determines how many characters can move along each Reeb edge, but it contains no information about what routes they should take. In this section, we describe our solution for creating these routes. Here, a logical route is defined as an ordered list of Reeb nodes from the start to the end of the Reeb Graph. Each logical route also has an associated capacity value describing the number of characters that can fit along it.

The problem of creating logical routes is that without a strategy, characters may end up crossing one another's routes. This can cause character collision and crowd congestion, which defeats our purpose of synthesizing fluid crowd movement. An example demonstrating this problem is shown in Figure 6. Image (a) shows the maximum flow in which the characters enter node **d** from nodes **b** and **c**, and they leave at nodes **e** and **f**. While we know that the number of characters entering and leaving node **d** is the same, we do not know the paths the characters take. Image (b) demonstrates one solution of logical routes, in which two groups of character cross paths and would potentially obstruct each other. Image (c) demonstrates a preferable solution of logical routes that fills the scene while avoiding route crossing.

To obtain a smoothly flowing solution for the crowd that minimizes route crossing, we propose a lightweight algorithm that is based on the logical classifications of the Reeb nodes. As shown in

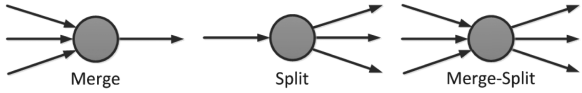

Figure 7: *The three logical classes of path connection for Reeb nodes.*

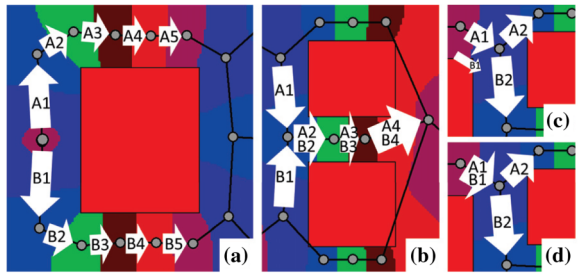

Figure 8: *The process of logical route creation.*

Figure 7, we consider three logical classes, including *merge* (more than one edge joining at a node), *split* (more than one edge emerging from a node) and *merge-split* (more than one edge joining and emerging). The nodes having exactly one incoming and one outgoing Reeb edge do not cause the mentioned route-crossing problem, and are not considered.

The first step of our algorithm is to identify the logical class for all the Reeb nodes in the scene. We take advantage on our harmonic field generated Reeb Graph, within which the gradient of every Reeb node is well defined. For each node, we examine the gradient of the neighbour nodes and evaluate if the corresponding Reeb edges are incoming or outgoing. Then, based on the number of incoming and outgoing edges, we can identify the merge, split and merge-split nodes.

The second step is to create logical routes with no path-crossing. Again, due to the use of harmonic field in the Reeb Graph, this can be easily achieved by stepping through all the Reeb edges from the starting points to the goals with increasing level of gradient. Without the path connection nodes as shown in Figure 7, the process is as simple as repeating the stepping and recording the Reeb nodes visited until the goal nodes are reached. The capacity of each logical route is constrained by the Reeb edge with the minimum capacity in the route. Figure 8(a) shows an example of the initial stage to create two logical routes, with the capital characters identifying the routes and the numerical values indicating the step number.

The tricky part of the algorithm is to deal with path connection nodes to minimize route crossing. During stepping, whenever a merge node is encountered, we order the routes based on their polar coordinates, and step all logical routes together thereafter. Figure 8(b) shows an example route ordering and stepping at a merge node. When a spilt node is encountered, the route of the incoming edge is cloned with its capacity split, such that there are multiple routes for future stepping each outgoing edge. Figure 8(c) shows how route A1 is cloned as A1 and B1, such that stepping can be continued at A2 and B2 separately. In the situation where the incoming edge contains multiple routes that are grouped at merge

nodes encountered before, we assign the routes to the outgoing edges based on their polar ordering, as shown in Figure 8(d). If the capacities of the multiple incoming routes does not map exactly to that of the outgoing edges, we clone the incoming routes and split their capacities. Finally, for the merge-split node, we perform the processes for both merge node and split node.

The result of the process is a set of logical routes. Each of them consists of a set of Reeb node to be visited from the start node to the goal node, as well as a capacity value indicating the number of characters that can fit into the route. Route crossing is minimized as we assign incoming and outgoing logical route based on a specific order. This allows us to eliminate obstruction between different groups of character in the scene during crowd synthesis.

5.2. Geometric paths creation

Here, we describe how the logical routes created are converted into geometric paths in the environment. We define a geometric path as the path taken by a set of characters during the synthesis.

The logical path only indicates the set of Reeb nodes to be visited but not the geometric position of the trajectories. In order to create a geometric path, we combine the information given by the guide lines, as they provide smooth paths from the start points to the goal points. For each logical route, the geometric paths are created as the set of guide lines that follows the route. In the case that no single guide line follows the entire logical route, we use segmented parts that follow different sections of the route and stitched their ends together. Since both guide lines and logical paths are created fundamentally from the same harmonic field, a solution of geometric paths can always be found.

Once we find the solution of all geometric paths, we assign characters to each of them based on the capacity of the paths. We assign characters to the shortest paths first, fill its capacity, and repeat with the next shortest path until all characters are assigned.

5.3. Run-time character control

Here, we explain how to synthesize the character movement during run-time, as well as the offline process for full body character rendering.

A naive approach to synthesize character movement is to control the kinematics of each of them individually, as well as apply collision detection and reaction. However, to reduce computational cost, instead of controlling individual characters, we use flocking [Rey87b] to control each group of characters that follows the same geometric path. Flocking is efficient at keeping the characters spreading out, avoiding collisions and adding randomness. Obviously, depending on system requirement, more advanced local controllers can be used. We selected flocking as we wish to provide an unvarnished insight into the quality of geometric paths we created.

Rendering of full body characters is performed as an offline process. Based on the synthesized locations of the characters, we plan their full body motion using the method from [SKY08] with a captured running motion database. While the system can plan collision free trajectories for each character, partial collisions of body parts such as arms and legs among characters may occur. We use the physical simulation system proposed in [SH12] to model the movement of the characters with joint torque and joint force. Under such a physical environment, body parts penetration among characters can be minimized.

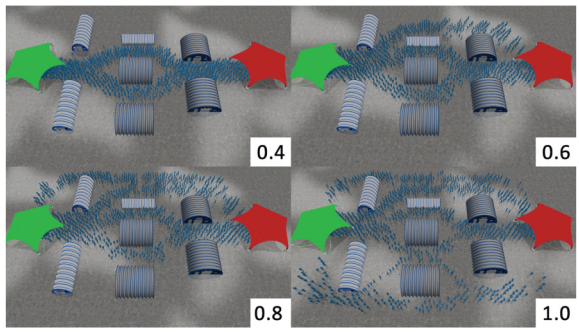

Figure 9: *The same scene with different $\theta_{cooperation}$ values.*

6. User Controls

Here, we explain how user controls are used to alter the level of cooperation and congestion in the crowd. By allowing such high-level controls, we provide users with the ability to alter the crowd behaviour efficiently without tweaking a large set of parameters or adjusting the trajectories of individual character.

6.1. Cooperation control

The cooperation value defines how characters cooperate with each other to maximize the flow of the overall crowd. When the system runs normally, the crowd is the most cooperative. As the value decreases, characters tend to take shorter paths even if that means creating congestion and reducing the efficiency of the crowd.

In the normal run state of our system, the cooperation value is set as $\theta_{cooperation} = 1.0$, which represents that the characters are fully cooperating. For $\theta_{cooperation}$ values of less than 1.0, some characters would not follow the assigned geodesic path:

$$\theta_{cooperation} = \frac{Character\ Following\ Assigned\ Path}{Total\ Number\ of\ Character}. \quad (4)$$

In our implementation, based on the given $\theta_{cooperation}$, we select the required number of characters assigned with the longest geodesic paths, remove their originally assigned path and assign them with a randomly selected, shorter geodesic path. As a result, some of the routes may not be used.

An example of adjusting $\theta_{cooperation}$ is shown in Figure 9. Here, it is clear how reducing the value of $\theta_{cooperation}$ affects the degree of overall crowd coordination. It shows how lower values divert characters from the longer paths to shorter ones, which are more immediately beneficial to them, but detrimental to the crowd as a whole with significant congestion appearing at the shortest paths. One example of applying the cooperation is to simulate less intel-

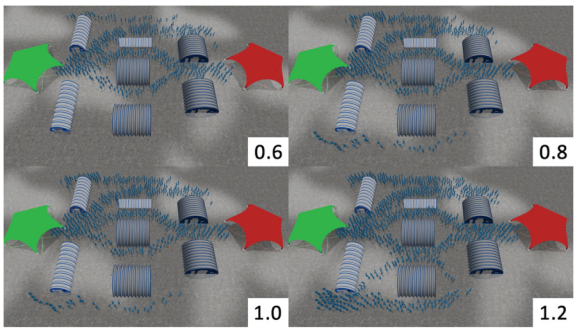

Figure 10: *The same scene with different $\theta_{congestion}$ values.*

ligent characters, such as orcs in a real-time strategy game, which do not consider the efficiency of the whole crowd but individual benefit.

6.2. Congestion control

The congestion value adjusts the capacity values for each geodesic path. As the physical capacity of the path is fixed according to the available space, adjusting the congestion value would result in the system assigning characters to a path above or below the original capacity. This creates an overall crowd behaviour of preferring under-utilized or over-congested pathways.

The scene congestion is adjusted by:

$$R_{CNew} = \theta_{congestion} * R_C, \tag{5}$$

where R_C and R_{CNew} are the capacities of the geodesic paths in the scene before and after the change. In our implementation, the value for $\theta_{congestion}$ can be altered down to a minimum of 0.0 and up to a maximum of 2.0. Mathematically, a value of $\theta_{congestion} = 2.0$ means that the scene is filled to two times its capacity, that is two times more than the maximum flow found for the scene. Once the value is changed, the system will assign the geodesic paths to the characters with the new capacity.

The result of this process is shown in Figure 10. Here, it is clear how lower values of $\theta_{congestion}$ lead to an emptier scene while values above 1.0 lead to extreme crowding and even congestion at the bottlenecks. The congestion value can be used to simulate characters' perception of the path capacity. For example, in a city, character can tolerate more congested paths. Whereas in a village, characters may prefer sparser paths.

7. Experimental Results

In this section, we present experimental results of our system. We first demonstrate crowd synthesis in different scenarios. Then, we compare our system with the other methods [TCP06] and point out our strengths. Finally, we discuss how to trade-off quality and computational cost in our system, such that the system becomes fast enough to be used in computer games. All experiments were performed on a laptop computer using one core of an Intel Core i7-3840QM CPU and 16 GB of RAM. Synthesis frame rate was set at 30 frames per second.

7.1. Scenarios simulation

We synthesized four scenarios to demonstrate our system as shown in Figure 11. We set $\theta_{cooperation} = 1.0$, $\theta_{congestion} = 1.0$, $h_{row} = 120$, $h_{col} = 200$, $g_{total} = 1600$ and $i_{total} = 8$ to synthesize the crowd. The number of characters, number of obstacles, Reeb resolution R_{total}, computation time in milliseconds for our simulated scenario are shown in Table 1. The computation time given here accounts for the process from analysing the scene to generating and assigning full paths for all characters. The cost for run-time simulation of character movement depends only on the flocking algorithm described in Section 5.3, which is always real-time even for the largest scene we simulated with thousands of characters.

The **monster escape** scenario portrays a popular movie and animation scene, where characters run away from disaster in an environment filled with obstacles. With our method, we ensure the high dynamic flow of the crowd as congestion is minimized.

The **castle invade** scenario is another popular scene in both animation and games, where an army moves towards a destination in a coordinated manner. Notice that the computational cost depends on the complexity of the environment rather than the number of characters. Therefore, even for simulating thousands of characters, the one-off scene analysis cost is only 2.5 s.

The ;**evacuation** scenario is built based on the floor plan of the Informatics Forum in the University of Edinburgh. We use a larger Reeb resolution to cover the fine details of the building. The green tents indicate starting points such as the door of a seminar room or a stair, while the red tents indicate exits. Notice that some characters take longer paths to facilitate the overall efficiency of the crowd, portraying a well-organized evacuation. If we need a less organized evacuation, where selfish characters seek for shorter path and form congestion, $\theta_{cooperation}$ can be adjusted.

The *typical gaming environment* is built to reproduce a common environment in real-time strategy games, in which the player has to move a crowd from one location to another in a complex terrain. As mentioned before, console games using traditional field-based crowd control would result in congestion under such a complex environment, which is frustrating for players and degrades the game quality. Using our method, we can obtain a high level of crowd coordination such that the crowd can move as quickly as possible.

Regarding the complexity of the method, the cost of computing the harmonic field and the guide lines are $O(h_{total})$ and $O(g_{total})$, respectively [DKG05]. Notice that differently from [DKG05], we do not perform remeshing but create guide lines at the same resolution of the input mesh. The complexity for calculating the maximum flow is $O(ev^2)$ [BK04], where e and v are the number of edges and nodes of the Reeb Graph, respectively, which are determined based on the number of obstacles and the resolution of the scene. Table 2 shows the percentage of computation time for different steps of topological scene analysis, which is obtained as the average values of the four scenarios in this section. Figure 12 shows how the computational cost changes by varying the number of obstacle, which modifies e

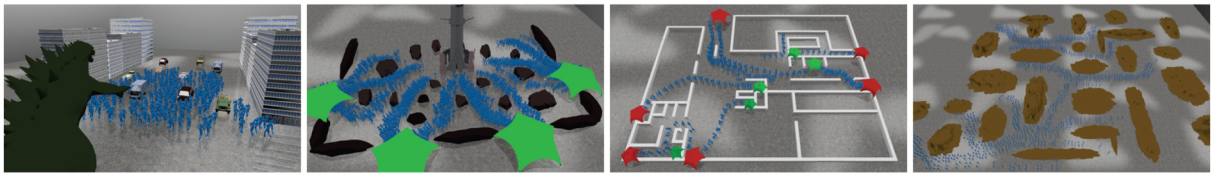

Figure 11: *Simulated scenarios: monster escape, castle invade, evacuation and typical gaming environment.*

Table 1: *Details of scenario simulation.*

Scenario	# char.	#obst.	Res.	Time (ms)
Monster	297	9	32	983
Castle	3348	23	32	2559
Evacuation	701	46	64	2356
Gaming	1144	26	32	2371

Table 2: *Computation time for each processing step.*

Processing step	% Computation time
Harmonic field creation	20.386%
Guide lines creation	10.653%
Reeb graph creation	11.543%
Capacities computing	13.442%
Maximum flow computing	0.009%
Logical route creation	23.145%
Geometric paths creation	19.911%

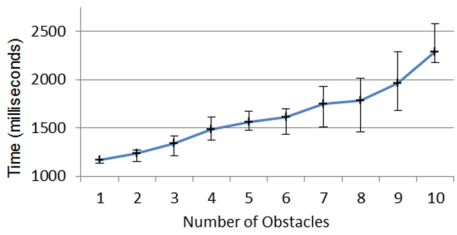

Figure 12: *The computation time against the number of obstacles. Standard deviations are shown as black line ranges.*

and v required in a scene and hence defines the complexity of the scene. The run-time computational complexity is purely dependent on the local planner. In our case, it is $O(n^2)$, where n is the number of character. This is because the flocking library we used calculates the influence from all characters in the scene for each character. For simulating large number of characters in real-time, a local influence approach can be implemented.

7.2. Comparison with other methods

In order to demonstrate the fluidity of the motion produced by our system, we compared it to the Continuum crowds method

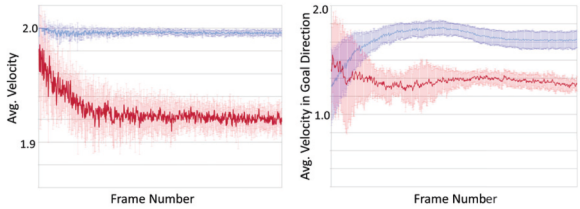

Figure 13: *The average velocity (left) the average velocity in goal direction (right) of our method (blue) versus the continuum crowd (red).*

[TCP06], which is one of the most popular field-based crowd control algorithms. It was chosen as a suitable candidate since it also provides characters with global knowledge of the scene without requiring detailed user input. The comparisons were made across 10 test cases, including environments containing bottlenecks and those where the crowd splits and flows require little prediction. Inspired by [WGO*14], we perform evaluation of the system to analyse the dynamism of the scene using the characters velocity.

The average magnitude (solid lines) and standard deviation (pale lines) of character velocity across all the scenes are shown in Figure 13, left-hand side, with the X-axis indicating simulation frame number. In our implementation, the maximum velocity of a character is 2.0. The result shows that our system achieves a near-optimal character speed with a significantly smaller variance across all test cases. The character speed of the Continuum crowd method drops to around 1.9 once the simulation starts, mostly because of congestion.

A high velocity character may not necessary moves towards the goal due to congestion. The average (solid lines) and the standard deviation (pale lines) of average velocity in the goal direction across all test cases are as shown in Figure 13, right-hand side. The result shows that the characters in our system move quicker towards the goal with an average velocity of 1.7, compare to the average velocity of 1.2 in the Continuum crowd system. Notice that in the first 50 frames, our system has a relatively lower average velocity towards goal. This is because the characters usually spread out in the beginning to efficiently navigate around obstacles in the scene.

We render one of the test cases to visualize the scene. As shown in Figure 14(a), with our system, the coordinated characters finish moving to the goal 300 frames faster than those from the Continuum crowds on average. Figure 14(b) shows that in the Continuum crowd system, all the characters rush towards the shortest route to the goal, which explains their high initial velocity towards goal. However,

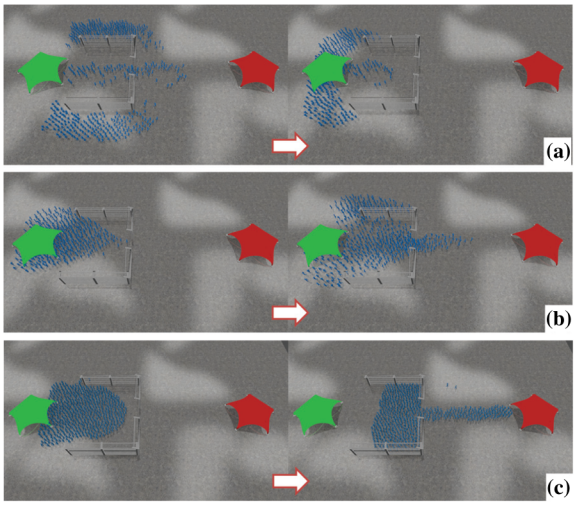

Figure 14: *Snapshots of (a) our system, (b) continuum crowds and (c) [vdBGLM11] in a bottleneck test case.*

due to congestion, the characters have to either stop or backtrack and find alternate paths, resulting in a significant drop in velocity towards goal.

We also compare our system with [vdBGLM11], which is an enhanced version of the reciprocal velocity obstacles algorithm [vdBPS*08]. Since [vdBGLM11] is an agent-based approach, characters do not have the information for global coordination. As a result, while characters can avoid collision with their neighbours locally, they cannot avoid congestion at bottlenecks. Figure 14(c) shows the snapshot of the result.

7.3. Simulation quality and computation time trade-off

It is easy to trade off the simulation quality and the computation time in our system. This is of particular importance in interactive applications such as real-time strategy games. Here, we define the simulation quality as simulation precision and the naturalness of crowd movement.

The trade-off can be adjusted in the following ways:

- **Reducing the number of guide lines g_{total} in Section** 4.2: This results in reducing the number of geodesic paths provided by each logical route. Crowd movement within the same logical route is then less spread out and natural.
- **Reducing the Reeb graph resolution R_{total} in Section** 4.3.1: This results in fewer Reeb nodes and small obstacles may not be registered in the topology of the scene.
- **Reducing the number of iso-flowlines i_{total} in Section** 4.3.2: This results in a less accurate estimation on the capacity of a Reeb edge, and hence a less accurate capacity for the logical routes. Minor congestion or under-filled paths may appear.

To evaluate the quality impact when adjusting the above parameters, an experiment is conducted as shown in Figure 15. Image (a) shows the simulation of our normal system, with $g_{total} = 1600$, $R_{total} = 64$ and $i_{total} = 8$. Image (b) shows the result of a fast simulation, with $g_{total} = 100$, $R_{total} = 32$ and $i_{total} = 1$. The red rectangles in the images highlight the impacts in synthesis quality, in which the crowd is less spread out and more artificial in the fast simulation.

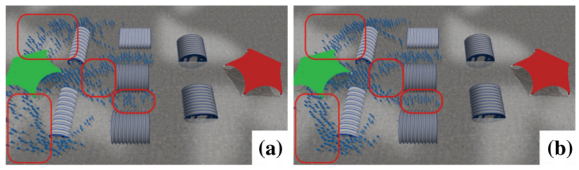

Figure 15: *Quality comparison between (a) normal speed simulation and (b) fast simulation.*

As for the computation time, the test scenes with 1, 2 and 15 obstacles require 812 ms, 1825 ms and 2714 ms, respectively using normal synthesis parameters. With the faster synthesis parameters, the computation time drops to 249 ms, 437 ms and 1138 ms, respectively.

By adjusting the trade-off, our system is fast enough to run in interactive applications where performance is of higher priority than simulation quality. For example, in computer games, we can dramatically tune down the mentioned parameters to achieve better performance. Artifacts such as less spread out crowd, paths of coarser resolution and slightly over-/under-filled paths may occur, but they will not heavily affect the gameplay.

8. Discussions

As an improvement from previous method, the geodesic paths generated by our system can prevent characters from crossing one another's paths. They are compatible with many local character control algorithms. We use flocking as our local planner to provide an unvarnished insight into the path quality, but more dedicated planner can be used.

The algorithm is capable of handling characters of different sizes or speeds by adjusting the equation to calculate the max-flow values of the routes, in which the parameters can be obtained from real-world statistics. For instance, slower and bigger characters occupy the more space for longer duration, and hence the flow values of the routes are decreased. We can also simulate characters of different intelligences and behaviour using cooperative and congestion controls. Finally, specific agent-based behaviour can be simulated by designing the local agent controller.

Although methods such as [KLLT08, HSK14] can potentially simulate scenes similar to those we created, they would require significant manual tuning and control by the animator. Specifically, all characters would need some form of pre-assigned paths to avoid causing congestion in any bottlenecks. This process is performed automatically in our system.

In the current implementation of our system, we minimize the crossing of logical routes and hence do not produce scenes that involve crossing groups of characters, such as two crowds meeting

Figure 16: *A potential limitation of the Max-flow framework.*

at a crossroads. If such a kind of scene is needed, we can update the logical route creation algorithm in Section 5.1, as well as change the way we deal with split nodes and merge-split nodes, to synthesize crowd crossing behaviour.

The grid resolution in Section 4 affects the quality of harmonic field gradient, guide lines, Reeb Graph and iso-flowlines. When tuning the resolution, we make sure that both the Reeb Graph and the grid of the scene have enough detail to capture the smallest object in the scene. In a large scene that contains small objects, a hierarchical grid resolution can be used such that we assign coarser resolution to open area and finer resolution around the objects. This can maintain a high quality of simulation with a reasonable computational cost.

8.1. Limitations

Our system aims at maximizing the flows of the crowd in order to avoid congestion, instead of minimizing the time for all characters to arrive the destination. The main reason of the design is that once congestion occurs, local controllers would not be able to carry out the solution from the high-level planner, which would deficit the purpose of the planner. Furthermore, accurately estimating the timing information requires the high-level planner to communicate with the local controller and perform path replanning during run-time, which would increase the run-time computational cost. In most situations, maximizing flows means minimizing time. However, in extreme situations such as the one show in Figure 16, our system would maximize the flow of the shorter solid route, and then start to fill the longer dotted route as well to avoid congestion, as described in Section 4.3.3. It can be expected that characters following the shorter route would arrive the goal much earlier than those taking the longer one. If overly long routes should be avoided in the cost of congestion, a simple solution is to design a route filtering pre-process before computing the flows. Another possible solution is to reduce the flows of the paths based on the approximated distance to the goal, such that longer paths are less preferred.

Another limitation of our system is that we assume characters to have homogenous characteristics. If different characters exist in the crowd, such as some moving slower than the others, congestion may occur. To minimize the impact of this problem, one solution is to estimate the mean value of the characters' parameters when evaluating the flows. Hybrid approaches that combine field-based and agent-based algorithms can also be explored.

9. Conclusion and Future Work

In this paper, we propose a new method for producing coordinated crowds moving through a given scene. This is done by global field-based planning such that congestion is minimized and the dynamism of the crowd is preserved. We also present high-level user controls to alter the crowd behaviour using the degree of cooperation and congestion. Through our experiments, we demonstrate how our system can efficiently synthesize smooth crowd movement with minimal congestion in different scenarios up to thousands of characters.

In the future, we will investigate network flow models so that the system can deal with dynamic obstacles and re-plan the distribution of characters in the background. Since our topological scene analysis framework requires a lot of geometric operations, we are also interested in applying graphics processor unit (GPU) computation to further speed up the system, such that it can be applied in limited devices such as game consoles.

Acknowledgment

This project was supported by the Engineering and Physical Sciences Research Council (EPSRC) Ref: EP/M002632/1.

References

[ACSD*03] Alliez P., Cohen-Steiner D., Devillers O., Lévy B., Desbrun M.: Anisotropic polygonal remeshing. *ACM Transactions on Graphics 22*, *3* (2003), 485–493.

[BCK13] Barnett A., Choi M., Komura T.: Topology-based global crowd control. In *The 26th International Conference on Computer Animation and Social Agents* (Istanbul, 2013), John Wiley & Sons Ltd.

[BK04] Boykov Y., Kolmogorov V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, *9* (2004), 1124–1137.

[CFS82] Chalmet L., Francis R., Saunders P.: Network models for building evacuation. *Management Science 28*, *1* (1982), 86–105.

[CJ03] Cova T., Johnson J.: A network flow model for lane-based evacuation routing. *Transportation Research Part A: Policy and Practice 37*, (2003), 579–604.

[Dav06] Davis T. A.: *Direct Methods for Sparse Linear Systems*. SIAM, 2006. http://dx.doi.org/10.1137/1.9780898718881.

[DKG05] Dong S., Kircher S., Garland M.: Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer Aided Geometric Design 22*, *5* (2005), 392–423.

[FS97] Feder H., Slotine J.-J.: Real-time path planning using harmonic potentials in dynamic environments. In *Proceedings of 1997 IEEE International Conference on Robotics and Automation* (Albuquerque, New Mexico, 1997), IEEE, vol. 1, pp. 874–881.

[GO07] Geraerts R., Overmars M.: The corridor map method: A general framework for real-time high-quality path planning. *Computer Animation and Virtual Worlds 18*, (2007), 107–119.

[Hen74] Henderson L.: On the fluid mechanics of human crowd motion. *Transportation Research 8*, *6* (1974), 509–515.

[HSK12] Henry J., Shum H. P. H., Komura T.: Environment-aware real-time crowd control. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2012), SCA '12, Eurographics Association, pp. 193–200.

[HSK14] Henry J., Shum H. P. H., Komura T.: Interactive formation control in complex environments. *IEEE Transactions on Visualization and Computer Graphics 20*, *2* (2014), 211–222.

[HSKK01] Hilaga M., Shinagawa Y., Kohmura T., Kunii T.: Topology matching for fully automatic similarity estimation of 3d shapes. SIGGRAPH '01, ACM, pp. 203–212.

[JPCC14] Jordao K., Pettré J., Christie M., Cani M.-P.: Crowd Sculpting: A space-time sculpting method for populating virtual environments. *Computer Graphics Forum 33*, *2* (2014), 351–360.

[KF85] Kisko T., Francis R.: Evacnet+: A computer program to determine optimal building evacuation plans. *Fire Safety Journal 9*, *2* (1985), 211–220.

[KG14] Küllü K., Güdükbay U.: A layered communication model for agents in virtual crowds. In *Proceedings of 27th International Conference on Computer Animation and Social Agents* (Houston, USA, May 2014).

[KGvdS13] Karamouzas, I., Geraerts, R., van der Stappen, A. F.: Space-time group motion planning. In *Algorithmic Foundations of Robotics X*, Frazzoli, E., Lozano-Perez, T., Roy, N., Rus, D., (Eds.), vol. *86* of *Springer Tracts in Advanced Robotics*. Springer, Berlin Heidelberg, 2013, pp. 227–243 http://dx.doi.org/10.1007/978-3-642-36279-8_14.

[KLLT08] Kwon T., Lee K. H., Lee J., Takahashi S.: Group motion editing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (New York, NY, USA, 2008), ACM, pp. 1–8.

[KSK97] Kanai, T., Suzuki, H., Kimura, F.: 3d geometric metamorphosis based on harmonic map. In *Computer Graphics and Applications, 1997. Proceedings., The Fifth Pacific Conference on* (Oct 1997), pp. 97–104, doi: 10.1109/PCCGA.1997.626179.

[KSKL14] Kim J., Seol Y., Kwon T., Lee J.: Interactive manipulation of large-scale crowd animation. *ACM Transactions on Graphics 33*, *4* (2014), 83:1–83:10.

[KSLO96] Kavraki L., Svestka P., Latombe J.-C., Overmars M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, *4* (1996), 566–580.

[LK06] Lau M., Kuffner J. J.: Precomputed search trees: Planning for interactive goal-driven animation. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), Eurographics Association, pp. 299–308.

[PAB07] Pelechano N., Allbeck J. M., Badler N. I.: Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), SCA '07, Eurographics Association, pp. 99–108.

[Par10] Park M.: Guiding flows for controlling crowds. *Visual Computer 26*, *11* (2010), 1383–1391.

[PCM*06] Pettré J., Ciechomski P. d. H., Maïm J., Yersin B., Laumond J.-P., Thalmann D.: Real-time navigating crowds: Scalable simulation and rendering. *Computer Animation and Virtual Worlds 17*, *3–4* (2006), 445–455.

[PPD07] Paris S., Pettré J., Donikian S., : Pedestrian reactive navigation for crowd simulation: A predictive approach. *Computer Graphics Forum 26* (2007), 665–674.

[PvdBC*11] Patil S., van den Berg J., Curtis S., Lin M., Manocha D.: Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics 17*, *2* (2011), 244–254.

[Ree46] Reeb G.: On the singular points of a completely integrable pfaff form or of a numerical function. *Comptes Randus Académie des Sciences Paris 222*, (1946), 847–849.

[Rey87a] Reynolds C.: Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH 21*, *4* (1987), 25–34.

[Rey87b] Reynolds C. W.: Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Computers & Graphics 21*, *4* (1987), 25–34.

[RI05] Rosell J., Iniguez P.: Path planning using harmonic functions and probabilistic cell decomposition. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005.* (2005), IEEE, pp. 1803–1808.

[SFF*10] Silveira R., Fischer L., Ferreira J., Prestes E., Nedel L.: Path-planning for rts games based on potential fields. In *Motion in Games*, Boulic R., Chrysanthou Y., Komura T., (Eds.), vol. *6459* of *Lecture Notes in Computer Science* (Zeist, Netherlands). Springer, Berlin Heidelberg, 2010, pp. 410–421. http://dx.doi.org/10.1007/978-3-642-16958-8_38.

[SH12] Shum H., Ho E. S.: Real-time physical modelling of character movements with microsoft kinect. In *Proceedings of the 18th ACM Symposium on Virtual reality software and technology* (New York, NY, USA, 2012), VRST '12, ACM, pp. 17–24.

[SKK91] Shinagawa Y., Kunii T., Kergosien Y.: Surface coding based on morse theory. *IEEE Computer Graphics and Applications 11*, *5* (1991), 66–78.

[SKY08] Shum H. P. H., Komura T., Yamazaki S.: Simulating interactions of avatars in high dimensional state space. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2008), ACM, pp. 131–138.

[Sno00] Snook G.: Simplified 3d movement and pathfinding using navigation meshes. *Game Programming Gems 1*, (2000), 288–304.

[ST05] Shao W., Terzopoulos D.: Autonomous pedestrians. In *Eurographics* (2005), SCA '05, ACM, pp. 19–28.

[TCP06] Treuille A., Cooper S., Popović Z.: Continuum crowds. *ACM Transactions on Graphics 25*, *3* (2006), 1160–1168.

[TSO*10] Torchelsen R. P., Scheidegger L. F., Oliveira G. N., Bastos R., Comba J. a. L. D.: Real-time multi-agent path planning on arbitrary surfaces. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 47–54. http://doi.acm.org/10.1145/1730804.1730813.

[VDAGHP10] van den Akker M., Geraerts R., Hoogeveen H., Prins C.: Path planning for groups using column generation. In *Motion in Games*, Boulic R., Chrysanthou Y., Komura T., (Eds.), vol. *6459* of *Lecture Notes in Computer Science* (Zeist, Netherlands). Springer, Berlin Heidelberg, 2010, pp. 94–105. http://dx.doi.org/10.1007/978-3-642-16958-8_10.

[vdBGLM11] van den Berg J., Guy S., Lin M., Manocha D.: Reciprocal n-body collision avoidance. In *Robotics Research, Springer Tracts in Advanced Robotics*. C. Pradalier, R. Siegwart, G. Hirzinger, (Eds.), Springer, Berlin, Heidelberg (2011), Vol. 70, pp. 3–19.

[vdBPS*08] van den Berg J., Patil S., Sewall J., Manocha D., Lin M.: Interactive navigation of multiple agents in crowded environments. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2008), I3D '08, ACM, pp. 139–147.

[vTCG12] van Toll W. G., Cook IV A. F., Geraerts R.: Real-time density-based crowd simulation. *Computer Animation and Virtual Worlds 23*, *1* (2012), 59–69.

[WGO*14] Wolinski D., Guy S., Olivier A., Lin M., Manocha D., Pettré J.: Parameter estimation and comparative evaluation of crowd simulations. *Computer Graphics Forum* (2014), 33, 303–312, doi: 10.1111/cgf.12328.

[YCP*08] Yeh H., Curtis S., Patil S., van den Berg J., Manocha D., Lin M.: Composite agents. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer Animation* (2008), ACM, pp. 39–47.

[ZRKS05] Zayer R., R ssl C., Karni Z., Seidel H.: Harmonic guidance for surface deformation. In *Eurographics 05*, (2005), 601–609.