A HIGH PERFORMANCE FRAMEWORK FOR AGENT BASED PEDESTRIAN DYNAMICS ON GPU HARDWARE

Paul Richmond
Dr Daniela M. Romano
Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello
Sheffield, UK
Web: http://www.dcs.shef.ac.uk/~paul
E-mail: paul@dcs.shef.ac.uk

KEYWORDS
Graphics, Parallel methods, Interactive simulation, Real-time simulation, Scientific visualization.

ABSTRACT

Pedestrian simulations have recently focused on top-down implementations ignoring more computationally intensive agent based dynamics. This paper presents a framework for agent based modelling on the Graphics Processing Unit (GPU) which demonstrates large scale pedestrian simulation and rendering. GPU hardware offers significant performance for dynamic large crowd simulations, however the process of mapping computational tasks to the GPU is not trivial and expert knowledge is required. An agent based specification technique is therefore presented, which allows the underlying GPU data storage and agent communication to be hidden. The framework allows the use of static maps to set static environment obstacles and a zoning technique is described for route planning. Parallel population feedback routines are also used to implement Level of Detail (LOD) rendering which avoids any costly CPU data read-back.
.

INTRODUCTION

Agent based dynamics are particularly well suited to replicating pedestrian and general crowd behaviours. The individual style of modelling closely resembles the cognitive process of decision making and allows complex interactions to emerge from specifying a number of simple rules. Whilst earlier work (Reynolds 1999) has focused on the specification of such rules in agent based environments, more recent work uses non agent based approximation techniques (Treuille et al. 2006, Courty and Musse 2005) to avoid direct evaluation of inter-agent interactions. Such techniques are computationally cheaper and often map to Graphics Processing Unit (GPU) hardware offering significant performance improvements. This has become particularly important as urban environment visualisation has scaled in accordance to GPU hardware requiring far larger population sizes at interactive rates.

The trend towards utilising the GPU for dynamics simulation is well documented (Harris et al. 2002). The parallel nature of GPU processing offers a massive speed up opportunity for algorithms, which are written with parallelism in mind. In the case of large scale real-time pedestrian dynamics, where populations can reach tens of thousands, utilisation of the GPU is almost a requirement. Without maintaining positional data of pedestrians in GPU memory, the transfer from system memory quickly becomes a significant bottleneck and limiting factor. GPU based alternatives have the significant advantage of maintaining data where it is required for rendering, eliminating this problem entirely.

Despite the obvious performance advantages of harnessing the power of the GPU, a significant disadvantage is the difficulty of mapping computational tasks to the GPUs architecture. Whilst CPU based programming allows simultaneous read write access to DRAM, the data flow of the GPU is more closely resembled by a stream processor. Traditionally this has been made available to programmers though varying programmable stages (the vertex, geometry and pixel stages). Those that use the GPU for general purpose computation, or General Purpose computation on GPUs (GPGPU), have used these programmable stages within a traditional graphics environment by placing data into textures and using the fragment processor to invoke a quad of parallel processes. More recently the introduction of NVIDIA CUDA and unified programmable stage processors has allowed more direct access to graphics hardware, removing the requirement to deal directly with textures and graphics primitives. Whilst this is advantageous the lack of standardisation between hardware vendors makes traditional graphics GPGPU the most flexible option. Further more expert knowledge of the underlying hardware is required in either case to achieve the best performance results.

This paper presents an agent based framework for pedestrian modelling on the GPU using graphics based GPGPU. Its primary goals are to hide the underlying GPU data storage and agent communication, allowing user focus on agent based pedestrian scripting. It builds upon previous work (Richmond and Romano 2008) allowing C like scripting to be used for agent behaviour with a complementary Object Orientated API for setting up initial conditions and initial agent data. Static maps can be set allowing agent interaction with obstacles which are also used to demonstrate route planning through the environment. Pedestrian animation and rendering is implemented through a key framed animation technique with a level of detail system which is maintained entirely on the GPU. Whilst the agent based pedestrian dynamics used in this paper are based on simple but well established work (described in the following section), it is advised that the described framework is suitable for inclusion of more advanced cognitive and emotional models (Romano et al. 2005).

AGENT BASED PEDESTRIAN DYNAMICS

Agent Base Modelling (ABM) of dynamic systems gained popularity through Craig Reynolds who, in 1987 demonstrated flocking bird behaviour through an agent based Boids model (Reynolds 1987). Unlike dynamic systems controlled through sets of partial differential equations, ABM examines the low level individual behaviours which when part of a social system result in emergence of non predictable social events. In the case of the Boids model the low level behaviour can be described by three simple rules; separation at close range, attraction to the perceived flock centre and velocity matching of perceived neighbours. This concept is extended in Reynolds later work (Reynolds 1999) to include advanced behaviours such as pursuit, evasion, obstacle avoidance and flow field following for general agent locomotion.

Helbing (Helbing et al 2002) introduced the social forces model as a class of equations which balance various environmental and social influences on pedestrians in both normal and panic situations. Essentially this builds upon previous work (Helbing and Molnar 1995) which mathematically formalises the concept of social forces as a behavioural description of conflict situations. Most important in this model is the description of a repulsive social force acting between pedestrians. In its simplest form, this force acts as a mechanism for collision avoidance giving greater precedence to situations in font of the pedestrian, over situations occurring behind. Additionally longer range attractive environment forces such as attraction to window displays, special attractions or acquaintances are modelled using a force summation which are combined with the social repulsive force. In emergency situations an additional short range body force and sliding friction force is described which avoids compression in high density populations. Whilst simplistic with respect to the dynamical rules, complex natural pedestrian activity can be observed in Helbings (Helbing and Molnar 1995) work. Phenomenon such as lane formation and intersection dynamics are just two of the emergent behaviours empirically observed in real world conditions.

Building upon Helbings (Helbing and Molnar 1995) work, whilst considering large population sizes, Quin et al. (Quin et al. 2003) demonstrated a technique which used a SWARM cluster of 10 processors to simulate 10,000 evacuating pedestrians using the social forces method. Much attention in this work is given to a spatial partitioning technique which distributes pedestrians across the processing node with communication between them handled with a Message Passing Interface (MPI) library. This technique, as also demonstrated more generically by the Flexible Large-scale Agent Modelling Environment (FLAME) (Adra et al 2008), is one which inspires this work. The use of spatial partitions is essential in producing large simulations and scalable algorithms, particularly with respect to pedestrians where there is no need for total communication between all agents.

Coutry and Musse (Courty and Musse 2004) present a GPU implementation of pedestrian dynamics which is inspired by Helbings social forces model (Helbing and Molnar 1995). Its uses a cellular system which allows agents to move between empty cells in discreetly partitioned space. As each cell occupies at most a single agent the social repulsive force is calculated on the fly during the update stage by considering a neighbourhood of surrounding cells. Improving upon this by avoiding the potential of agents occupying the same grid space, Courty and Musse (Courty and Musse 2005) propose an implementation of a dynamically created force field texture. This technique uses GPU hardware blending to compute the sum of all forces for each discrete partition. The result is rendered to an off-screen texture, the force field texture, and is then used during the update stage to avoid direct communication between pedestrians is the system. Similarly to this Treuille et al. (Treuille et al. 2006) uses the same technique to construct a density field of pedestrians which is in turn used to determine a more complex discomfort field. The discomfort field is constructed in real time by considering local collision avoidance and global path planning. Performance of this technique is dependant on the resolution of the discomfort grid. With a grid of 60^2 Treuille (Treuille et al. 2006) reported performance of 10,000 pedestrians 5fps before rendering in-between frames.

As recent pedestrian modelling has tended towards less agent based techniques, it is no surprise that recent high performance agent based simulations are concentrated on more general agent based and swarming applications. Reynolds more recent work on PSCrowd (Reynolds 2006), demonstrated through schools of fish, is analogous to pedestrian behaviour and presents a 2D environment performance of 15’000 low resolution (36 polygon) fish at 60fps. The technique used to implement this is based on the previous work (Reynolds 1999) and uses a spatially partitioned 3D or 2D environment. The high performance speed is achieved by utilising the Playstation3’s cell processor to batch a number of spatial buckets across to the cell processors 8 parallel Synergistic Processing Units (SPU’s). Unlike Reynolds earlier work, the neighbourhood heuristic in this case is based on N-nearest neighbour, with each agent maintaining its own neighbourhood list. As interaction is not based on a defined radius or limited vision, this technique is highly dependant on the size of the neighbourhood lists.

More general agent based work has recently been demonstrated by D'Souza et al. (D'Souza et al. 2007). This work concentrates on implementing a number of 2D agent based models on the GPU with particular attention to performance. Similar to the previously described methods, spatial partitioning is used however each partition is responsible for maintaining only a single agent. Although this likens the system to that of cellular one, agents have continuous values and collisions (multiple agents occupying the same space) are resolved using a multi pass priority system. Agent communication is achieved in the same fashion cellular automaton and performs an expanding search into neighbouring cells. Whilst this technique boasts a performance of up to 2 million simple agents in real time, it is important to understand the limitations of such a technique when applied to a pedestrian system. Firstly fine grained partitioning requires large amounts of space. If a larger partition size is used, space is saved with the trade-off that there will be more costly collisions to resolve. Secondly if agents are well distributed, particularly into local groups across a large area then large amounts of partitioned space are wasted by holding no useful data. Decreasing the partition size again helps to resolve this however as in the first case higher concentrated areas have to then resolve additional collisions.

AGENT BASED MODELLING ON THE GPU

As this work aims to present a useable framework for agent based pedestrian specification, rather than a single simulation example, the mapping of agent data to the GPU is a process which it is important to abstract. The idea behind this is that both agent scripting and the programming interface should be able to get and set agent data variables without explicit knowledge of the underlying GPU memory architecture. This is achieved through a translation function 'F' which provides a mapping for both agent update scripts and the getting and setting of initial agent data into a number of 2D stacked 32bit floating point textures (agent space). Similarly to previous work on particle systems (Latta 2004, Kipfer et al. 2004) a 1D list of variables is easily translated into 2D texture space with a 2D position *(i, j)* in each stacked texture '*t*' representing an individual set of data. As texture access is read/write only, '*2t*' textures are required in total with data being stored in up to all four of the red, green, blue and alpha colour channels respectfully (figure 1). Within OpenGL the Frame Buffer Object extension allows simultaneous writing to Multiple Render Targets (MRTs), on our implementation hardware (a NVIDIA GeForce 8800 card) up to 8 targets are supported giving a total of 32 agent variables. For a potential non communicating system of agents, simulation is achievable by performing *N* parallel operations by rendering a single quad primitive. Assuming the quad primitive is the same dimension as agent space and rendered from an orthogonal perspective, rasterisation will invoke a parallel operation for each of the *(i, j)* agent positions.

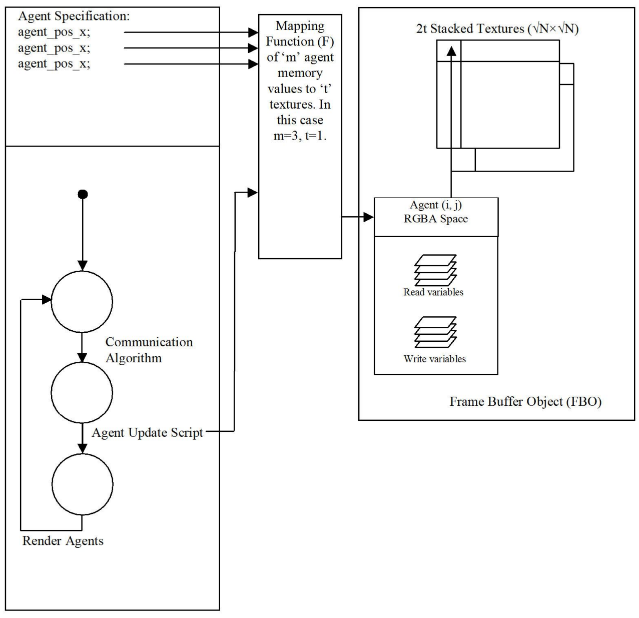


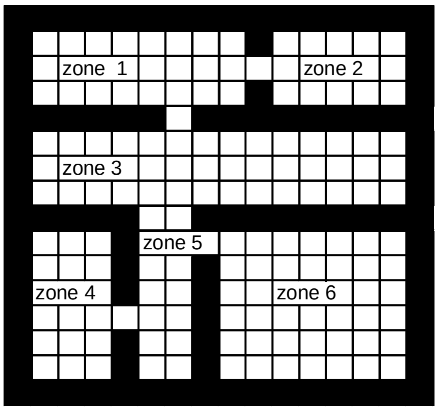
Figure 1 - Mapping of three agent variables to 2D Texture Space

In many of the previous examples partition space is used to either directly hold agent data or hold pointers to data in agent space. Similarly this work splits space into discrete spatial partitions however the partition size is equal to the agent's communication radius with each partition holding any number of agents. Agent communication can then be achieved by consulting all agents within an agents own and neighbouring partitions. On the CPU this task is trivial and each agent in a partition can be stored as a linked list. On the GPU this is significantly more difficult as dynamic storage sizes are unsuitable for parallel computation. Instead a dynamic structure is computed before each agent update and adapting techniques used in rigid body particles physics (Harada 2007, Green 2007) a matrix is calculated allowing each agent within each spatial partition to be located in agent space. The first stage of the technique involves sorting the agents in agent space depending on their location. As with particle based work (Kipfer et al 2004, Green 2007) each partition is assigned a unique spatial bin identifier based on their x and y position. Agents can easily determine their location identifier and are sorted using cache efficient bitonic sorting (Govindaraju et al. 2005) into a theoretical 1D (stored physically in 2D texture space) list according to this. A vertex scattering technique is then applied, where for each agent a vertex is drawn at the origin with texture coordinates allowing each of the agent sort values to be looked up from agent space. Each sort value is compared to that of the previous agent and if the sort value is the first in the sorted 1D list then a linear search is performed until the last sort value is also found. Assuming an off-screen buffer size with the same dimensions as partition space is used, the start and end positions of each partition identifier can be scattered by offsetting the vertices position with the start and end indices preserved as texture coordinates. Vertices which are processed and are found not to be boundary indices are simply positioned outside of the off-screen buffer and ignored. The end result of this is that each occupied partition space in the off screen matrix contains the start and end positions of all agents within it. This is then used in the same way as a CPU list to iterate through each agent testing for agents within the communication radius. Whilst not all agents tested will be suitable for communication this improves dramatically on the worst case O(n^2) and unlike nearest neighbour techniques guarantees communication for all agents within the set radius.

AGENT BEHAVIOUR SCRIPTING

Use of the agent based pedestrian API is dependant on an appropriately defined agent update script. Agent scripts are defined in a C like environment and must override a function, agentMain, accepting an agent and global variables structure as arguments and returning a single agent structure. As reference to variables in the agent structures are mapped at compile time with the previously described mapping function, agent update scripts are free to reference agent variables directly. With respect to communication, agent scripts use two placeholders `FOR_EACH_AGENT_A`, and `END_FOR_EACH` to hide the underlying algorithm. Further vision test can be applied to each agent if the desired behaviour requires this. For agent populations with no environmental influences this is all that is required to compute an agent's new internal variables and hence provide a convincing simulation. The globals argument to the main update function offers an extension to this by allowing global static variables to be set between each update stage. Single Float and 2D data array values can be set using the programming interfaces AgentPopulation class. In addition to this role the class also handles the initial setup, agent script compilation (to valid cg code), agent initialisation and stepping of the simulation loop.

Using the API to simply get and set global 2D data it is possible to encode environmental forces (Courty and Musse 2005) avoiding the costly computation of each agent wall combination as described in Helbings model (Helbing and Molnar 1995). As the granularity of this environmental force field texture is independent from agent interaction large grained force fields can be used, providing they have sufficient detail to capture the smallest static obstacle. 2D data arrays are stored in GPU memory as textures and as a result up to 4 values can be stored in each array. For the purposes of obstacle avoidance the first two components of the array (red and green) are enough to hold a directional velocity. In the remaining components of the array the examples in this paper store a height map value, used for rendering static geometry at a later stage and a non regular environment space identifier. This identifier has the purpose of zoning the environment to allow longer range path planning where pedestrian's behaviour is beyond that of a random walk. Figure 2 demonstrates an environment map with (black) static obstacles split into 6 zones. Two additional 2D data sets encode firstly a navigation lookup grid which indicates the next zone to move into from the current zone (vertical) to the desired long range goal zone (horizontal). Secondly an additional set of data is required to store an x and y (goal) point value for each of the zones. These are used as goal points which the agent will move towards. In environments where zones consist of complex obstacles with narrow gateways such as door ways, experimentation has shown that the doorway itself must use a zoned area. This step ensures that pedestrians intelligently pass through gateways avoiding a situation where there desired path is blocked.



	Zone 1	Zone 2	Zone 3	Zone 4	Zone 5	Zone 6
Zone 1	-	2	3	3	3	3
Zone 2	1	-	1	1	1	1
Zone 3	1	1	-	5	5	5
Zone 4	5	5	5	-	5	5
Zone 5	3	3	3	4	-	6
Zone 6	5	5	5	5	5	-

	Goal X	Gloa Y
Zone 1	*x*	*y*
Zone 2	*x*	*y*
Zone 3	*x*	*y*
Zone 4	*x*	*y*
Zone 5	*x*	*y*
Zone 6	*x*	*y*

Figure 2 – A simple illustrative zoned environment with corresponding data tables

PEDESTRIAN BEHAVIOUR

The pedestrian behaviour in this paper is influenced mainly by Reynolds work (Reynolds 1999) however as many factors draw strong parallels with Helbings social forces model (Helbing and Molnar 1995) the exact rules are somewhat of a hybrid. Formally the following equation (1) is able to describe the force exerted on each pedestrian during the update stage for all examples within this paper.

$$F_i = R_i + Cr_i + G_i + M_i \qquad (1)$$

The total force exerted on each agent F_i, is the result of a social repulsion force R_i, a close range interaction force Cr_i, a short term goal force G_i and an environmental force field force M_i. This force is then used as a directional steering force to make some change to the pedestrian internal velocity. This altered velocity is then checked to ensure it does not exceed the pedestrians maximum velocity and if required the velocity is normalised accordingly. The social repulsion force uses the same preference to events in the direct line of sight as in Helbings (Helbing and Molnar 1995) model. The symbol λ is simplified to represent a scalar value indicating the size of angle between the pedestrian i's line of sight and pedestrian j's position. Additionally as in Reynolds work (Reynolds 1999) agents are given a limited vision which filters agents outside their field of view. More formally the following equation (2) describes the force R_i where the letter j represents only agents from within the limited vision filter.

$$R_i = S\sum_{j}^{0} \lambda_{ij}\left[P_i - P_j\right]_i \left(\frac{I}{(|P_i - P_j|)^2}\right) \qquad (2)$$

The static value S indicates a scalar value controlling the influence of the social repulsive force. The positions P_i and P_j represent the positions of agent i and j respectfully and the distance between them represents the directional force vector between the two agents. This force is further scaled by the inverse square of the distance between the two agents. The value I is used to scale the effect of the inverse distance effect and in most examples has been tweaked depending on the interaction radius between agents. Unlike the social repulsive force, the force Cr_i is independent of the direction between agents. Its influence is over a far smaller radius are rarely has effect on pedestrians unless there is a high concentration in which it acts mainly as collision avoidance. The force G_i acts as an influence towards a specific goal point in the environment. In the case of a random walk the goal position is directly in front of the pedestrian encouraging them to follow their current path. In cases involving longer range goals, this goal value is found as a result of environmental lookups as described in the previous section.

PEDESTRIAN VISUALISATION

Pedestrian rendering is achieved through two very different methods. The first of these uses a primitive directional triangle. In this case each triangle primitive is rendered at the origin with multi texture coordinates reflecting an agent position in (i, j) agent space. A vertex shader then looks up agent positional and velocity data and translates and rotates the primate object accordingly. As an entire population represented as primitive objects contains relatively few OpenGL calls in total, the entire population can be stored in a single display list. For more advanced pedestrian representations where individual model sizes become much larger this technique quickly becomes unsuitable. In such cases it is necessary to store only a single model representation in a display list. The single display list is then called for each agent with the multi texture coordinate value set before the display list is called, allowing each instance of the model to be translated by a differing set of agent values.

As pedestrians do not move as static objects it is important to animate them with walking behaviour as they move around the environment. Key-framing provides an ideal animation technique as it is computationally cheap and is easily capable of representing simple human locomotion. Through experimentation it is evident that reasonable walking behaviour can be achieved through interpolation between only two key frames. For improved fidelity multiple key frames can be used however as all draw calls are stored in a single display list it is necessary to store the positional and normal information for each key frame model in the list. Whilst this has a visually improved effect on animation of close pedestrians the overall performance degradation makes interpolation between two key frames the preferred option.

With the previous technique every pedestrian in the population is rendered with the same detail level. A more suitable technique is to therefore apply a LOD rendering system which varies the pedestrian's fidelity depending on distance to the viewer.

This is achieved through the use of a generalised feedback system available for retrieving data about the agent population without CPU read-back from the graphics card. Parallel reduction is used to reduce values in agent space to singular values for a number of common reduction functions such as minimum, maximum, sum and count. For the purposes of a LOD system is it required that the total number of each detail level is know. An agent variable therefore is used to hold a LOD level and is calculated during the agent update stage. A reduction function for each detail level then uses a filtered count function, counting only the number of occurrences of the specified detail level. After the parallel reduction is complete the agent data must then be sorted according to the LOD levels. This ensures that calling a display list for each detail level, the number of times reported by the feedback step, matches the detail levels to the agent data. Whilst this technique is computationally more expensive due to the secondary sort, it allows a massive reduction in rendering overheads when high resolution models are required. This technique is demonstrated in figure 3 which shows pedestrians coloured by their corresponding LOD. Additionally the same technique can be applied to achieve variance in pedestrian representation. In this case varying pedestrian models each with an associated value are used within the population with the value used as feedback and sorting key. This technique also allows simultaneous LOD rendering as long as each LOD pedestrian representation combination has an associated identifier and display list representation.

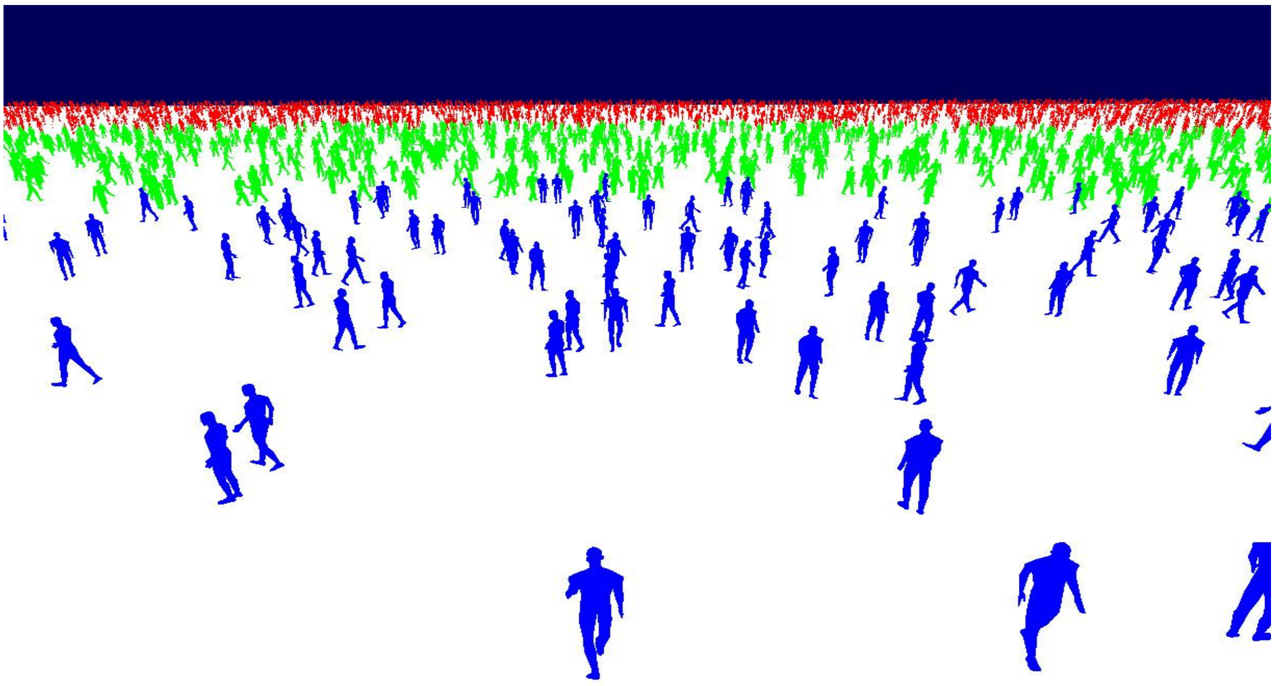


Figure 3 - 65'000 fully interacting agent based pedestrains rendered by LOD level

PERFORMANCE AND RESULTS

In order to test the simulation and rendering performances a range of pedestrian population sizes have been tested using varying rendering fidelities. The results obtained are based on a single PC with an AMD Athlon 2.51Ghz Dual Core Processor with 3GB of RAM and a GeForce 8800 GT. In all cases pedestrian behaviour incorporates all forces described in equation 1. Figure 4 additionally demonstrate a more complex force map representing the Peace Gardens area of Sheffield city centre. The results of a pedestrian simulation compared to satellite imagery are also show in figure 9. Simple zoning is demonstrated in figure 5 and has an immeasurable performance effect on the simulation compared to the following results presented in this section.

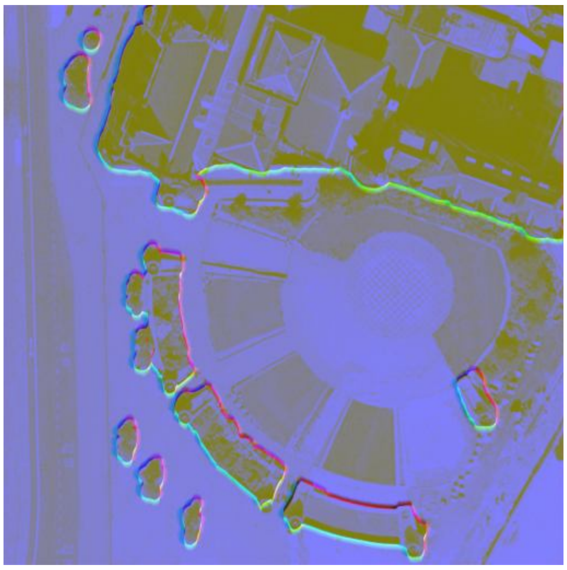


Figure 4 - A Force Map of Sheffield Peace Gardens

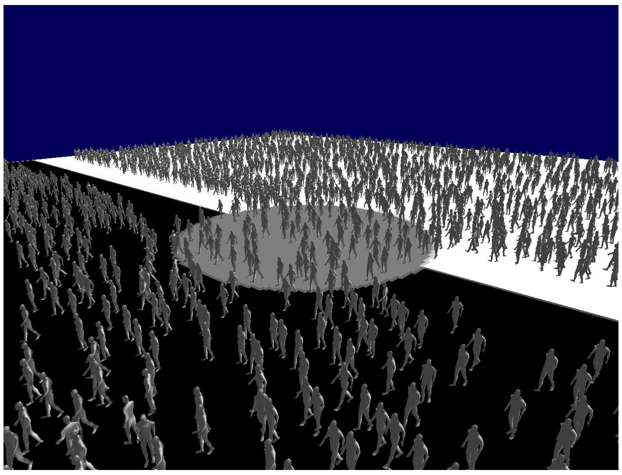


Figure 5 - Zoning around a congestion point (Black to White boundaries represent walls)

Simulation performance is considered by increasing the population size whilst maintaining a roughly static population density. Figure 6 demonstrates the population density by showing the number of pedestrians considered for communication (lookups) and the number actually inside the pedestrians communication radius (communications). As the population density is constant the communications to lookup ratio remains at roughly 35% in all cases. Figure 7 shows a performance chart which demonstrates the effect of increased population sized on performance. Two pedestrian vision (or interaction) radii are used, the first of 4m is suggested in Helbings work (Helbing and Molnar 1995) whilst the second 32m radius acts to demonstrate the performance in cases of longer range social force planning or higher congestion population densities. In both cases an environment force map is used to simply direct pedestrians away form the outer edges of the environment. From the results it is clear that the simulation performance which also includes the basic direction triangle rendering is suitable for large population sizes. More specifically interactive population sizes of 262’144 pedestrians can be maintained at 13 fps or 65’536 at 42 fps for a 4m pedestrian vision.

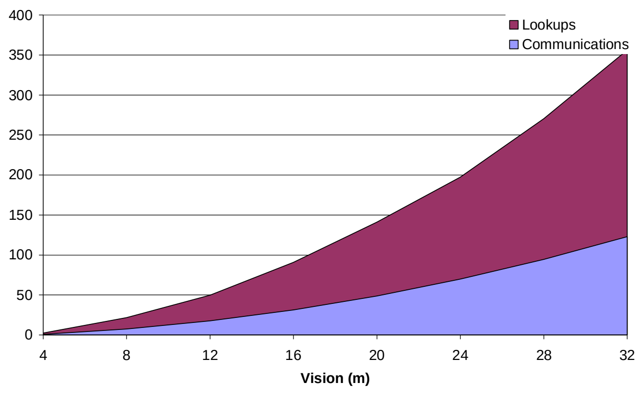


Figure 6 - Pedestrian vision compared to pedestrian lookups and inter-agent communications

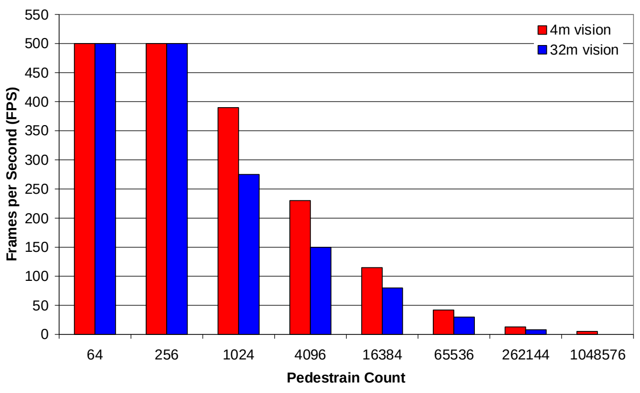


Figure 7 - Simulation and rendering performance for primitive agents with 4m and 32m vision

In order to consider more advanced rendering polygon models of increasing complexity have been used (detail level 0 represents lowest complexity, detail level 2 the highest). Figure 8 demonstrates that even at relatively high population sizes of 16384 pedestrians, high detail models (level 2) of up to 1000 polygons can be used at 40fps. The more modest pedestrian representation of 400 polygons (level 1) with the same population size achieves a performance of 40 fps and the more intelligent dynamic LOD rendering improves further achieving 50fps. What is noticeable is the cost of using a dynamic LOD for agent population sizes of less than 1024. In fact when agent sizes are this small the overhead of the additional sort step suggests that high resolution models for agents can be used whilst sustaining a performance of almost 200fps.

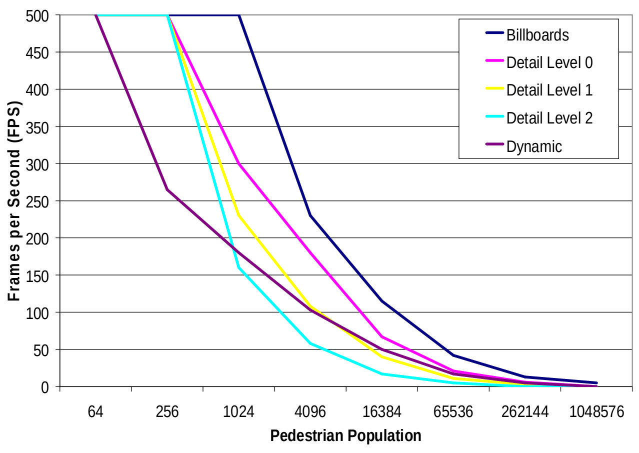


Figure 8 - Simulation and rendering performance for advanced pedestrian rendering at various detail levels

With respect to previous work the performance of the simulations in this paper are beyond that of existing social forces implementations (Courty and Musse 2004, Courty and Musse 2005). Whilst improvements in GPU hardware have some part to play in this, the decision to maintain data storage and simulation entirely on the GPU plays an important role. In contrast with hardware assisted agent based systems Reynolds PS3 work (Reynolds 2006) achieves decoupled simulation and rendering (1/8 of the population per rendering frame) of 10’000 agents at 60 fps in contrast with 16,000 at 67fps in our own system when compared with detail level 0 which is similar in complexity. Its is no surprise that work by D’Souza et al (D’Souza et al. 2007) work boasts a higher performance than that of our own however it is important to note that our own system takes into account pedestrian rendering and animation and is furthermore not limited to a regular grid with pedestrians being well distributed without any performance degradation. Likewise the performance of Greens (Green 2007) physically based particle demonstrations which use the same boundary scatter technique are slightly higher than our own results (most likely as a result of utilising the CUDA radix sort which limits the algorithms to G80 hardware) however the ridged particle and partition size vastly reduces the number of particles considered during the more simplistic update stage.

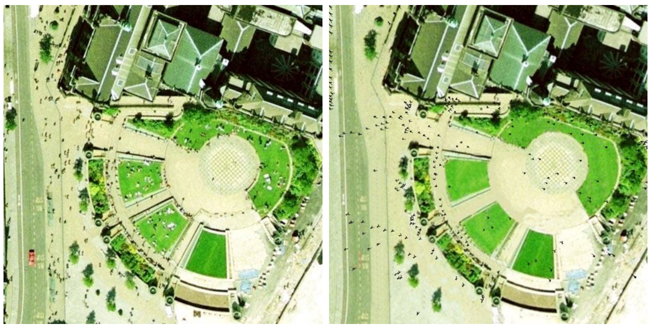


Figure 9 - Satellite imagery of Sheffield Peace Gardens (Left) and our simulation (Right)

DISCUSSION AND FUTURE WORK

In summary this paper has presented a high performance agent based pedestrian simulation. Further more the work has been designed with a scripting based interface to allow the incorporation of more advanced agent based behaviours in the future. As pedestrian data and simulation has remained entirely on the GPU rendering of pedestrians has been considered. Using real time feedback a LOD system has been presented which has improved the performance and allowed high detail models of pedestrians to be used with large pedestrian population sizes. Whilst pedestrian behaviour has demonstrated both long range inter pedestrian interactions and path planning using zoned areas the work could benefit from the inclusion of more advanced long range path planning as in (Pettré et al 2006). Other future work will concentrate on integration with large scale city modelling and automatic path finding within city environments. Pedestrian representation will also be addressed by considering individualised walking styles through kinematics and improved variance across the population through the use of more varied polygon model templates.

REFERENCES

Adra, S.F; Coakley, S; Kiran, M. and McMinn, P. 2008. “An Agent-Based software platform for modelling systems biology”, Epitheliome Project Report, Sheffield University

Courty, N. and Musse, S. R. 2005. “Simulation of large crowds in emergency situations including gaseous phenomena”. In Proceedings of IEEE Computer Graphics International 2005, pages 206--212.

Courty, N. and Musse, S.R. 2004. “FASTCROWD: Real-Time Simulation and Interaction with Large Crowds based on Graphics Hardware”. Short Paper in ACM SCA 2004 - Symposium on Computer Animation, 2004. Grenoble, France. 2004.

D'Souza, R. M; Lysenko, M. and Rahmani, K. 2007. “SugarScape on steroids: simulating over a million agents at interactive rates”, Proceedings of Agent2007 conference. Chicago, IL

Govindaraju, N.K; Raghuvanshi, N; Henson, M. and Manocha, D. 2005, “A Cache-Efficient Sorting Algorithm for Database and Data Mining Computations using Graphics Processors”, UNC Tech. Report 2005

Green, S. 2007, “CUDA Particles”, NVIDIA Whitepaper, November 2007.

Harada, T. 2007. “GPU Gems 3: Real Time Rigid Body Physics on GPUs”, Addison Wesley, pages 611-632.

Harris, M. J; Coombe, G; Scheuermann, S. and Lastra, A. 2002. “Physically-Based Visual Simulation on Graphics Hardware”. In Proc. 2002 SIGGRAPH / Eurographics Workshop on Graphics Hardware 2002.

Helbing, D. and Molnar, P. 1995. “Social Force Model for Pedestrian Dynamics”. Physical Review E, Volume 51 pages 42-82.

Helbing, D; Farkas, I. J; Molnar, P. and Vicsek, T. 2002. “Simulation of Pedestrian Crowds in Normal and Evacuation Situations”. In Pedestrian and Evacuation Dynamics, Springer, Berlin, pages 21-58.

Kipfer, P; Segal, M. and Westermann, R. 2004. “UberFlow: a GPU-based particle engine”. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware (Grenoble, France, August 29 - 30, 2004). HWWS '04. ACM, New York, NY, pages 115-122

Latta, L. 2004. “Building a Million Particle System”, In proceedings of Game Developers Conference, San Francisco, CA

Pettré, J; de Heras Ciechomski, P; Maïm, J; Yersin, B. Laumond, J.P. and Thalmann, D. 2006. “Real-time navigating crowds: scalable simulation and rendering”. Computer Animation and Virtual Worlds 17, 3--4, 445--455.

Quinn, M; Metoyer, R. and Hunter-Zaworski, K. 2003. “Parallel Implementation of the Social Forces Model”. In Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics (August 2003), pages 63–74

Reynolds, C. 2006. “Big fast crowds on PS3”. In Proceedings of the 2006 ACM SIGGRAPH Symposium on Videogames (Boston, Massachusetts, July 30 - 31, 2006). sandbox '06. ACM, New York, NY, pages 113-121

Reynolds, C. W. 1987. “Flocks, Herds, and Schools: A Distributed Behavioural Model”. In Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.

Reynolds, C. W. 1999. “Steering Behaviors For Autonomous Characters”. In the proceedings of Game Developers Conference. San Jose, California. Miller Freeman Game Group, San Francisco, California. pages 763-782.

Richmond, P. and Romano, D. 2008, “Agent Based GPU, a Real-time 3D Simulation and Interactive Visualisation Framework for Massive Agent Based Modelling on the GPU”, In Proceedings International Workshop on Super Visualisation (IWSV08) 2008. *In Press.*

Romano, D.M; Sheppard, G; Hall, J; Miller, A; and Ma, Z. 2005. “BASIC: A Believable, Adaptable Socially Intelligent Character for Social Presence. PRESENCE 2005”. The 8th Annual International Workshop on Presence, 21-22 September 2005, University College London, London, UK.

Treuille, A. Cooper, S. and Popović, Z. 2006. “Continuum crowds”. ACM Transactions on Graphics (TOG), v.25 n.3, July 2006, pages 1160-1168