INTERACTIVE, GPU-BASED URBAN GROWTH SIMULATION FOR AGILE URBAN POLICY MODELLING

Michel Krämer and Andreas Kehlenbach
Fraunhofer Institute for Computer Graphics Research IGD
Competence Center for Spatial Information Management
Fraunhoferstr. 5, 64283 Darmstadt, Germany
Email: michel.kraemer@igd.fraunhofer.de
Email: andreas.kehlenbach@cs.uni-frankfurt.de

KEYWORDS

GPU computing, Agent-based modelling (ABM), Urban simulation, Urban policy modelling, Urban planning, Geographic information systems (GIS)

ABSTRACT

In this paper we present a novel approach of simulating urban growth by utilising the computation power of modern GPUs. The simulation results can be used in urban policy modelling to reduce turnaround times in the policy cycle. We use a state-of-the-art agent-based simulation model that consists of rules to describe human behaviour. The simulation incorporates geospatial information such as land-use, current population density and road network data. In order to simulate the phenomena of urbanisation, in our model citizens more likely settle near roads or existing settlements/cities. In this paper we present our implementation that is based on the FLAME GPU framework. Each agent on the GPU represents a group of citizens at a specific location. In order to evaluate our approach we present a practical use case. We measure the performance of our implementation and compare it with a Java-based solution. Finally, we discuss our approach and show opportunities for agile and interactive urban policy modelling.

INTRODUCTION AND MOTIVATION

The term "urban sprawl" describes the problem of modern cities growing quickly resulting in wide-spread developments with a very low density. This often has negative effects on environment and therefore on people's health: more land is covered with buildings or streets; public transport in suburbs is often not sufficiently developed and so citizens have to use the car to get to their job or to the city centre which effectively leads to a higher air pollution. Besides, urban sprawl may also affect the cultural life and family life. People living in suburbs sometimes participate less in cultural events than people living near the city centre. Long travels to work and back to home reduce the time an employee can spend with his or her family.

Urban planning and policy modelling therefore aim for creating more compact but at the same time sustainable and healthy cities. This development requires infrastructure changes that have to be well thought out. So, urban planners more and more involve citizens in the discussion about urban development plans in order to create a city that is well received by everyone. They make use of simulations based on geospatial information. Innovative techniques such as 3D visualisation help urban planners to present the simulation results to decision makers and to the public.

Modern urban policy modelling deploys a so-called policy cycle (see Krämer et al., 2013). Simulations and 3D visualisations are used to gain feedback from decision makers and citizens. This feedback can then be incorporated in new simulations which are presented to the public again. This loop repeats until a general agreement on the planning has been found. The shorter the feedback cycle is, the faster a final decision can be made.

Creating such simulations is currently a time-consuming task that may take several hours or even days with existing solutions (see section "Performance" below). Urban planners often make use of modern satellite imagery to improve their calculations. For example, satellite images or LIDAR data spanning several years allow urban planners to calculate urban growth and hence to estimate future trends. The ongoing development of sensor technology leads to more accurate data sets which may be exploited to achieve better simulation results. However, at the same time the volume of data to process becomes larger and larger which makes them harder to process with standard geospatial information systems (GIS). Nonetheless, quickly creating simulations based on such data sets is a crucial part for the urban policy feedback cycle.

Modern computer architectures with multi-core CPUs and GPUs allow for creating high performance applications (cf. Owens et al., 2007). However, current GIS solutions do not fully take advantage of this yet. In practice, urban planners process raster data or point clouds such as satellite images or LIDAR data respectively with software tailored to simple workstations. In recent years these workstations have evolved and already include sophisticated graphics hardware. With this hardware it now becomes possible to not only create high performance 3D visualisations but also to make use of the thousands

Proceedings 27th European Conference on Modelling and Simulation ©ECMS Webjørn Rekdalsbakken, Robin T. Bye, Houxiang Zhang (Editors)
ISBN: 978-0-9564944-6-7 / ISBN: 978-0-9564944-7-4 (CD)

and millions of cores offered by a modern GPU to create geospatial simulations.

Modelling the behaviour of citizens in an urban environment can be rather complex in that it is non-linear and possibly chaotic. A lot of individual factors have to be taken into account that make the model large and hard to comprehend. Agent-based modelling (ABM) attempts to simplify such problems. Agents are autonomous units that act on their own, just like citizens. Modelling urban life becomes a lot easier by considering only one citizen or a group of similar citizens and by representing them as individual agents. Modern graphics hardware allows agent-based simulations to run on the GPU. So, it is possible to create high-performance simulations modelling urban life on the graphics card.

To summarise, in order to create sustainable, compact cities, urban planners deploy a feedback cycle that is based on geospatial simulations. The shorter this cycle is, the faster decisions can be made. However, geospatial data—which provides the basis for such simulations—becomes larger and larger and so simulations take more and more time with current GIS technology. In this paper we therefore present a new approach of interactively simulating urban development with modern GPU hardware. We use agents to model real urban life. We describe our implementation and evaluate its performance compared to a pure Java application. We conclude with a final discussion on the applicability of our approach to a practical use case, and we show opportunities for agile urban policy modelling.

RELATED WORK

Agent-based simulation has already been applied in the area of geographical information systems (GIS). For example, Gimblett proposes to model social and ecological phenomena such as population dynamics, disease epidemics or urban growth with agents (Gimblett, 2002). Gilbert and Troitzsch explain how to use the multi-agent approach to perform social simulation including population changes or business forecasting (Gilbert and Troitzsch, 2005). Gebetsroither presents the MASGISmo platform (Multi-paradigm Agent-based System Dynamics GIS modelling platform) that he uses to simulate social behaviour, natural resources, land use change, or environmental changes (Gebetsroither, 2010). He claims agents to be more appropriate to describe human behaviour than differential equations that try to model complex system dynamics. On the other hand, strategic problems and long-term policy development also have an influence on urban development. Gebetsroither therefore combines multi-agent simulation with elements from the area of system dynamics. He also includes stochastic variation to increase realism. This multi-paradigm approach is based on an idea by Scholl (Scholl, 2001).

The MASGISmo platform uses the RepastJ framework which is a Java library that is able to perform agent-based simulation using multiple CPU cores. RepastJ does not exploit the possibilities of modern graphics hardware. Although geographical information systems (GIS)—such as MASGISmo and others—could make great use of the performance gain offered by GPUs, this possibility has not been fully exploited yet. However, since the advantages of performing parallel computations on the GPU have already been described (Owens et al., 2007; Lupton and Thulin, 2008; Nickolls and Dally, 2010) there is some work trying to apply this approach to GIS. Balz and Haala, for example, use GPUs to perform SAR (Synthetic-Aperture Radar) image interpretation in realtime (Balz and Haala, 2006). They implement algorithms such as SAR rasterisation or LIDAR point triangulation on the GPU. They state that their solution is able to "simulate and visualise huge amounts of 3D data" and that it is therefore "the best choice for simulating city models" (Balz and Haala, 2006).

Combining the multi-agent approach with the advantages of GPU-based calculation offers a great opportunity for geospatial simulation. For example, Strippgen and Nagel use GPUs to simulate urban aspects—in their case traffic (Strippgen and Nagel, 2009). They use agents to model human behaviour. In their algorithm each agent has a certain plan for the whole day consisting of activities and routes. An agent participates in the traffic simulation when it moves from one activity to another. Strippgen and Nagel are able to achieve a high performance gain by using the GPU compared to a pure Java solution.

There are a number of frameworks available that support agent-based simulation as well as GIS operations. AnyLogic, for example, is a commercial solution developed by XJ Technologies (http://www.anylogic.de). MASON (http://cs.gmu.edu/∼eclab/projects/mason) and NetLogo (http://ccl.northwestern.edu/netlogo) are open source products written in Java and Scala respectively. Another open source solution is Repast (http://repast.sourceforge.net) which is available for Java, .NET and Python. Apart from that, there are libraries that support spatial operations on the graphics card. CudaGIS (Zhang and You, 2012), for example, offers the possibility to model geospatial primitives on the GPU but it does not support agent-based simulations. There is currently only one framework that supports both, agent-based simulation as well as GIS operations on the GPU. Its name is FLAME GPU (http://www.flamegpu.com). It is an extension to the FLAME framework which was written in C. FLAME GPU uses the same XML-based approach to describe agents as FLAME, but runs the simulation on the graphics card. In this work we hence use FLAME GPU.

EXAMPLE USE CASE AND REQUIREMENTS

In this section we describe a simple, yet realistic use case that we will utilise later to evaluate our approach. The use case is about simulating the development of population density in a certain area. Our test data set consists of

three rectangular grids in the ESRI ASCII grid file format. Each of them covers an area of 4300×2200 cells with a precision of 100 meters per cell in both directions. The first grid provides information on current land use. Each cell contains a number that means a certain type of land use—e.g. 1 for forest, 2 for river, 3 for building land, etc. The second grid contains information about population density. Each cell in this grid contains the number of citizens living there. The third grid represents the road network. A cell denoted with 1 contains a road whereas a cell with a 0 does not.

While testing this data with pure Java applications such as MASGISmo (see above) we observed the following:

- Since performing an urban growth simulation with existing tools takes a long time (often more than several hours up to a few days) it cannot be used well for agile and interactive policy modelling as it is described above.
- Existing tools using agent-based modelling are only able to manage a certain number of agents (depending on the system environment). Up to this number the performance typically increases, but beyond this break-even point the simulation will actually become slower.
- If the used data sets are large—even though the number of agents is low—a single computation step takes a noticeable amount of time.

These issues lead us to the following requirements:

- The user should be able to control parameters to interactively affect the simulation results.
- Our implementation should be able to manage much more agents as traditional implementations.
- It should significantly decrease the amount of time needed for one simulation step.

SIMULATION MODEL

In order to let our simulation create realistic results, based on the results from Gebetsroither we defined the following rules (cf. Gebetsroither, 2010):

- At the beginning the population grows.
- If the population reaches a specific density, we assume that from then on the population growth stagnates.
- We assume that citizens more likely settle near roads and existing cities or settlements. So, the population grows faster in these areas.

We use the information from the land use grid to let citizens settle down on grid cells which are denoted as building land whereas we do not allow citizens to settle down on cells which are denoted as forest, river, etc. At the beginning, we initialise our simulation with the contents of the population density grid. Later this grid is used to store population density changes—i.e. the simulation results. The third grid containing the roads is used for the last rule, namely that the population grows faster near roads as well as existing cities and settlements. Information on the latter is gained from the other two grids.

At the beginning we associate one agent to each population grid cell. This agent represents the group of citizens in this cell but not one particular citizen. This works, because the population density grid only contains abstract numbers about how many citizens live in each cell. An agent is able to let the population density grow or decline in its cell. Therefore it uses information from neighbouring cells. We use the Moore neighbourhood here, which means we only take direct neighbours into account.

We first start with a randomly generated number of citizens who want to settle down in the area. We let the agents decide where these citizens settle down. They do so with decreasing probability near roads and already settled citizens. If a citizen settles down, the population density in the respective cell will be increased.

In each simulation step a certain number of new citizens are born. To simulate negative growth or stagnation we introduce a mortality rate. If the birth rate is higher than the mortality rate the overall population will grow. Otherwise the overall population will decline. We dynamically alter both rates so that the population first grows and then stagnates.

The chosen model leads to a population growth in two dimensions. On the one hand, if agents settle on new cells the populated area will grow and new cities or settlements will be founded. On the other hand, a growing population density in a few single cells will lead to high-rise buildings.

In our use case we want to simulate urbanisation and so we do not model the aspect of citizens moving away from the cities.

IMPLEMENTATION

In this section we present our simulation tool and how we implemented it based on the FLAME GPU framework.

In order to load the input data into our tool we enhanced FLAME GPU to support ESRI ASCII grids in addition to the standard XMML format containing the simulation model. An ERSI ASCII grid consists of a small header containing the grid's size or resolution, a so-called NODATA value and the geospatial location of the covered area's lower-left corner. The NODATA value is used for all empty grid cells that do not contain a valid value. It is typically negative whereas all other values can be integers or floating point numbers. The values are separated by a whitespace character. For our use case we prefer ESRI ASCII grids over the XMML format already supported by FLAME GPU because our input data

is quite large and XMML is not designed for such an amount of data. ESRI ASCII grids store values in a much more compact way. Besides, the input data is already available as ESRI ASCII grid files so we don't have to convert them.

In order to be able load the large grid files into memory we split them into smaller chunks. The chunk size is defined by the FLAME GPU memory model which relies on a fixed number of agents that must be a power of two. As described above, we use one agent per cell, so our chunk size also has to be a power of two. FLAME GPU requires the developer to define this number at compile time. Splitting the grids into chunks allows us to use a fixed number at compile time as well as to load grids of arbitrary size.

In FLAME GPU agents are implemented as special functions that are executed on the GPU. Such functions are also known as *CUDA kernels*. Each simulation step consists of two kernels that are called consecutively. In the first kernel all agents send out a message containing their state—i.e. the population density in their respective grid cells. In the second kernel the agents decide whether to let population grow or not and then update their internal state. To build up communication between agents we use a Communicating X-Machine (COMX), a computation model provided by the FLAME GPU framework. A COMX consists of several X-Machines—a model that is similar to Finite State Machines (FSM)—that run in parallel. These machines communicate by sending messages over communication channels called *ports*. COMX are superior to pure Finite State Machines, because they are able to cover dynamic and static aspects of a system. In FLAME GPU messages sent by agents are saved to a global message list that resides in the graphics card's global memory.

Listing 1 outlines the first kernel and listing 2 the second one.

```
1  __FLAME_GPU_FUNC__ int ↩
       output_state(xmachine_memory_cell* xmemory, ↩
       xmachine_message_state_list* state_messages, ↩
       RNG_rand48* rand48)
2  {
3      // Add cell state to the global message list
4      add_state_message<DISCRETE_2D>( ↩
           state_messages, xmemory->color, ↩
           xmemory->population, xmemory->state, ↩
           xmemory->x, xmemory->y);
5      return 0;
6  }
```

Listing 1: The first CUDA kernel. Each agent sends its state as a message to the global message list.

In listing 2 we first iterate through the global message list. As described above, our simulation model requires citizens to more likely settle down near neighbouring roads and other citizens. So, we skip messages that are not from our direct neighbours. Based on the information received we then check if the current agent will increase population in its cell or not. The agent finally updates its internal state.

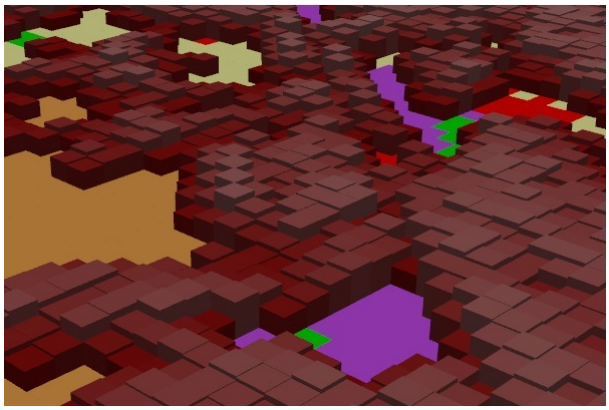

Figure 1: 3D visualisation of simulation results in our tool.

```
1  __FLAME_GPU_FUNC__ int ↩
       update_state(xmachine_memory_cell* xmemory, ↩
       xmachine_message_state_list* state_messages, ↩
       int* color_table, RNG_rand48* rand48)
2  {
3      // Initialize
4
5      // Iterate through global message list
6      xmachine_message_state* state_message = ↩
           get_first_state_message<DISCRETE_2D>( ↩
           state_messages, xmemory->x, xmemory->y);
7      while(state_message)
8      {
9          // Messages sent by neighbours are treated ↩
               separately.
10         // Get next message
11         state_message = ↩
               get_next_state_message<DISCRETE_2D>( ↩
               state_message, state_messages);
12     }
13
14     // Is settling allowed on this cell?
15     // Increase or decrease population in this cell.
16     // Save new state.
17
18     return 0;
19 }
```

Listing 2: The second kernel. The agent first evaluates all messages received from its direct neighbours. Then it decides whether to increase population or not and finally updates its internal state.

Our tool allows the user to interactively affect the simulation results. The user can change the population maximum for a single cell. That means if one cell reaches a maximum number of citizens new ones that want to settle there as well are redirected to neighbouring cells. The user can also interactively alter the land use grid. For example, he or she might declare a certain area as building land where citizens can settle down. The interaction possibilities help urban planners in the aforementioned policy modelling feedback cycle. By letting the user interactively change the simulation the feedback cycle's turnaround time can be reduced significantly.

Our tool contains a 3D visualisation of the simulation results. The calculated population density is displayed with blocks of different heights. The greater a cell's population density is, the higher the respective block will be. The cell's colour depends on the actual land use—e.g.

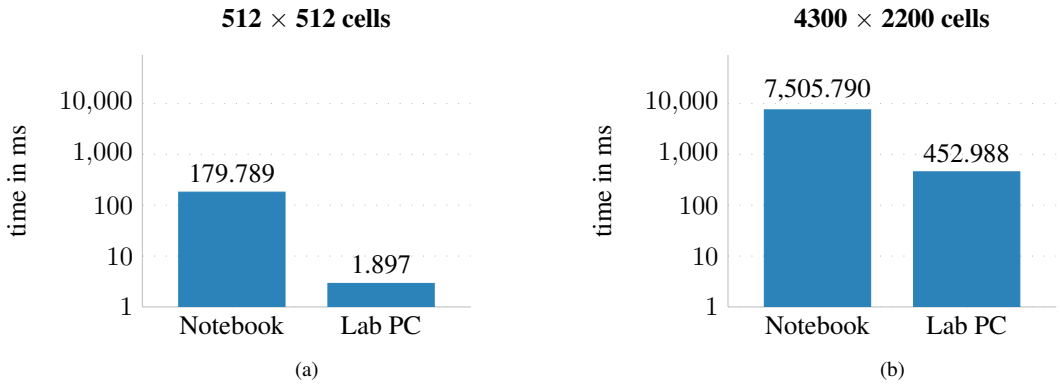

Figure 2: Subfigure (a) shows the number of milliseconds needed to perform a single simulation step on a grid chunk with 512×512 cells; (b) shows measurements for the complete grid with 4300×2200 cells.

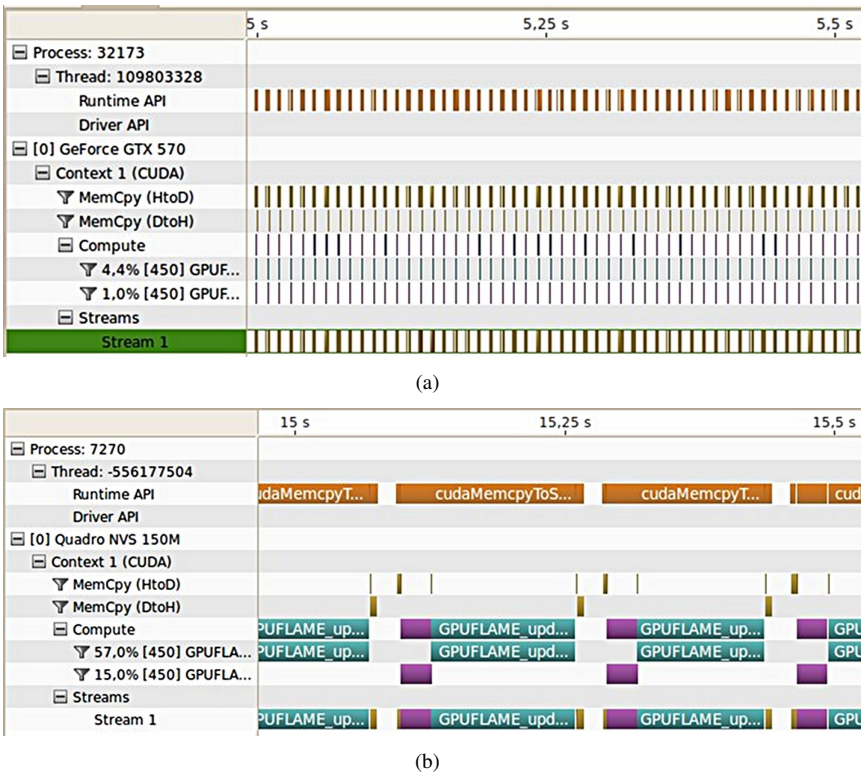

Figure 3: Comparison of the executed CUDA kernels per second between the lab PC (a) and the notebook (b).

settlements are displayed in red, forest in green, etc. If there is no population at all in a cell, it will be visualised as a flat plane. Figure 1 shows a screenshot of our tool. Scrolling over the grid, zooming in and out and freely rotating the grid allows the user to inspect and analyse the simulation results.

PERFORMANCE

In this section we present the results from evaluating our tool's performance based on time measurements and deep profiling with the NVIDIA Visual Profiler. The performance evaluation was executed on two computers:

- a lab PC equipped with an Intel Core i7 CPU, a NVIDIA GeForce GTX 750 (1,280 MiB RAM), and a NVIDIA Tesla C2075 GPU (5,375 MiB RAM);
- a notebook with an Intel Core 2 Duo CPU and a NVIDIA Quadro NVS 150M GPU (255 MiB RAM).

We evaluated two data sets: a single grid chunk with 512×512 cells and a complete grid with 4300×2200 cells. Figure 2 shows that only the simulation of the whole grid on the notebook is not very fast (almost 7 seconds per simulation step). The notebook's GPU simply has not enough computation power. But in comparison to

the plain Java application MASGISmo the results of both computers are fine. On our notebook the Java application was able to simulate about 100 agents in about 1 second. It therefore needed more than 2,600 seconds to compute a chunk with 512×512 cells. The GPU-based solution only needs a tiny fraction of this, even on the notebook.

Figure 3 depicts the results of profiling our implementation with the NVIDIA Visual Profiler. It shows that on the lab PC the GPU is not fully used to its capacity. It very often has to wait for more data to be transferred from the computer's main memory to the graphics card's memory. Performance can be improved by putting data transfer and simulation into separate streams that run in parallel. Nevertheless, this approach would not lead to much benefit on the notebook. In order to gain comparable results on both computers we did not implement this. An approach which dynamically splits up a grid depending on the size of the graphics card's memory could help to achieve optimal and scalable results on a given PC.

CONCLUSION

In this paper we described a novel approach of simulating urban growth using modern GPU hardware. We presented our simulation model which is based on rules that mimic the properties of urbanisation. In this paper we also presented a practical use case for our approach. We were able to show that it works well with the given data set. Also, we evaluated our implementation's performance and compared it to the existing Java application MASGISmo. Although the Java solution also uses agent-based modelling, our implementation is a lot faster. This has two reasons: a GPU is able to process a lot more agents within a single computation step than a CPU; and due to a GPU's superior computational power a simulation step takes much less time.

As a first step towards agile and interactive policy modelling our implementation allows the user to change the maximum number of citizens per cell and to alter current land use. These interaction possibilities are based on realistic procedures in urban planning. Restricting the number of citizens per cell corresponds to limiting the maximum number of stories for a building. Altering the land use grid corresponds to reassigning area usage types—for example, from agricultural land to building land.

Current GIS systems are not able to fully exploit the possibilities of modern computer hardware yet. Our GPU-based approach can be used to significantly speed up processing geospatial data. In particular, urban growth simulations can be performed in much less time. As shown above, our simulation only needs a tiny fraction of the time needed by MASGISmo. This opens up the possibility for real agile urban policy modelling that includes urban planners as well as stakeholders such as decision makers and even the public into the discussion. The fast turnaround time allows urban planners to deploy a feedback cycle that consists of planning, simulating, presenting the plan and the simulation results to the audience, and finally gaining feedback. This cycle repeats until a sustainable solution is found that is well received by every stakeholder. Moreover, the interaction possibilities in our implementation allow stakeholders to simulate different urban planning scenarios and to directly discuss about them.

In this paper we only presented a single use case for urban growth. We also tested only one raster data set. However, geospatial applications typically have to deal with many heterogeneous use cases and requirements as well as with lots of different data exchange formats. Apart from that, our simulation model only uses one modelling technique (agent-based modelling). MASGISmo on the other hand uses a multi-paradigm approach to increase realism—which by the way contributes only a little to the bad performance of this pure Java solution. Nevertheless, the results presented in this paper show that there's a lot of potential to make use of GPUs and agent-based modelling in geospatial applications. In particular, we think that our approach can be applied to other urban planning scenarios as well—for example, traffic simulation (Promnoi et al., 2009; Strippgen and Nagel, 2009; Caceres, 2012) or pedestrian and crowd simulation (Richmond and Romano, 2008; Passos et al., 2008).

ACKNOWLEDGEMENT

Research presented here is partly carried out within the project "urbanAPI" (Interactive Analysis, Simulation and Visualisation Tools for Urban Agile Policy Implementation), funded from the 7th Framework Program of the European Commission, call identifier: FP7-ICT-2011-7, under the grant agreement no: 288577, started in October 2011.

REFERENCES

Balz, T. and Haala, N. (2006). Improved real-time sar simulation in urban areas. In *International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3631–3634.

Caceres, N. (2012). Traffic Flow Estimation Models Using Cellular Phone Data. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–12.

Gebetsroither, E. (2010). Combining Multi Agent System modeling and System Dynamics modeling. In Trappl, R., editor, *Proceedings of the 20th European Meeting on Cybernetics and Systems Research (EMCSR)*, volume 2, Vienna, Austria.

Gilbert, G. N. and Troitzsch, K. G. (2005). *Simulation for the Social Scientist*. Open University Press, 2nd ed. edition.

Gimblett, H. (2002). *Integrating geographic information systems and agent-based modeling techniques for simulating social and ecological processes*. Oxford University Press, USA.

Krämer, M., Ludlow, D., and Khan, Z. (2013). Domain-specific languages for agile urban policy modelling. In *Proceedings of the 27th European Conference on Modelling and Simulation (ECMS)*.

Lupton, G. and Thulin, D. (2008). Accelerating HPC Using GPUs. Technical report, Hewlett-Packard Company.

Nickolls, J. and Dally, W. J. (2010). The GPU Computing Era.

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. J. (2007). A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113.

Passos, E., Joselli, M., Zamith, M., Rocha, J., Montenegro, A., Clua, E., Conci, A., and Feijó, B. (2008). Supermassive crowd simulation on gpu based on emergent behavior. In *Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment*, pages 81–86.

Promnoi, S., Tangamchit, P., and Pattara-Atikom, W. (2009). Road traffic estimation with signal matching in mobile phone using large-size database. *12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6.

Richmond, P. and Romano, D. (2008). A high performance framework for agent based pedestrian dynamics on gpu hardware. In *Proceedings of EUROSIS ESM*.

Scholl, H. J. (2001). Agent Based and System Dynamics Modeling: A Call for Cross Study and Joint Research. In *Hawaii International Conference on System Sciences*.

Strippgen, D. and Nagel, K. (2009). Multi-agent traffic simulation with CUDA. *2009 International Conference on High Performance Computing Simulation*, pages 106–114.

Zhang, J. and You, S. (2012). CudaGIS: Report on the Design and Realization of a Massive Data Parallel GIS on GPUs. In *ACM SIGSPATIAL IWGS Workshop*.

AUTHOR BIOGRAPHIES

MICHEL KRÄMER is deputy department head of the Spatial Information Management competence center of the Fraunhofer Institute for Computer Graphics Research IGD in Darmstadt, Germany. His research interests are in Compiler Construction, Language Recognition and Artificial Intelligence as well as Big Data and Cloud Computing. He's development lead of the 3D GIS area and has contributed to various open source products. Michel Krämer holds a master's degree in computer science from the THM University of Applied Sciences, Gießen, Germany where he's now also a lecturer. His email address is michel.kraemer@igd.fraunhofer.de.

ANDREAS KEHLENBACH holds the master's degree of Science in Computer Science from the Goethe University Frankfurt a.M., Germany. His master thesis was about interactive urban planning simulation on the GPU, which was discussed in this paper. His email address is andreas.kehlenbach@cs.uni-frankfurt.de.