2014-06-30

Improved Computer-Generated Simulation Using Motion Capture Data

Seth A. Brunner
Brigham Young University - Provo

Improved Computer-Generated Crowd Simulation Using Motion Capture Data

Seth A. Brunner

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Parris Egbert, Chair
Michael Jones
Mark Clement

Department of Computer Science

Brigham Young University

July 2014

ABSTRACT

Improved Computer-Generated Crowd Simulation Using Motion Capture Data

Seth A. Brunner
Department of Computer Science, BYU
Master of Science

Ever since the first use of crowds in films and videogames there has been an interest in larger, more efficient and more realistic simulations of crowds. Most crowd simulation algorithms are able to satisfy the viewer from a distance but when inspected from close up the flaws in the individual agent's movements become noticeable. One of the bigger challenges faced in crowd simulation is finding a solution that models the actual movement of an individual in a crowd. This paper simulates a more realistic crowd by using individual motion capture data as well as traditional crowd control techniques to reach an agent's desired goal. By augmenting traditional crowd control algorithms with the use of motion capture data for individual agents, we can simulate crowds that mimic more realistic crowd motion, while maintaining real-time simulation speed.

Keywords: Graphics, Crowd Simulation, motion capture, natural phenomena

ACKNOWLEDGMENTS

Thanks goes out to my professor Dr. Parris Egbert, family and especially my wife Bethany and my son Killian whom without I probably never would have finished.

Table of Contents

Chapter 1

Introduction

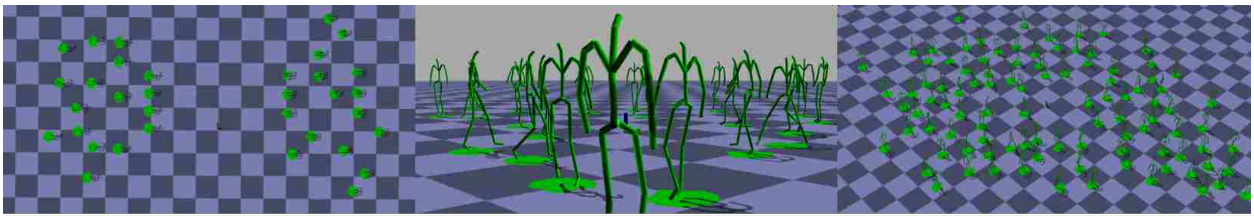


Figure 1.1: Screen shots from our crowd simulation system. The left most picture is an image of two crowds moving towards each other. The center image is a close up of the crowds in our system. The far right is a crowd formed when the agents were put in a cirlce and try to get around each other.

As far back as 1987 [17], crowds were becoming an interesting problem for computer aided animation. Producing very large crowds for live action films was impossible without hiring a large number of extras to create the desired crowd. An early example is Cecil B. Demille's The Ten Commandments. Demille hired thousands of extras for the exodus scene. Large crowds of people create the difficulty of directing the crowd to get the desired appearance.

Many approaches have been created to remedy the problem of creating a large crowd that is cost effective and directable. One technique is through animation. Animated crowds used for live action and animated films are prohibitively expensive to animate by hand because of the amount of time that the artist would need to spend on each individual in the crowd. Each agent needs to be animated realistically in a fashion that makes them appear as individuals while still maintaining the look and feel of an autonomous crowd. As the size of the crowd increases the complexity of the movement of the crowd increases for the animator.

As computers became more powerful, solutions to creating dynamic crowds through procedural means became more popular. The need for more realistic crowds has pushed research into various areas, each one focusing on weaknesses in the most simplistic crowd simulation model. Up until now, these areas have focused on how to make the overall crowd motion look more realistic, rather than focusing on the possible movements of the individuals within the crowd. The result is that you can simulate overall crowd movement that looks good from a distance but as you get close to the individuals within the crowd there are many artifacts that are distracting for the viewer and destroy the illusion that the crowd is real. The motion picture industry has dealt with this problem in films such as The Lord Of The Rings Trilogy and The Lion King. Some common techniques are keeping the camera a large distance away from the crowd, or by having the crowd or camera moving so fast that details about the specific characters are blurred or difficult to notice.

The goal of this work is to address the unrealistic movement of the individual agents of the crowd while maintaining visually pleasing overall crowd movement. This work takes the approach of traditional crowd simulation where the agents are represented by a position in space and a velocity. During each movement planning phase the agents determine their current desired velocity and orientation. In the planning step of the crowd simulation process the agents are limited to a specific set of movements that have been collected from motion capture data. The motion capture data represents movement that humans are capable of making. This will ensure that the agents are making movements we are used to seeing people make everyday. Our solution produces realistic individual agent movement while at the same time generating realistic crowd movement. We are able to do so in interactive-time with a reasonable number of agents.

The three main contributions of this work are 1) our use of real-world data in the planning step of crowd simulation, 2) our weighting function for determining which real-world data will be of best use and 3) our hybrid obstacle avoidance algorithm called Cell Marking.

Chapter 2

Related Work

2.1 Crowds From Rule-based Models

In 1985 Amkraut, Girard, and Karl submitted a piece to The Electronic Theater at SIGGRAPH. This piece featured a flock of birds that flew through a virtual environment. This example is one of the first crowds explored in graphics research. Two years later, Reynolds published Flocks, Herds and Schools: A Distributed Model[17] where he presents what has become known as boids or "bird-oid" objects. Boids are objects in a three dimensional space that have the following behaviors: 1) Collision Avoidance, 2) Velocity Matching and 3) Flock Centering. Each of these behaviors is represented as a vector with a magnitude between 0 and 1. They are then averaged together using a weighted average and a priority system. Although animal flocking behavior and human crowd behavior are not identical, Reynolds did adapt pieces of his flocking simulation to human characters [18]. Fiorini[7] developed an obstacle avoidance algorithm called velocity obstacles. This algorithm is similar to a maneuvering board used in the Navy. It uses the radii and velocities of the objects in a scene to determine the best possible velocity for the current agent. The algorithm scales well for multiple dynamic obstacles in a scene and has been used widely for crowd simulation to guarantee that individuals in a crowd do not collide with each other or with other obstacles placed in an environment. The work by Vandeberg et. al. [23] extended the velocity obstacle idea and was able to eliminate oscillations in the behavior of the agents as they chose their paths. Xiong [24] presents a rule based method for crowd control. His solution considers all of the valid velocities present, then uses a cost function to determine a score for each

valid velocity and a set of predefined rules to determine which velocity is the best choice in that scenario. These cost functions are able to choose proper obstacle avoidance paths and trajectories for his individual agents. However, the underlying functions do not take into account the real world data available to the individual agents and the possible motions they could use for avoidance and for reaching their goals.

2.2 Crowds From Social Forces

One of the key papers on human motion in crowds is the paper titled "Social Force Model For Pedestrian Dynamics" [10] by Helbing et. al. Even though crowds for humans had been addressed before, Helbing was the first to describe how social forces can be applied to human crowds. In addition to achieving their goals, the individuals are influenced by others through avoidance of strangers or attraction towards friends or friendly objects. Similarly Braun et. al. [2] expanded on that work to make agents have their own personality. These social forces are defined in each individual in the crowd and their desired orientation and velocity vectors are determined by how these different social forces add up. Later work by Pelechano et. al. [16] improved the social forces method by including the simulation of an increased number of agents as well as social actions, such as pushing and emotional forces. The work described in [11, 19] explores research in psychology to model the behavior of crowds. Ricks and Egbert [19] use transactional analysis to drive the individual behaviors. They found that by using transactional analysis, crowds could be created that would stop to talk to each other or even walk side by side while heading to their destinations. Kim et. al. [11] use what they call "General Adaptation Syndrome" to model how individuals in a crowd will react in certain circumstances and to certain levels of stress. Initially people will start in a calm and relaxed state. When an irritant is introduced, their stress levels will rise and will drive them to make decisions. If the high stress levels are not lowered or the situation resolved, the person would die according to the theory. If the stressant is removed, then the person will slowly return

to a normal state. This is applied to agents in a virtual world to achieve realistic behaviors from agents within a crowd.

These papers present a simple underlying model for how crowds should be constructed. However, they fail to address how the integrity of the motion of the crowds may be compromised by not involving real world data for available actions of a human agent in the decision making process for the final velocity and orientation.

2.3 Crowds From Cellular Automata

Cellular Automata models for crowd simulation provide a very basic representation of crowd simulation. The work presented in [1, 3, 9, 20] demonstrates how cellular automata can be used for crowd simulation. The idea behind cellular automata is that the world is divided into individual cells that can be occupied by one individual at a time or can be represented with a density of many agents. With each iteration of the simulation, the agents move into the empty or less dense cells around them that move them closer to their desired location. The underlying movement from cell to cell is sometimes called flow fields.

Chenney [4] improved on cellular automata techniques by creating velocity fields in individual tiles. The tiles are then stitched together and the edges and boundaries are corrected to accommodate the flow between tiles. Agents are then placed on the tiles, and the agents follow the flow creating a crowd simulation.

Tecchia et. al.[21] also improved upon existing cellular automata algorithms to build crowds. They divide the crowd into 4 cellular automata layers. The first layer is for resolving all collisions between dynamic agents. The next layer is used to determine any collisions between the environment and the specific agent. The third layer is used to define possible behaviors of the agent based on their position in the grid and environment. The last layer is used to relay to the agent possible actions e.g. pushing an elevator button. They are able to accomplish directability with their agents and give the user a lot of power in creating realistic dynamic crowds.

The simplicity of the cellular automata algorithm makes these methods desirable. The world is discretized and therefore makes it easy to know who is where and how to keep agents from interpenetrating each other. However, these types of algorithms alone are inadequate for representing real crowds. They require a high resolution grid to get realistic motion from individuals and they do not take into account planning the motion and trajectory of an individual in the crowd from the possible real world actions and movements that are available to a specific agent.

2.4 Crowds From Motion Capture Data

More recently, work has been done to bring motion capture and real world data into crowd simulations. Lee et. al [13] and Courty [5] present techniques for crowd animation using motion capture data. Although the techniques have differences, the underlying principles are very similar. They film crowds and capture the motion of individual agents within a crowd. From this information they build a velocity field that represents how the crowd moves within the environment. They are then able to place agents within the virtual environment and get realistic crowd motion. This motion capture focuses on the overall movement of the crowd and uses the motion capture to extract the behavior of the crowd. However, it does not address the individual agent's movement as a product of motion capture data and traditional crowd simulation techniques.

Lerner et. al. [14] also use motion capture to create realistic crowd motion. Their technique extracts individual behavior in certain situations that might be experienced in a virtual world. They extract the proper velocity vector of the reaction captured and store these reaction vectors in a database. In the agent's planning phase they take the state of the virtual world and compare it with the states of the world for the reaction vectors. They then choose the reaction for the state that best matches the situation they are in. They do focus on real world data to create their reaction velocity vectors. However, the goal of their paper focuses on creating crowds through example by searching a database for reactions to

certain scenarios. Our system differs from this in that we are trying to take traditional crowd simulation algorithms that have proven to create visually appealing crowds and improve the individuals' movements within those crowds.

2.5 Other Approaches

Fulgenzi et. al. [8] combine Velocity Obstacles with Cellular Automata using probabilities to determine future collisions. The system is meant for determining robot movement in a dynamic world and does not scale well to simulating characters in a crowd. They are successfully able to navigate a world using one agent on their grids.

In summary the current crowd simulation models fail to address the problem of the movement of individuals within a crowd. The main focus in the current systems is the overall motion of the crowd and individual motion is largely ignored. By ignoring the individual's motion a wide range of possible crowd motions and uses are lost.

2.6 Motion Transitions

Maintaining realistic motion throughout a chosen movement is important and stitching together multiple motions requires generating transition motion on the fly. Egbert et al. in [6] present research into transitions between motions and do so in a four part process. They 1) look for points in the two motions that are optimal for transitioning, 2) align the motions to get as little difference between the two transition frames as possible, 3) search through their motion libraries for an example transition that best fits the situation and 4) modify the motion between the two frames to create a transition. They are able to achieve realistic and smooth transition motion in real-time.

Kovar et. al. in [12] present a different solution to generating motion transitions. They first would put all motions into a "'Motion Graph" where each edge represents a clip of motion and nodes are points that connect the motions. To determine which two motions can be stitched together they take a window of the two motions they are comparing. This

window corresponds to the end of the current motion and beginning of the next motion. A distance formula is used to determine how far apart each frame in this window is from its corresponding frame in the next motion. The window size used in the paper is one third of a second. This comes out to 10 frames at 30 frames a second.

Chapter 3

Realistic Motion in Crowds

The main problems with the current crowd simulation models are that they focus on the behaviors of individuals within the crowd and how that shapes crowd behavior but ignore the actual real-world motions that individuals can make. When crowds are formed without the real-world data playing a part in the planning step, the agents are able to make movements that are unrealistic. Figure 3.1 shows an agent making a turn. It demonstrates how much foot skating happens during a turn with traditional methods. Our work overcomes these obstacles by introducing real-world data into the planning step of the agents in the simulation. The three main contributions of this work are our use of real-world data in the planning step of crowd simulation, our weighting function for determining which real-world data will be of best use and our hybrid obstacle avoidance algorithm called Cell Marking.

3.1 Base Simulation

Making a simulation that runs quickly over hundreds or thousands of agents is important for crowd simulations. In order to make this happen, research turned quickly to a very simple model for representing each agent. Each individual within the crowd was represented as a point in the simulation space and a velocity vector. The simplicity of this model allows the simulation to scale well.

Current systems have used the point and velocity model since the earliest crowds and it has worked well because of its ability to scale to a large number of agents. After the paths of the agents have been planned, an animation cycle is applied to the agents in the

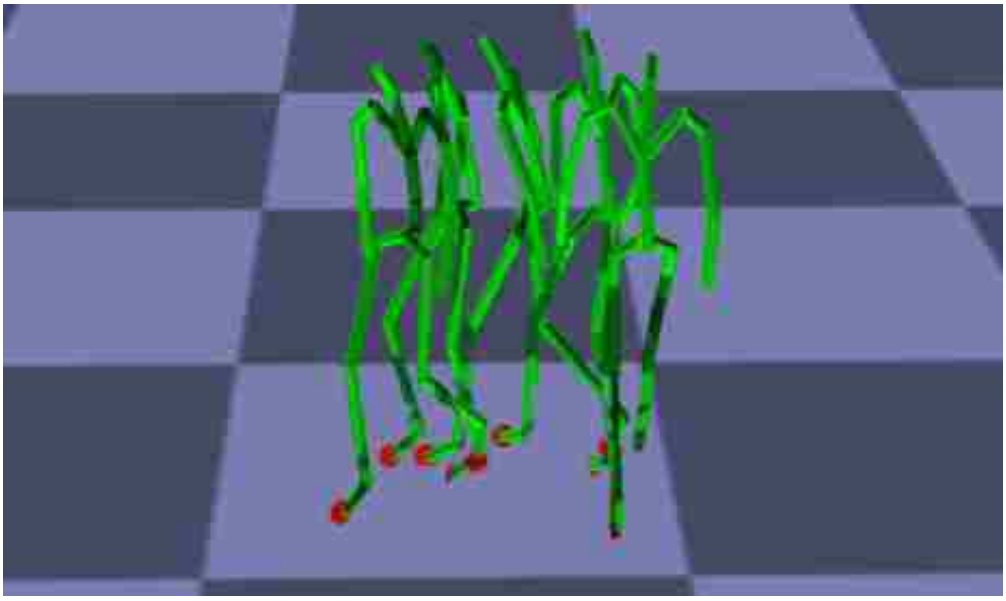


Figure 3.1: An agent makes a turn using traditional crowd simulation. The red circles represent each time the agent turns or moves without moving its feet in the direction it is moving.

crowd simulation to give viewers the feeling that these agents are moving as a result of their animation. However, this process carries with it a lot of undesirable artifacts. Individuals in the crowd are able to run faster or slower than their animation is moving, turn without making similar body motion and they may miss valuable human movements such as sidestepping and pivot turning. Figure 3.6 shows an example of an agent sidestepping using motion capture. Our work overcomes the movement artifacts seen in current crowd systems by using real-world data to determine how the agents should move.

3.2 Augmented Simulation

Our simulation builds on this base simulation previously described by representing each agent as a point in space and a velocity vector. Keeping track of this information is useful for the obstacle avoidance algorithm and also in determining the desired direction the agent should be moving to reach its goal. The first thing we want to do in each planning step is to have each agent determine what his desired velocity will be. The desired velocity represents the

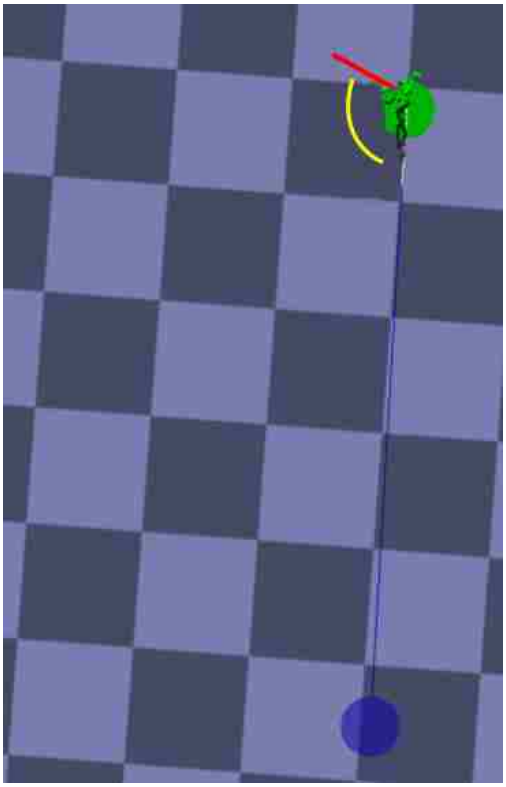


Figure 3.2: An agent plans his next move. The red line represents his current velocity and orientation. The blue circle represents the agent's goal. The blue line from the agent to the goal represents the agent's desired velocity and orientation. The yellow arc represents the change in orientation the agent needs to make to reach its goal.

velocity that the agent needs to move himself closer to his goal while still avoiding other agents and obstacles within a scene. Figure 3.2 illustrates the explained example.

To find the desired velocity we first find the velocity that will point us closest to our goal. To determine that we find the vector from the agent to the goal. This will give us our desired velocity and the direction we need to move to get closer to our goal. Next we need to change the desired velocity to avoid other obstacles and agents within the simulation.

The underlying base simulation of our algorithm is the Reciprocal Velocity Obstacles, or RVO, algorithm presented in [23]. RVO is a traditional rule based algorithm that uses a series of rules to navigate an agent within a crowd.

The algorithm builds on the base simulation of having a point, a velocity and a goal. It then uses the velocities and geometry of obstacles and other agents within the scene to

choose an optimized velocity that will be close to, but not necessarily in the exact direction of the goal. Each agent is represented by a radius and a velocity vector. During the planning phase of each step, the following steps are performed for each agent:

1. The current agent's radius is reduced to a point
2. Each of the other obstacles or agents radii in the scene are increased by the value of the current agent's radius.
3. A velocity obstacle shape is created for each of the other agents and obstacles in the scene. This is done by taking the position of the current agent and finding a point on the circle created by the other agent's position and its increased radius that creates a line that is tangent to the circle created at the other agent's position. There are two tangent lines from this point to that circle. Both of these tangents make up the velocity obstacle shape. See Figure 3.3c for a visual representation of the velocity obstacle shape.
4. Transform the Velocity Obstacle from step 3 in space by the velocity of the other obstacle or agent.
5. Choose a velocity that is outside of the Velocity Obstacle created from step 4.

Once these steps have been performed for the first object, the space within the velocity obstacle will represent all velocities that could result in a collision between the two agents. This algorithm is then applied to each of the other objects and agents in the scene. The union of all of the velocity obstacles is then computed. It is the set of final vectors that could result in a collision. Once that final set of velocities has been determined, collision free movement can be guaranteed by choosing a velocity outside of that velocity obstacle. Figure 3.3 demonstrates this process.

Our obstacle avoidance algorithm is a hybrid algorithm that combines rule based methods like Reciprocal Velocity Obstacles, RVO, and a grid based obstacle avoidance algorithm. We discuss the details of this algorithm in section 4. With the new desired velocity

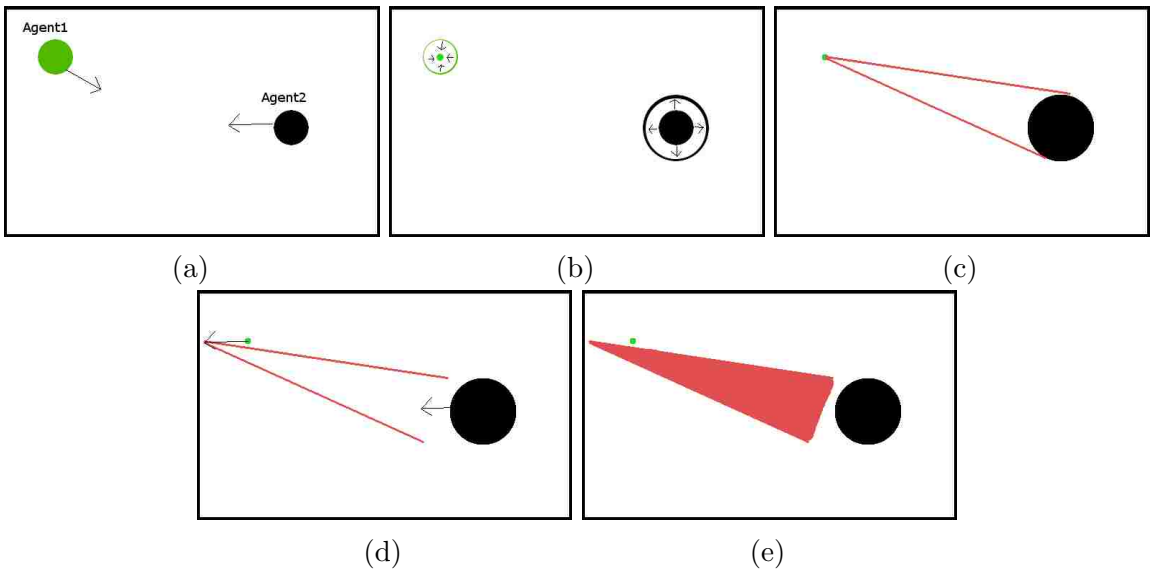


Figure 3.3: The Reciprocal Velocity Obstacle algorithm. (a) shows how the agents are layed out in a scene. (b) shows Agent1's reduced radius and Agent2's increased radius. Agent2's radius increases by the radius of Agent1. (c) shows the Velocity Obstacle being constructed from Agent1's position and the lines tangent to Agent2's new radius. (d) shows the velocity obstacle being translated by the velocity of Agent2. (e) shows the final velocity obstacle. The filled in triangle represents all velocities that Agent1 can choose that will result in a collision between Agent1 and Agent2.

produced from the velocity to the agent's goal and the obstacle avoidance algorithm, we are then able to choose a motion capture clip that will move us toward our goal while still avoiding other agents. The manner in which we select the particular motion capture data to be used is presented in the next section.

3.3 Motion Capture Data

The motion capture data we use for this system comes from the Carnegie Melon Motion Capture Library [22]. The motion capture data from this library represents our real-world data that we use for planning. The motion segments represent long movements like a walking turn, a side step, and a turn and walk. Most of them will take longer than a couple of seconds. Figures 3.4, 3.5 and 3.6 show visualizations of some of the sample motion capture data. In order to be able to get the most realistic motion for the agent we would need to use the entire segments recorded. However, to get the most realistic motion for the overall crowd we need

to give the agent the ability to choose his motion as often as possible, sometimes multiple times per second. The goal is to create realistic motion segments for individuals that allow for an overall realistic crowd movement. Thus, we need to find a suitable compromise that will reasonably meet both requirements. We found that a full second of motion capture data applied to the agent appears realistic and still allows us to maintain realistic crowd motion.

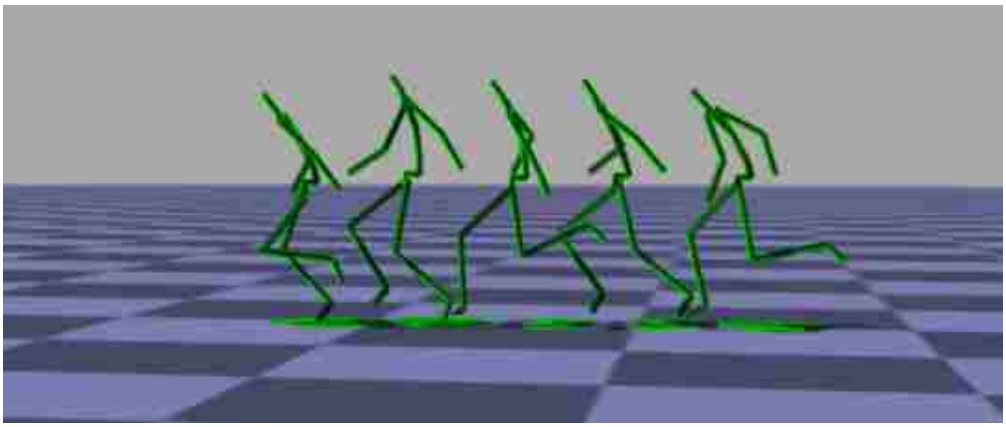


Figure 3.4: Run motion capture data sample.

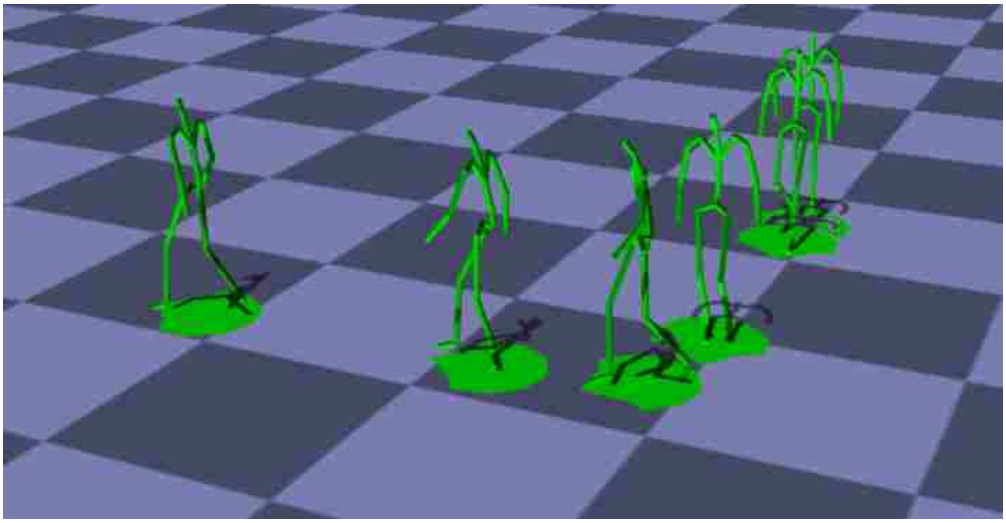


Figure 3.5: Motion capture sample of an agent making a sharp turn.

This means that we can choose a small one second segment from all of the motion capture data and apply it to our agent. The agent will execute the selected motion capture clip and then choose another one second segment of motion capture data to run. The resulting

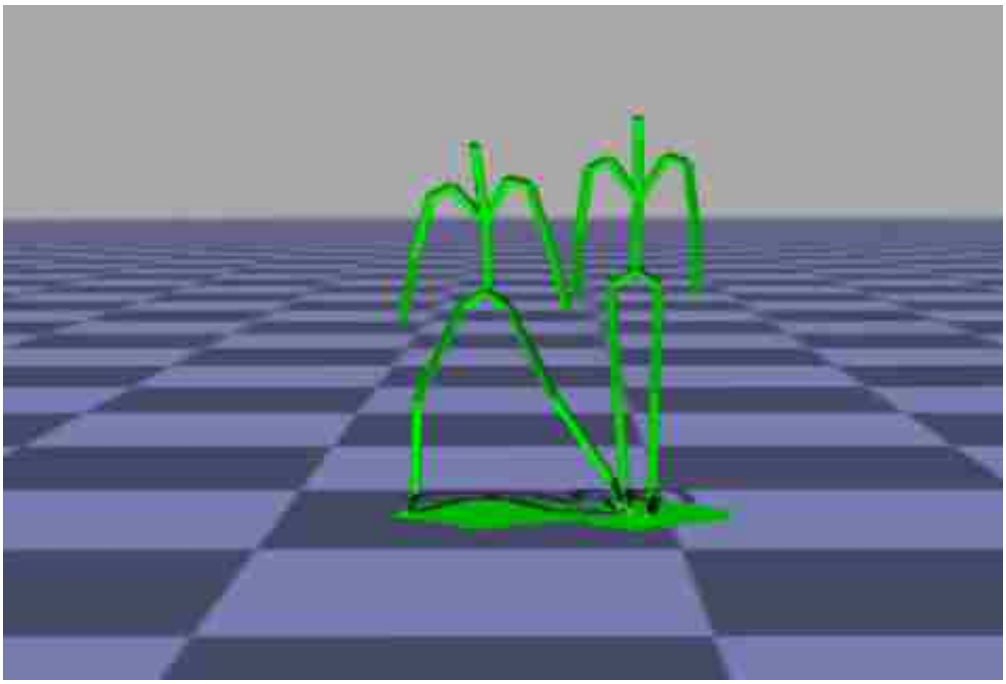


Figure 3.6: Sample motion capture of an agent side stepping.

compilation is a series of unrelated motions. Left in this state, the agent's movements will spontaneously pop and will seem very disjointed.

3.4 Transitions

Motion that is popping and disjointed will not appear realistic. To overcome the unrealistic motion, we apply transitions between the two motion capture segments being played. The work presented in [6] does a good job of creating real-time motion transitions and an overview of their approach can be found in section 2. Our approach to this problem is similar to parts of their work and the work in [12]. To be specific our goal was to reduce the amount of foot skate and unrealistic motion in a transition. To do so we used the idea of motion alignment from [6] and frame differencing from [12]. Getting good transitions between motions produces realistic individual motion but always choosing the next motion that will give us the best transition will not always produce the most realistic crowd motion. There are multiple

variables that go into deciding the best next motion. The process for picking a new motion and how transitions play into that is described in the next section.

To successfully use the transitions in our project we change a few of the assumptions we originally made when selecting motion. Instead of choosing just one second of motion we choose one third of a second more frames beyond the number of frames in one second of motion, in this case 10 frames. Our program assumes that there are 33 frames in one second of motion. To accurately do transitions we will need to guarantee that our motion has 43 continuous frames. The transition then happens with the last 10 frames of the previous motion and the first 10 frames of the new motion. A new motion is then produced for the first 10 frames of each new motion by interpolating between the overlapping frames. The downside of interpolating between frames is that it is possible to choose two motions that are very different from each other and then produce foot skate and potential for unrealistic rotation of joints. The benefits is that the transitions are quickly computed and, if the two motions are close enough to each other, a realistic transition.

3.5 Choice Function

Knowing which motion capture segment to choose next can be a difficult decision to make. [6] determined which motion segment to choose next by finding the set of frames in the next motion that best aligns with the previous frames. Their work has the benefit of using entire motions and they have categorized their motions as such, e.g. walk, jump, skip, etc. Choosing any frame within the range of the next motion allows them easily find and match up frames for transitioning. Our motion segments are not so simple. The goal is to 1) choose the motion that best achieves our desired goal and 2) allows us to have a realistic transition. This means the segment needs to move in the direction of the desired velocity and also needs to be as fast as the desired velocity. The final orientation of the movement needs to be close to the desired orientation for the agent. It also needs to be collision free and be able to minimize foot skate and error in the motion when the transitions happen.

Our system uses a choice function that takes all of these requirements into account while choosing the next best motion segment. We score the motions with the following function:

$$\begin{aligned} motionScore &= diff(m1, m2) * w_m \\ &+ (vel_{opt} - vel_m) * w_{vel} \\ &+ ||(loc_{opt} - loc_m)|| * w_{loc} \\ &+ (\theta_{opt} - \theta_m) * w_\theta \end{aligned} \tag{3.1}$$

$m1$ and $m2$ are the previous and next motions respectively. w_m is the weight that the difference in transition clips has on the overall score of that specific motion. vel_{opt} is the optimal or desired velocity, vel_m is the new motion's overall velocity and w_{vel} is the weight that the velocity has on the score of the motion. loc_{opt} is the desired end location, loc_m is the location that the new motion will end at and w_{loc} is the weight that the difference in desired location and actual location has on the score for the new motion. θ_{opt} is the desired orientation of the agent, θ_m is the final orientation of the agent after completing the new motion clip and w_θ is the weight that the orientation of the agent has on the score of the new motion. The choice function scores every possible motion segment using Equation 3.1 and then orders the motions from lowest score to highest score. The lowest scoring motion is the best possible motion that can be used by that particular agent for that particular one second interval. In addition to using the best velocities, locations and orientations, we also want to make sure that joint movements flow smoothly through the transition frames. We define diff(m1,m2) to assist in this as follows:

$$diff(m1, m2) = \sum_{i=0}^{10} \sum_{j=0}^{n} (m1_i^j - m2_i^j) \tag{3.2}$$

For n joints in a motion capture skeleton $m1_i^j$ represents the position of joint j at frame i of the transition for the previous motion and $m2_i^j$ represents the the same joint and frame for the new motion.

Including Equation 3.2 as part of the motion scoring function allows the agent to put a preference on segments that will create better looking transitions because it will reward motions that are closer to the previous motion for the transition frames. This alone will not guarantee that the best motion will always be the motion that has a fairly small transition difference. We also create a threshold for the value returned by the transition function. We use the following for our threshold values:

$$Include \quad Motion = \begin{cases} True & if \quad diff(m1, m2) < \delta \\ False & otherwise \end{cases} \tag{3.3}$$

δ is the threshold of the accumulated difference of the ten transition frames. It has an impact on which motion segments are available to the agent and it is user selectable. When δ is a lower number the transitions will look perfect but the chosen movements will not necessarily move the agent in the direction towards its goal. A larger δ increases the amount of difference permitted between transition frames allowing more obvious sliding and unrealistic motion in the transition motion. For the purposes of this paper we found that a δ value of 50 units maintained realistic transitions while still allowing for agents to move through the world in a realistic fashion. δ represents the aggregate difference over the 10 transition frames of the two motions. While using a threshold of 50 did not produce perfect transitions the result may be good enough because we are willing to forgive small amounts of error if the overall motion looks smooth and realistic.

The choice function we have developed is capable of moving agents towards their goals while still avoiding most collisions. Unfortunately the choice function alone cannot guarantee collision free movement for the agents. This problem is also compounded by the fact that each agent only chooses a new movement every second. One second is a large amount of time for an agent to move without adjusting its velocity so as to not collide with obstacles in the scene. Our system overcomes these collisions through our hybrid obstacle avoidance algorithm called Cell Marking, which we discuss in chapter 4.

Chapter 4

Cell Marking

The use of the choice function, as outlined in Chapter 3, gives us a mechanism for moving agents toward their goal. However, that movement is not guaranteed to be collision free. In order to ensure that objects move to their destinations without collisions, we have developed a new technique called Cell Marking.

Cell Marking is a hybrid obstacle avoidance algorithm that uses a rule-based algorithm, such as RVO and a Cellular Automata algorithm to guarantee that no collisions can occur in a crowd simulation.

4.1 The Grid

The underlying cellular automata structure is important for being able to move the agents around the world collision free. The information that the grid needs to store is which cells are occupied by each agent. When using a grid as an occupancy grid it is quite easy to determine which agents are occupying which cells. However, the agents are not all computing their next set of motion frames on every frame. Each agent will only compute this set of frames once every one second and all agents will not necessarily need to be thinking on the same frames. When an agent plans its next move it has to account for the entire movement an agent is currently making. If it does not take the agent's entire movement into account it will allow for collisions between agents. Our system overcomes this by storing more than just the current frame in a grid cell.

The grid cells hold valuable information about which agents will occupy that cell. To be able to make collision free movement the grid cells need to remember which agents will occupy that grid cell for the full time any agent will be planning for. We have determined that to guarantee collision free movement agents must plan movement for twice the amount of time they wish to move. For our particular system we are using one second of movement as the movement time. This means that each agent is planning two seconds into the future. The reason behind this is explained in section 4.2.

Our grid cell data structure is broken up into three specific parts. The first part is information about occupancy for the cell. It keeps a list of boolean values, one for each frame, to define if the grid will be occupied for that frame or not. The second part of the grid cell is a list of which agents are occupying that cell on a specific frame. Lastly we store information about which frame is the current frame of the simulation.

For optimization of space the grid cells do not need to store every frame of the simulation. They only need to have information regarding the current frame and then two times the length of a movement of an agent. As was mentioned earlier we used one second for the length of movement for our agents and since we ran our simulations at 33 frames a second we store 66 frames in our grid cells. The 10 seconds of transition frames are apart of that 33 seconds and the next 33 seconds. That means that each of the lists in the grid cells are only 66 units long and the current frame is then a value between 0 and 66 for any given frame. The current frame is rolled over to 0 again when the frame after the 66th frame is reached. This gives the benefit that regardless of which frame we are on in the grid cell, an agent can treat the current frame as frame 0. On each frame of the simulation these grid cells are updated and the current cells are marked as unoccupied opening up that cell.

Adding the time element to the grid cell allows an agent to occupy a cell in the first few frames of its movement and then open up that cell for other agents to occupy later in that movement. This maintains the collision constraint and guarantees only one agent can occupy one cell per frame. This will allow the agents to plan the best route possible in a

large crowd or when multiple agents are next to each other. The grid is updated each time step and the current frame in the occupancy grid is updated to represent that there are no occupying agents in those cells. Leaving these cells unoccupied allows the next planning phase of the agents to occupy cells that have not yet been occupied by any agents.

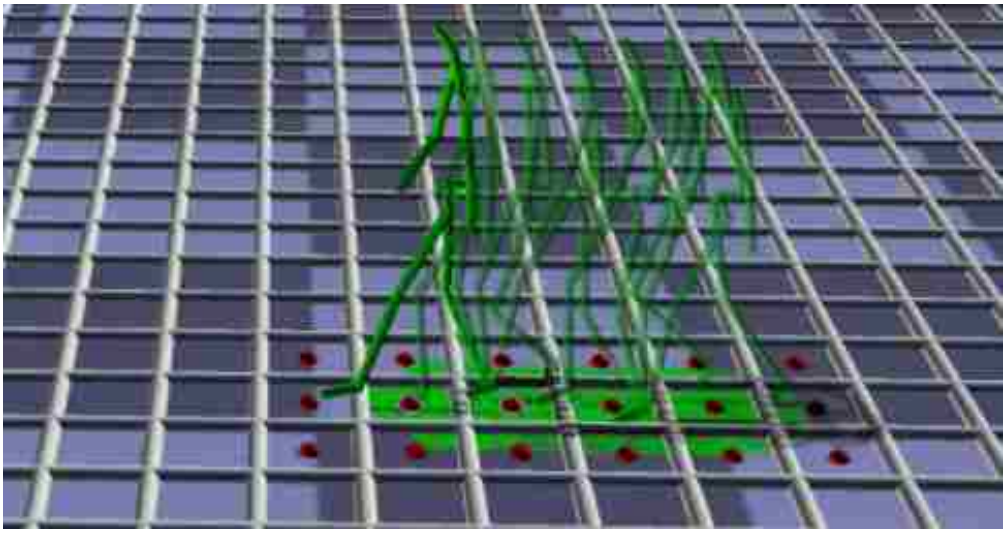


Figure 4.1: An agent advancing through its chosen movement. The small spheres in the grid cells represent which cells on the occupancy grid the agent will occupy during the clip.

We found that using a full grid for the movement of the agents was very costly and the simulation could only simulate movement for a small number of agents before it was no longer at interactive frame rates. This is because of the $O(n^2)$ time complexity of using a two dimensional array with n being the number of cells in the width of the grid. We found that with the size of grid we needed to simulate some of our bigger simulations we had a very sparse grid. That means that each frame we would update every single grid cell in our two dimensional array even though most of those cells never had any agents in them during that second of the simulation. To remedy this we used the Hash Map data structure to represent our grid. This allowed us to reduce our time complexity of the grid update algorithm to $O(m)$ where m is the number of occupied cells in the grid. To run the simulation m would have to be much smaller than n^2 otherwise the agents would have no place to move within the grid. Thus, we only update grid cells that are guaranteed to be occupied by agents on a

given time frame. If the cell is found to be unoccupied in the foreseeable future it is removed from the table.

Another added benefit of using the Hash Map representation of our grid was that we did not have to define a grid size. One problem we were running into while the agents were moving was that they were constrained by the edges of the grid. If they got close to the edges we ran into poor behavior and in some cases had agents trying to move outside of the grid causing memory issues. Using the Hash Map allowed us to create an unbounded grid. The Hash Map is capable of representing any point in space without setting boundaries ahead of time allowing our agents to move freely within the space without fear of them brushing up against their boundaries.

4.2 The Algorithm

The algorithm we use for picking the next motion segment an agent will use is described in the following section. Figure 4.2 visually demonstrates how that process is done.

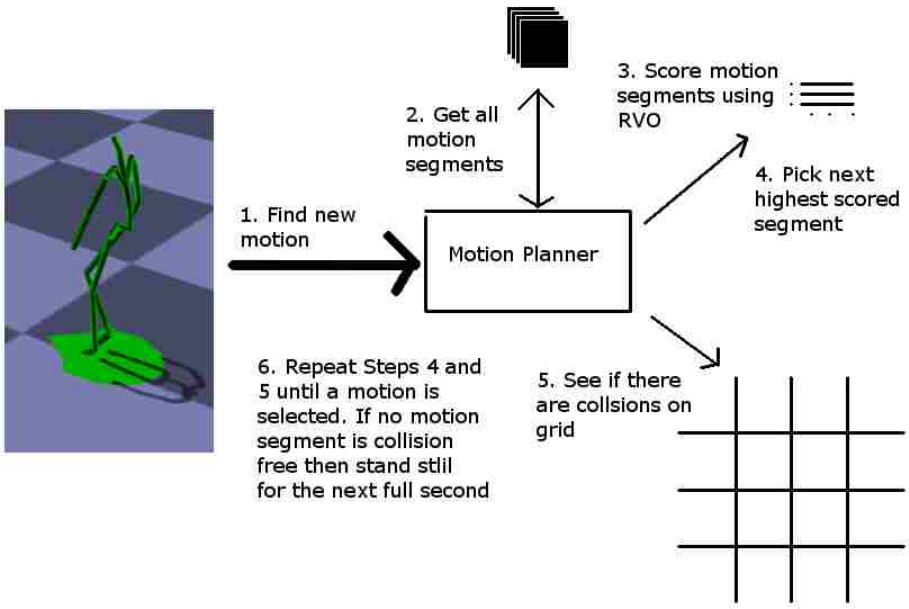


Figure 4.2: The process for choosing a new motion for the next full second.

Our simulation uses RVO to score all of the possible velocities for the next time step with velocities resulting in a collision scoring low and velocities resulting in collision free movement scoring high. After we get all of the scored velocities from RVO, we then find the velocity from RVO that is our optimal velocity. The scoring function will then score all of our available motion capture segments as described.

Once the motion capture segments have been scored appropriately, the agent that is currently planning will start with the motion segment that scores the highest. This is the motion segment that will get the agent closest to his desired goal. There is no guarantee that this movement will result in collision free motion however. The agent is constrained by real-world data obtained through motion capture data or through plausible transitions given the current rate of movement. The desired velocity returned by the obstacle avoidance algorithm may represent a movement or motion that is impossible to obtain given the motion capture information. This means that the motion segment that scored the highest will not necessarily match the desired velocity perfectly. It will represent the best effort that this particular agent can make in getting to that particular velocity.

The grid becomes useful because we know that our motion segment data is discretized into the number of frames that are represented in one second of the motion segment. To obtain the movement that this particular agent should make this time step the agent starts with the highest scoring motion capture segment. It runs the agent through the segment and checks the grid at each frame to see if there is a potential collision between two agents on any given frame on any given tile on the grid. If the agent is able to plan a route on the grid for the time frame without hitting a square that is occupied for the specific frame he is checking then he flags that motion as his chosen motion and marks the cells in the grid for the given frames that he will be occupying. The next agent then plans his move.

When an agent finds that the chosen motion segment will collide with an obstacle or agent in the simulation it will discard that motion segment and test the next highest

scored motion segment in the scored motion segments list. The agent proceeds to go through motions in the scored list until a motion is found that will fulfill a collision free path.

There are times when an agent will pass through every single motion in the scored motion list and not find a suitable motion to keep it from colliding with other agents. To maintain collision free motion some extra planning has to happen when an agent is choosing its motion. The assumption is that each agent can at least stop moving if no motion is suitable and still be collision free. This is done in the planning step for the agent. Instead of only planning the next second of motion, the agent also plans into its motion an extra second of motion. The extra motion is the motion of the agent transitioning to a stop from its current motion. This extra motion is not a guarantee that the agent will stop but gives the agent a motion it may execute in the case that no other motion was suitable for the given state of the environment. When agents are planning their new motions they take into account the extra motion of each agent to guarantee that they will not collide with that particular agent. In the case that an agent is able to find a valid motion it will overwrite the frames in the grid of the previously planned second of motion and continue on with the simulation.

Once the agent has chosen a motion it now has a full second of motion to execute and will execute that full motion without thinking again until the end. When the agent hits the end of the first second of motion it planned it runs the algorithm all over again. In the case that it finds that there are no suitable motions to make it still has a full second of backup motion it can execute to stop moving. In the case that the best motion for the agent is stopping it will plan a full two seconds again of the agent not moving. It will continue to choose stopped motion until a suitable motion frees up in subsequent frames.

Chapter 5

Results

The goal of this work is to create a crowd simulation that maintains the overall crowd realism of current crowd simulation systems while giving the individual agents within the crowd realistic movements and actions.

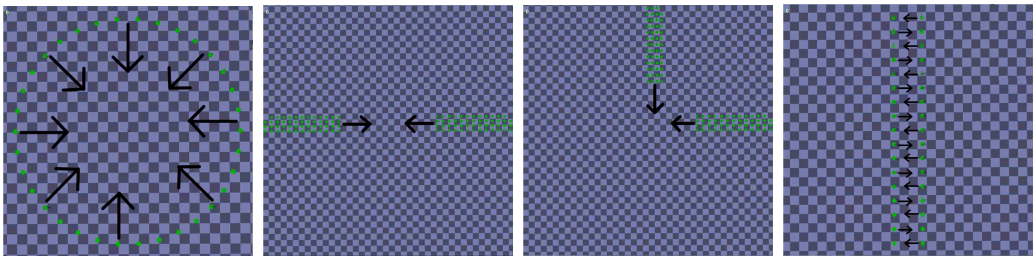
We performed a user study to further validate the results found in our research. The user study was done with 31 participants. Each participant was shown three videos. The first two videos were a closeup of a crowd simulation system using RVO and the second of our own system. We altered the ordering of the first two videos so that roughly half of the participants saw the video of our crowd simulation first, while the other half viewed the RVO crowd simulation video first. The participants were then asked to rate which of the two simulations had a more realistic appearance. Participants were asked to rate the simulations based on overall feel of the agent's movements as well as how the agents' movements looked with regards to their feet and interactions. The third video was a zoomed out view of the crowd. This video was a video of our current system and the participants were then asked if they thought that the visible crowd movement was plausible or not.

We found that 23 out of 31 participants said that our system produced more realistic individual movements than the crowds produced using RVO with motion capture applied post process. Seven said that the crowds produced using RVO were more realistic, and one person said that neither one was better than the other.

On the third video 27 out of the 31 participants said that the overall crowd movement produced by our system was plausible and four of the participants said the results were not plausible.

5.1 Realistic Crowd Motion

We use four different scenarios to test our crowd's ability to adapt to different situations. The four scenarios we use are similar to the examples used in [15]. We compare results from our crowds with the crowds produced using RVO. The scenarios we use are shown in figure 5.1 and are described below.



(a) Circle Scenario (b) Group Swap Scenario (c) Group Cross Scenario (d) Line Swap Scenario

Figure 5.1: Testbed scenarios for our crowd simulation system.

Circle figure 5.1a: In the circle scenario we line the agents up in a circle all facing the center of the circle. The agents are then given goals on the opposite side of the circle from their current location. This specific example tests how well agents will be able to interact with each other when they are all converging onto the same point. The circle scenario is an excellent testbed for a large number of characters coming together in close proximity. The goal for this scenario is to have all agents make it to their destinations without collisions, and with minimal stalling. At the center point of the circle, congestion can be a huge issue. Handling the congestion is what makes this particular problem interesting.

Group Swap figure 5.1b: The group swap scenario has two groups of agents facing each other. The agents in each group are given locations that match the agents in the group

opposite them. This particular scenario challenges how the agents will handle confrontation from another group when there are others around them heading in the same direction.

Group Crossing figure 5.1c: The group crossing scenario similarly has two groups of agents in two different locations. Instead of being placed opposite from each other the groups are now placed perpendicular to each other. They are then given goals directly in front of them that will require them to move through the center of the scene. Both groups will meet in the middle and try to figure out how to get around each other while remaining true to their desired goal.

Line Swap figure 5.1d: The line swap scenario has two parallel lines of agents facing each other. Each agent is given a goal that is occupied by the corresponding agent in the line across from him. The agents attempt to pass each other with as little deviation as possible while still maintaining realistic interactions.

These four examples display the ability of our system to maintain realistic crowd motion with respect to current crowd simulation systems. We show how this system compares to the RVO system in Table 5.1.

Table 5.1: Comparison between RVO and CellMarking. We ran a simulation with 100 agents in a crowd for both algorithms. The times represent the amount of time in seconds in the simulation world that it took from start until every agent in the simulation had reached its goal.

	Circle	Group Swap	Group Crossing	Line Swap
RVO	50.5s	96.62s	88.12s	16.97s
CM	50.8s	121.94s	94.12s	22.29s

In the Group Swap scenario, the RVO approach allows for the agents to reach their goals in less time than the Cell Marking approach. In the other three scenarios, times are similar, with the RVO approach taking somewhat less time than the Cell Marking approach. While lower time to reach the goal state is a desired property, it is not the only one. Cell Marking is guaranteed to be collision-free, whereas RVO is not. Thus, while agents may make it to their goal in a shorter amount of time, they may have to collide with other agents in order to do so.

RVO is guaranteed to be collision free as long as agents pick velocities outside of the velocity obstacle produced. There are cases though where it is impossible to choose a velocity outside of the velocity obstacle for the agent. In this case the agent chooses the velocity that will penalize him the least or make his collisions minimal. The penalty function used is the same used in [23] and is shown in Equation 5.1.

$$penalty_i(v_i') = w_i \frac{1}{tc_i(v_i')} + ||v_i^p ref - v_i'|| \tag{5.1}$$

Where v_i' is the new velocity, w_i is an aggressiveness factor that can be different for each individual, $tc_i(v_i')$ is the expected time to collision and $v_i^p ref$ is the desired velocity.

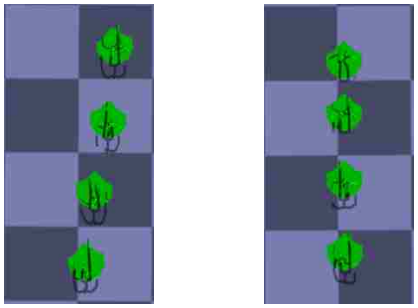
Table 5.2: Comparison between RVO and CellMarking with regards to collisions. We ran a simulation with 100 agents in a crowd for both algorithms. To determine the score for this we take the average number of agents colliding per frame. Then we average that over the number of frames. A score of 1 would mean that every single agent was colliding with at least one other agent for every single frame of the simulation. The scenarios that required closer interactions with agents had larger values.

	Circle	Group Swap	Group Crossing	Line Swap
RVO	0.0048	0.0184	0.0335	0.0020
CM	0	0	0	0

Cell marking does not allow collisions to occur. If the agent realizes there will be a collision present he will either stop or choose another route. Table 5.2 shows the rate of collision for each algorithm in each scenario. The end result is that agents in congested areas spread out more and thus have longer paths to follow. In the case of the circle this is not as evident because the velocity obstacles push the agents around the outside of the circle like a vortex. The end result is a very quick solution to reach their goal. In the case of the Groups the agents converge on each other at a central mixing point. They tend to clump and push to the outside. The result is that it takes longer for the agents to maneuver through the crowd. However, in the end they are able to get through and reach their goals collision free.

5.2 Realistic Individual Motion

The results of the previous section show that our technique gives comparable run times as RVO in many cases, and in addition guarantees collision free movement. The real focus of this research, however, is on achieving realistic movement from the individual agents. We have shown that our system produces realistic crowd formations. We now show that the individuals within our crowds make more realistic movements than current crowd simulation systems.



(a) (b)

Figure 5.2: Individual motion in our system (a) vs. individual motion in RVO (b). In image (a) an agent walks along a path using motion capture data to determine where he will end up. Notice that the agent does not make a perfectly straight line. This is because the motion capture data used has a slight pull to the right. In image (b) an agent plans his path and then uses motion capture data over the top. The agent does not deviate from the straight path even though the over laying motion capture has the agent slowly pulling to the right. This causes unnatural movement in agent (b). This is because although agent (a) is not necessarily walking in a straight line the agent is moving as the underlying motion dictates. Agent (b) appears to be sliding as he is walking since he ignores the underlying movement.

Two specific areas of comparison between our system and current systems are in the reduction of footskate and on turns. Footskate occurs when an agent's foot movement does not map well to the overall movement of the agent's body. In current systems footskate will happen at any point in the agent's movement but is most pronounced when the agent is turning. In many systems the agent will continue to walk forward while turning on a point on the ground. Some systems try to hide the footskate by having the agent move in an arc. The use of motion capture data for our system reduces footskate. Since the data

being used is actual real-world captured data, the feet will move in such a way across the ground that realistic motion is achievable. Transitioning between two pieces of real world data reintroduces some footskate but we have found that we can minimize the introduced noise through judicious selection of parameters to the choice function.

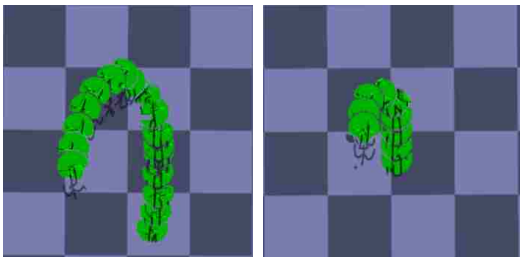
We learned that you can produce transitions that appear perfect if you bring the allowed threshold between difference in motions very low. Unfortunately the agents would choose motions that did not get them closer to their goals. This compromised the overall crowd motion and gave the individuals that appearance that they were confused and had no real goal. Raising that threshold higher than our tuned value had the effect of introducing transitions that were very unrealistic. We also found that the overall crowd motion was not any better either because of the collision free constraint enforced by the simulation.

The velocity and location parameters from the choice function would have the expected results. The more important you make those the more likely the agent is going to choose motions that get them to their goals quickly. The desired orientation from the choice function had the benefit of determining how likely your agents were to choose motions that oriented them towards their goals. We had a couple of motions in our motion capture library that allowed an agent to move in a direction other than the direction they were facing. If the orientation parameter was not set very high then agent's would sometimes choose motions that moved them sideways instead of turning towards their goals.

Our tuned parameters maintain realistic transitions with agents that turn and move toward their goal. They also allow the agent the freedom to sidestep around an obstacle to be able to maneuver quicker around it to get to its goal.

Figure 5.2 demonstrates some differences that arise between our approach and typical crowd simulation systems. The agents are planning a path along a straight line. Current systems will determine the straight line path and move the agent along that line represented in figure 5.2b. They will then apply the animation over the top. In this case the animation forces the character to lean to the right. This motion did not show up in the path for the

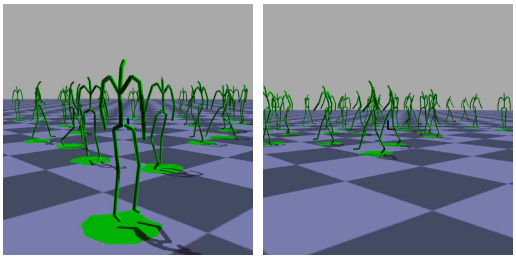
agent. Instead the agent continues in a straight line and sliding occurs at the feet because the agent does not move in the direction of the motion capture data used for the animation. Anytime there is sliding present the reality of the simulation will be reduced. Our system is represented in figure 5.2a. The agent is constrained to move in the way that the motion capture data allows. Ideally the agent would choose motion capture data that would keep it moving in a straight line but there is no motion capture data that will allow that. As a result the agent deviates from a straight line and corrects its trajectory when it realizes that it will not hit the goal if it continues on that course. Our approach produces motion that is more true to life than other current systems.



(a) (b)

Figure 5.3: Difference in agent turning between our system and typical systems. In our system (a) an agent makes a turn using motion capture data to determine where he will end up. In a typical system (b) an agent plans his turn and then uses motion capture data of walking forward over the top, producing a quicker, but less realistic turn.

Another obvious visual difference between our system and current systems is the individual agent's turns. Unlike other systems, our solution does not apply animation post process to the agent after the velocity and orientation of the agent have been chosen. The agents choose their velocity and orientation based on the motion capture that is available. This means that making a turn is not as simple as rotating around a point and heading to the destination. When an agent needs to turn it must find a movement in the motion capture data that supports the turn it wishes to make. The result is that the agent makes a realistic, albeit non uniform, turn and is able to make a variety of different kinds of turns each time as shown in figure 5.3.



(a) (b)

Figure 5.4: A close up of our crowd system in (a). A close up of current crowd systems in (b). The uniformity in current systems gives and unnatural look.

An undesirable characteristic of current systems that is overcome by our system is the fact that all the agents in the system make the same or similar movements every frame. The result might be compared to synchronized walking. Our system has the benefit of having a variety of different movements available and because of the cost function involved, each agent will have different needs and will therefore choose different types of movements based on those needs. The result is that many different types of turns are possible. The agents are also able to make quick pivot turns, side steps and any other type of turning motion that is represented in the motion capture dataset. In the case of a side step our agents are able to move in a direction that they are not facing. Current systems are unable to achieve natural, but varied, turning motion in their characters. Figure 5.4 is a close up of two crowds. Figure 5.4a shows a crowd in our system. The agents are facing different directions and are in the middle of different types of movements. Figure 5.4b shows results given by current systems. Each agent is in the middle of the same movement, and agents close together appear to be moving in a coordinated fashion. This is due to the unnatural and uniform motion present in current systems.

Chapter 6

Discussion and Future Work

Two major difficulties arise in planning and simulating crowds. The first is the integrity of the movement of the individual within a crowd while achieving realistic crowd motion. The second is being able to generate these crowds within a reasonable framerate. Artists and animators need near real time motion planning in order to accurately direct the crowds. Current crowd systems have addressed the problem of creating realistic group crowd motion in a reasonable amount of time. They have failed to address the individual motion of the agents within the crowd. This becomes obvious as you come closer to an individual within the crowd and can see how they move along the ground. Current systems typically ignore this problem in the planning phase but come up with ways to address it in the post processes. Our solution improves upon the methods that have previously been used. We do so by incorporating into the planning step the individual motion of the agents within the crowd. We constrain the individuals to only be able to make movements based on the motion capture data provided instead of making unrealistic human movement. The planning is built upon the work of current systems. This way we are able to capture the group motion that current systems already do a good job of, but then augment that group motion with realistic individual motions. We are also able to do this in interactive time.

There is additional work that could improve this system even further. The most obvious area of future work would be increasing the number of agents that this simulation can handle. The largest number of agents we used to make comparisons between the two algorithms was 102 agents. We did increase the number of agents used when we ran our speed

tests, however. Some work could be done to make this algorithm faster to allow for more agents. One area of optimization we see is in the obstacle avoidance algorithm. Currently the agents are using a grid to avoid obstacles. If the underlying grid could be removed it could potentially increase the number of agents that could be run in the simulation as well as achieve different movements from the agents.

Another area of future study is in the transitions between movements. The solution used for this work was very simplistic and there are other more complex algorithms that could potentially produce better results. An example would be the work shown in [12]. A solution for transitions that can be done in real time and reduce footskate would increase the overall realism of this simulation.

For our research we took a strictly collision free approach to crowd simulation. Some of the current systems allow for collisions within their models. It would be interesting work to find a balance between collision free movement and allowing collisions. Some possible benefits would be in how quickly the agents are able to reach their goals. Also some of the more abrupt stops manifest in this algorithm may be diminished if agents were allowed to choose movements that resulted in minor collisions.

Lastly the resolution of the underlying grid for collision avoidance could be increased. Bigger grid cells mean that agents will take longer routes to try and avoid each other. Reducing the grid resolution while maintaining simulation speeds should provide an environment in which the agents could potentially have more realistic movements.

Chapter 7

Conclusion

Our system builds upon current crowd simulation systems and uses their techniques for forming realistic crowd movement. We then improve upon their weaknesses. The current systems do not use real world data in the agent planning step of their algorithm. This limits the ability of those systems to achieve a totally realistic crowd because the agents within the crowd are able to move in ways that are unrealistic and are not representative of real people. In the first part of our work we introduce a choice function that helps determine which motion capture data will best fit the agents' current goals. This function is adjustable so that it can be tuned for the desired application. Lastly we introduce a new hybrid obstacle avoidance algorithm called Cell Marking. This allows the agents to move along collision free paths while choosing the best motion capture data for their current situation. We were able to show that the crowds produced by our simulation were realistic and that the individual's movements and choices also produce realistic and plausible motion.

References

[1] Victor J Blue and Jeffrey L Adler. Cellular automata microsimulation for modeling bi-directional pedestrian walkways. *Transportation Research Part B: Methodological*, 35(3):293–312, 2001.

[2] Adriana Braun, Soraia R Musse, Luiz PL de Oliveira, and Bardo EJ Bodmann. Modeling individual behaviors in crowd simulation. In *16th International Conference on Computer Animation and Social Agents, 2003.*, pages 143–148. IEEE, 2003.

[3] Carsten Burstedde, Kai Klauck, Andreas Schadschneider, and Johannes Zittartz. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A: Statistical Mechanics and its Applications*, 295(3):507–525, 2001.

[4] S. Chenney. Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242. Eurographics Association, 2004.

[5] N. Courty and T. Corpetti. Crowd motion capture. *Computer Animation and Virtual Worlds*, 18(4-5):361–370, 2007.

[6] C. Egbert, P. Egbert, and B. Morse. Real-time motion transition by example. *Articulated Motion and Deformable Objects*, pages 138–147, 2010.

[7] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *IEEE International Conference on Robotics and Automation, 1993.*, pages 560–565. IEEE, 1993.

[8] Chiara Fulgenzi, Anne Spalanzani, and Christian Laugier. Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *2007 IEEE International Conference on Robotics and Automation*, pages 1610–1616. IEEE, 2007.

[9] Tomoki Hamagami and Hironori Hirata. Method of crowd simulation by using multiagent on cellular automata. In *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003.*, pages 46–52. IEEE, 2003.

[10] D. Helbing and P. Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[11] Sujeong Kim, Stephen J Guy, Dinesh Manocha, and Ming C Lin. Interactive simulation of dynamic crowd behaviors using general adaptation syndrome theory. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 55–62. ACM, 2012.

[12] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *ACM SIGGRAPH 2008 classes*, page 51. ACM, 2008.

[13] K.H. Lee, M.G. Choi, Q. Hong, and J. Lee. Group behavior from video: A data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 109–118. Eurographics Association, 2007.

[14] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. In *Computer Graphics Forum*, volume 26, pages 655–664. Wiley Online Library, 2007.

[15] Jan Ondřej, Julien Pettré, Anne-Hélène Olivier, and Stéphane Donikian. A synthetic-vision based steering approach for crowd simulation. *ACM Transactions on Graphics (TOG)*, 29(4):123, 2010.

[16] N. Pelechano, JM Allbeck, and NI Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 99–108. Eurographics Association, 2007.

[17] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

[18] C.W. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference. http://www.red3d.com/cwr/steer/gdc99*, 1999.

[19] Brian C Ricks and Parris K Egbert. More realistic, flexible, and expressive social crowds using transactional analysis. *The Visual Computer*, 28(6-8):889–898, 2012.

[20] Andreas Schadschneider. Cellular automaton approach to pedestrian dynamics-theory. *arXiv preprint cond-mat/0112117*, 2001.

[21] Franco Tecchia, Céline Loscos, R Conroy-Dalton, and YL Chrysanthou. Agent behaviour simulator (ABS): A platform for urban behaviour development. 2001.

[22] Carnegie Mellon University. CMU Graphics Lab Motion Capture Database. In *http://www.mocap.cs.cmu.edu*.

[23] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008.*, pages 1928–1935. IEEE, 2008.

[24] M. Xiong, M. Lees, W. Cai, S. Zhou, and M.Y.H. Low. A Rule-Based Motion Planning for Crowd Simulation. In *International Conference on CyberWorlds, 2009. CW'09.*, pages 88–95. IEEE, 2009.