A pedestrian simulation for very large scale applications

Christian Gloor[a], Duncan Cavens[b], Eckart Lange[b], Kai Nagel[a], Willy Schmid[b]
[a] Institute for Computational Science
ETH Zürich Zentrum, 8092 Zürich, Switzerland.
{chgloor,nagel}@inf.ethz.ch
[b] Institute for Spatial and Landscape Planning
ETH Zürich Hönggerberg, 8093 Zürich, Switzerland.
{cavens,lange,schmid}@nsl.ethz.ch

December 22, 2003

Abstract

Dieser Text stellt das Projekt "Planning with Virtual Alpine Landscapes and Autonomous Agents" (ALPSIM www page, accessed 2003) vor, welches vom Programm "Habitats and Landscapes of the Alps" des Schweizer Nationalfond finanziert wird. Das Projekt untersucht, inwieweit es möglich ist, Multiagentensimulationen einzusetzen, um landschaftliche Veränderungen in Touristengegenden in den Schweizer Alpen zu untersuchen. Das Projekt verwendet simulierte Agenten, welche die Landschaft "sehen" und entsprechend auf Veränderungen reagieren. Dieser Text beschreibt die allgemeinen Projektziele sowie Aspekte der Computerimplementation.

This paper introduces the project "Planning with Virtual Alpine Landscapes and Autonomous Agents" (ALPSIM www page, accessed 2003), which is funded by the Swiss National Science Foundation program "Habitats and Landscapes of the Alps." The project explores the feasibility of using autonomous agent modeling to evaluate future scenarios in a tourist landscape in the Swiss Alps. The project uses simulated people (agents) who "see" the landscape as surrogates for real people, in order to test their reactions against the simulated scenarios. This paper describes the overall project goals and the computational approach used to attain them.

1 Introduction

As many planning problems focus on processes that evolve over time in a complex environment, it is often difficult to evaluate the long term implications of a planning decision. Computer simulations have long been used as a method for evaluating proposed future scenarios in planning (Timmermans, 2003). However, most simulation efforts in spatial planning have focused on large spatial scales (such as at the city and regional levels) and on relatively abstract concepts (such as land use patterns, traffic and economic development), while one can argue that the planning decisions that have the most impact on individual citizens tend to be either at a relatively small scale or have very local impacts.

At these small scales (such as the sub-watershed or village), visual elements and the overall visual quality of the proposed planning intervention are extremely important. This is particularly true with areas dependent on tourism, which are often promoted based on their scenic qualities. Aesthetics are hard to quantify and therefore hard to integrate into a computer model. Because of the persisting perception that visual quality is too subjective a concept, it has largely been ignored in planning models and simulation. This is, in our opinion, a large oversight, as there is a whole class of planning problems where aesthetics are important, yet visual quality questions are largely left to designers at the individual project scale, with little consideration for how individual aesthetic choices combine to impact the larger landscape.

Our project integrates these aesthetic qualities with other factors such as availability of recreational opportunities, congestion and service levels using an agent based approach. We focus on their impacts on tourism, in particular summer hikers. Others (Gimblett, 2002) have used agent-based approaches to model tourists, but their focus has been primarily been on congestion issues. While congestion is a problem for the busiest and most famous tourism areas, it our contention that the encroachment of development and changes to the management of the landscape have the potential for far greater impacts on the attractiveness of a given area for tourists.

Figure 1: The landscape is a mixture of pasture and coniferous forests, dominated by Norway Spruce (Picea abies). The test site is characterized by significant topography and is considered ideal for walking and hiking.

The specific test site is a valley in the Gstaad-Saanenland region of south-western Switzerland. The communities of Schönried and Saanenmöser are at the two ends of the site; their economies are highly tourism dependent. While the primary tourism draw to the area used to be winter skiing, long term climate change is forcing the community to focus its efforts on building up a more diversified tourism economy. This includes capitalizing on its already strong reputation for summer hiking. The landscape is a mixture of pasture and coniferous forests, dominated by Norway Spruce (Picea abies). The test site is characterized by significant topography and is considered ideal for walking and hiking (see figure 1). The trails are very accessible to a wide range of hiking abilities due to the summer operation of one chair-lift and two gondolas. In the high season, the area is busy with hikers and walkers who easily fill the two main parking lots in Schönried.

A recent study in the area (Müller and Landes, 2001) identified that the biggest attraction for summer tourists are the area's scenic qualities. Hiking and walking is the primary recreational activity in the summer months. The focus on views was confirmed by our own study in 2002 (Cavens and Lange, 2003), which confirmed that views and landscape variety are the most important factors that influence hikers in their choice of hiking routes.

In addition to the community's desire to diversify its recreational economy, there are landscape policy issues that have the potential to change the desirability of the area for summer tourism. These issues include changes to the pattern of the landscape due to changing agricultural policy, shifts in forestry practices, closing of the gondolas and/or chairlifts, and increased holiday home construction. Any of these changes would have complex repercussions for the tourism industry: future scenarios to test the agent model will be selected from them.

2 Modeling Approach

2.1 Overview

Our approach is to model each tourist individually as an "agent". The approach is adapted from one used in traffic microsimulations. A synthetic population of tourists is created that reflect current (and/or projected) visitor demographics. These tourists are given goals and expectations that reflect existing literature, on-site studies, and, in some cases where sufficient data is not available, are based on experts' estimates. These expectations are individual,

meaning that each agent could potentially be given different goals and expectations.

These agents are given "plans", and they are introduced into the simulation with no "knowledge" of the the environment. The agents execute their plans, receiving feedback from the environment as they move throughout the landscape. At the end of each run, the agents' actions are compared to their expectations. If the results of a particular plan do not meet their expectations, on subsequent runs the agents try different alternatives, learning both from their own direct experience, and, depending on the learning model used, from the experiences of other agents in the system.

After numerous runs, the goal is to have a system that, in the case of a status quo scenario, reflects observed patterns in the real world. In this case, this could, for example, be the observed distribution of hikers across the study site over time.

A "plan" can refer to an arbitrary period, such as a day or a complete vacation period. As a first approximation, a plan is a completely specified "control program" for the agent. It is, however, also possible to change parts of the plan during the run, or to have incomplete plans, which are completed as the system goes.

2.2 Modular Structure

Any mobility simulation system does not just consist of the mobility simulation itself (which controls the physical constraints of the agents in a virtual world; see Sec. 3), but also of modules that compute higher level strategies of the agents. In fact, it makes sense to consider the physical and the mental world completely separately (Fig. 2). The most important modules of the mental layer are:

- **Route Generator**. It is not enough to have agents walk around randomly; for realistic applications it is necessary to generate plausible routes. In terms of graph language, this means that agents need to compute the sequence of links that they are taking through the network. A typical way to obtain such paths is to use a Dijkstra best path algorithm. This algorithm values individual links based on generalized costs, such as different views or different temperature levels.

- **Activity Generator**. Being able to compute routes, as the route generator does, only makes sense if one knows the destinations for the agents. A new technique in transportation research is to generate a (say) day-long chain of activities for each agent, and each activity's specific location. For our hiking simulations, possible activities include: be-at-hotel, visit-a-restaurant, etc. In a hiking simulations, unlike typical travel simulations, the travel (= the hike) connecting these activities will generally not be seen as a negative but as a positive part of achieving their goals and expectations. As a result, for our hiking simulation the route generator module needs to connect the activities in a way such that the connecting route reflects the expectations of the agent in terms of aesthetics, difficulty, as well as travel time.

- In the **synthetic population generation** module, the agents are generated. This includes demographic attributes to each agent, such as age, gender, income, etc. In the future, this should follow some demographic information about the tourist population in the area of interest; at this point, this is entirely random.

- **View Module** This module describes to the system what individual agents "see" as they move through the landscape. The agents field-of-view is analyzed, and events are sent to the system describing what the agent sees. This module is introduced in section 4.2,

- **Agent Databases**. The strategy modules so far are capable to generate plans, and the mobility simulation is able to store exactly one plan per agent and execute it. There should, however, also be a place where plans and in particular their performance is memorized, so that memory can be retrieved later.

  The agent database modules connect all of the modules in the system. For the project described in this paper, we envisage multiple agent databases arranged in a hierarchy (Fig. 3). More about how these modules find a good strategy and how they learn from previous simulation runs can be found in section 4.

- **Viewers** are built so that they directly plug into the live system. The simulation sends agents' positions to the viewer, which allows to look at the scenario from a bird's eye view and observe how the agents move. It is also possible to send that same data stream to a recorder which records it to file, while a player can read the file and send the data stream to the viewer exactly the same way it would come from the simulation directly. Finally, in order to deal with data conversion issues, it is also possible to pipe the data stream from the simulation through the recorder directly to the player and from there to the viewer.

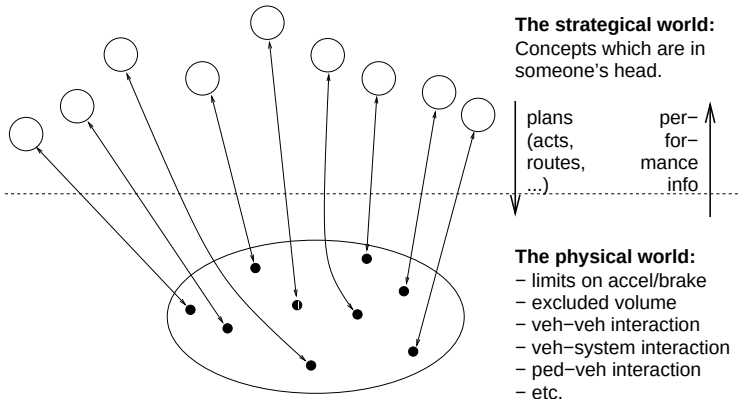

Figure 2: Physical and strategic (mental) layers of the framework.

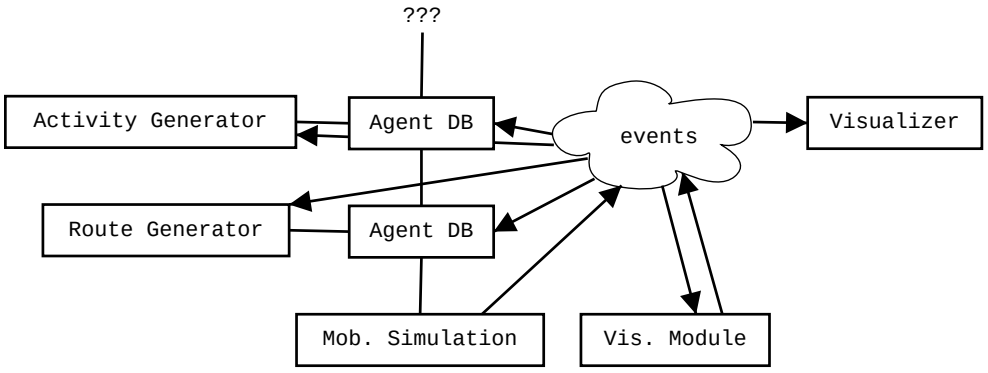

Figure 3: Software structure. Logical modules and the message flows that connect them. The cloud is a "broadcast network", which means that all messages sent to this network are distributed to all modules attached.

We have implemented two different viewers. One displays a 2D view, and is suited for situations where a lot of detailed information is needed, for example while debugging (Fig. 8). Also, a 3D viewer has been implemented (Fig. 12), as one of our overall project goals is to integrate decisions based on visual stimuli. The 3D viewer connects to the simulation using the same protocol as the 2D viewer. The user can move independently of the agents or can attach the camera viewpoint to a specific agent and see the landscape through the eyes of the agent. In order to reduce code duplication, the 3D viewer is essentially the same software as the view module described above.

3 Mobility Simulation

3.1 Selection of the pedestrian movement model

As mentioned above, the mobility simulation takes care of the physical aspects of the system, such as interaction of the agents with the environment or with each other. Typical simulation techniques for such problems are:

- In **microscopic simulations**, each particle is represented individually.
- In macroscopic or **field-based simulations**, particles are aggregated into fields. The corresponding mathematical models are partial differential equations, which need to be discretized for computer implementations.
- It is possible to combine microscopic and field-based methods, which is sometimes called **smooth particle hydrodynamics** (SPH; Gingold and Monaghan, 1977). In SPH, the individuality of each particle is maintained. During each time step, particles are aggregated to field quantities such as density, then velocities are

computed from these densities, and then each individual particle is moved according to these macroscopic velocities.

- As a fourth method, somewhat on the side, exist the queuing simulations from operations research. Here, particles move in a networks of queues, where each queue has a service rate. Once a particle is served, it moves into the next queue.

For our simulations, we need to maintain individual particles, since they need to be able to make individual decisions, such as route choices, throughout the simulation. This immediately rules out field-based methods. We also need a realistic representation of inter-pedestrian interactions, which rules out both the queue models and the SPH models.

For microscopic simulations, there are essentially two techniques: methods based on coupled differential equations, and cellular automata (CA) models. In our situation, it is important that agents can move in arbitrary directions without artifacts caused by the modeling technique, which essentially rules out CA techniques. A generic coupled differential equation model for pedestrian movement is the social force model (Helbing et al., 2000)

$$m_i \frac{d\mathbf{v}_i}{dt} = m_i \frac{\mathbf{v}_i^0 - \mathbf{v}_i}{\tau_i} + \sum_{j \neq i} \mathbf{f}_{ij} + \sum_W \mathbf{f}_{iW} \tag{1}$$

where m_i is the mass of the pedestrian and $\mathbf{v}_i$ its velocity. $\mathbf{v}_i^0$ is its desired velocity; in consequence, the first term on the RHS models exponential approach to that desired velocity, with a time constant τ_i. The second term on the RHS models pedestrian interaction, and the third models interaction of the pedestrian with the environment.

The specific mathematical form of the interaction term does not seem to be critical for our applications as long as it decays fast enough. Fast decay is important in order to cut off the interaction at relatively short distances. This is important for efficient computing, but it is also plausible with respect to the real world: Other pedestrians at, say, a distance of several hundred meters will not affect a pedestrian, even if those other pedestrians are at a very high density. We use an exponential force decay of

$$\mathbf{f}_{ij} = \exp\left(\frac{|\mathbf{r}_i - \mathbf{r}_j|}{B_p}\right) \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} , \tag{2}$$

which seems to work well in practice. $\mathbf{f}_{ij}$ is the force contribution of agent j to agent i; $\mathbf{r}_i$ is the position of agent i. Alternative more sophisticated formations are described by Helbing et al. (2000). For the environmental forces, $\mathbf{f}_{iW}$, the same mathematical form as for the pedestrian-pedestrian interaction is used.

3.2 Path following

The main problem with existing pedestrian models for our purposes is that they do not work when confronted with such large areas as are necessary to cover a complete hiking area: Covering an area of $(50\ km)^2$ area (necessary to allow hikes of length $25\ km$ in all directions from a central starting point) with cells of $(0.25\ m)^2$ results in 10^{10} cells, which needs at least $40\ GByte$ of memory. This makes the straightforward application of a technique based on cells impossible. Models based on continuous coordinates have the similar problem that all objects need to be represented, and again a straightforward implementation goes beyond available computer memory. All models have the problem that long-distance path following has never been looked at, at least not to our knowledge.

We introduced (Mauron, 2002; Gloor and Nagel, 2002) a model that uses only sparse information which fits into computer memory, runs efficiently on our scenarios, and has agents follow paths without major artifacts. This model will be described in the following.

For our simulation, we need to assume that the geometry is given by a graph (network) of hiking path plus some terrain features, and that the desired velocity of the hiker is consistent with this graph. The maybe first implementation that comes to mind is as follows:

- Make the desired velocity point directly to the next way-point: $\mathbf{v}_i^0 = v_i^0 (\mathbf{R} - \mathbf{r}_i)/|\mathbf{R} - \mathbf{r}_i|$, where $\mathbf{r}_i$ is the hiker's current position, R is the position of the next way-point, and v_i^0 is the magnitude of the desired velocity. Once a way-point is reached, $\mathbf{R}$ is moved to the next way-point.

- Set the environmental force field to zero on the path, and such that it pushes the hiker back onto the path outside.

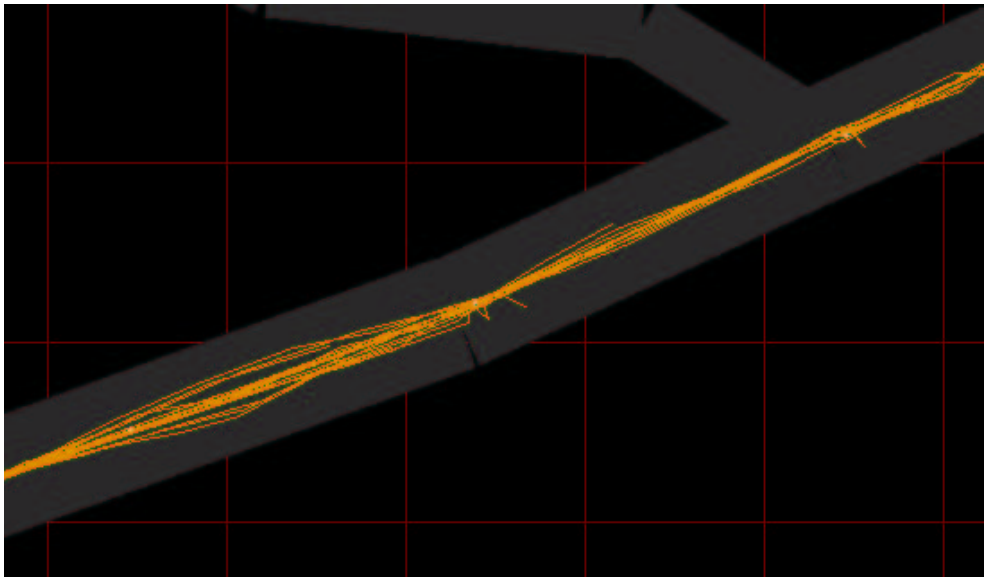

Figure 4: Traces of hikers in the naive model, where they are all pulled toward the same way-point. Note how the trajectories focus near the way-point, and diverge before and after. The width of the path remains unchanged.

However, this approach has the disadvantage that, close to a way-point, agents are artificially pulled toward that way-point even if that does not make sense (see Fig.4). This could be avoided by switching to the next way-point before actually reaching this way-point, but then without special measures pedestrians may not be able to execute a switchback, because the next way-point may pull them back on the current segment. Also, the approach quite in general does not work once curved segments are allowed between way-points.

We developed a more specialized approach (Mauron, 2002; Gloor and Nagel, 2002), in which this artifacts do not exist. Our model uses a path-oriented coordinate system (see Fig. 5) for the computation of the desired velocity. It also uses a so-called *path-force*, which pulls the agents back on the path when he moves away from its center (e.g. due to interaction with other agents or obstacles). Fig. 6 demonstrates that the new approach keeps the pedestrians on their side of the path even in the vicinity of way-points.

3.3 Computational aspects

It was argued earlier that a cellular automata representation of space did not seem appropriate for our purposes, because of problems with off-axis movement. Instead, we use a continuous representation of space. However, some aspects of our simulation, like the calculation of forces that affect the agent, depend on the spatial location of the agent. These forces are relatively expensive to calculate, since one needs to enumerate through all possible objects that could influence a given location.

Yet, since those forces do not depend on time, they can be pre-computed before the simulation starts. In order for this to be successful, some coarse-graining of space is necessary. For this, we use cells of size $25cm \times 25cm$, and assume that all time-independent forces are constant inside a cell. The resulting force field (Fig. 7) becomes non-continuous in space, but this is not a problem in practice since this only influences the acceleration of pedestrians. That is, the acceleration contribution from the environmental forces can jump from one time step to the next, but since time is not continuous, this is not noticeable.

Pre-computing the values for all cells in a hiking region of, say, $50km \times 50km$, does not fit into regular computer memory. To avoid this problem, we implemented two methods: lazy initialization, and disk caching. By **lazy initialization**, we mean that the values are computed only when an agent really needs them, also knows as *Virtual Proxy Pattern* (Gamma et al., 2001, pages 207–217). In practice, the simulation area is divided into blocks of size $200m \times 200m$. Every time an agent enters one of these blocks, the values for *all* cells inside that block are computed. Since hiking paths cross only a small fraction of those blocks, the cell values for many blocks in our hiking area will never be calculated.

In addition, the cell values, once computed, are stored on disk (**disk caching**). Every time when an agent

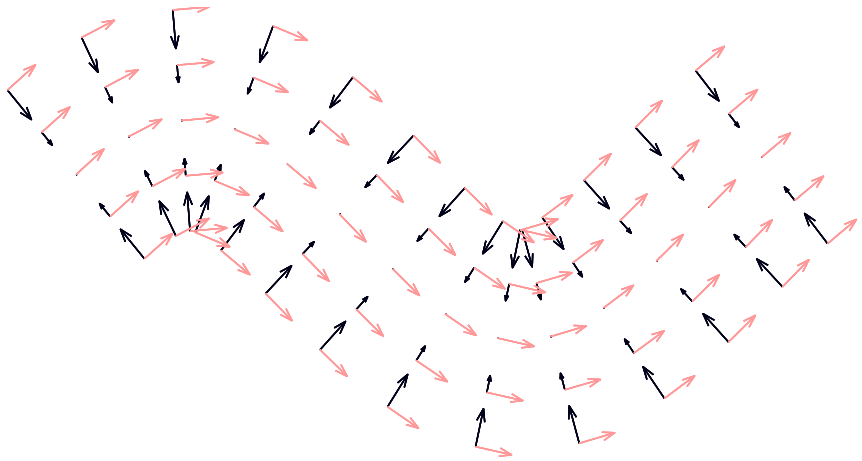

Figure 5: Path-oriented coordinate system for the computation of the desired velocity and the path forces. The light arrows show the desired velocity, which drives the agent forward along the path. The dark arrows show the path force, which pull the agent toward the middle of the path.

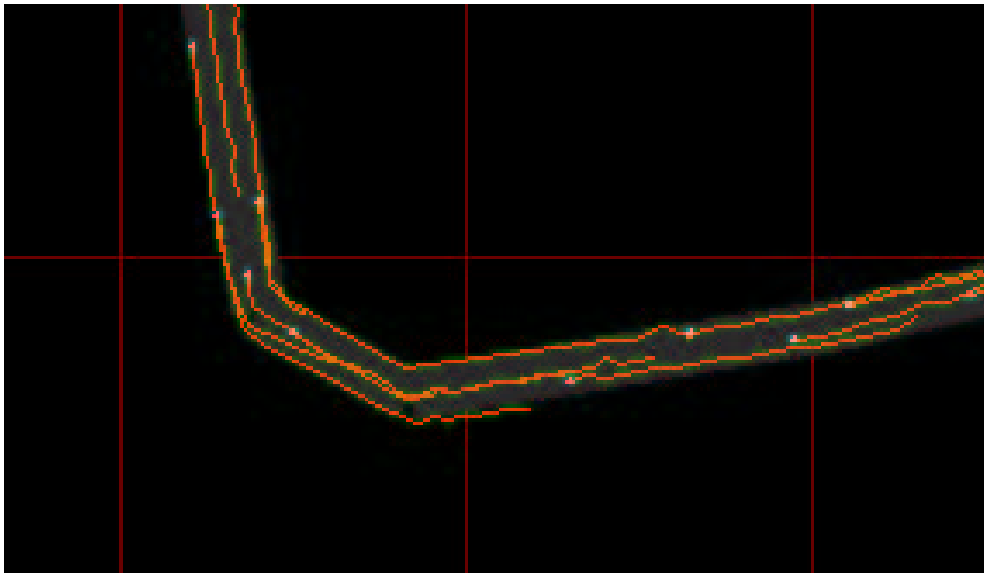

Figure 6: Traces of pedestrians walking in the same direction according to our model. Note that they stay on their side of the path, even at a bend.

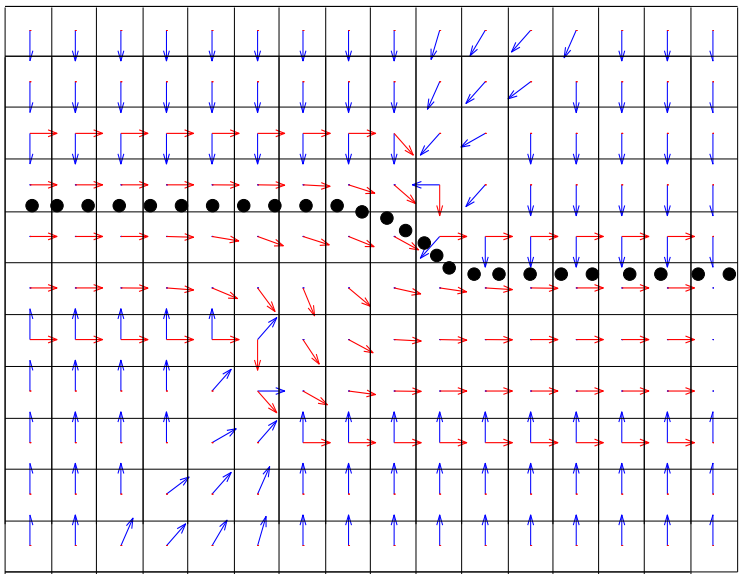

Figure 7: The hybrid simulation technique. The forces (arrows) are valid for the whole cell; a pedestrian's trajectory (dots) can follow arbitrary positions.

encounters a block for which the cell values are not in memory, the simulation first checks if they are maybe on disk. Computation of the cell values is only started when those values are not found on disk. In consequence, a simulation started for the first time will run more slowly, because the disk cache is not yet filled.

If the simulation runs out of memory, then blocks which are no longer needed, i.e. which have not been crossed by an agent for a long time, are unloaded from memory. If they are needed again, they are just re-loaded from disk. This corresponds to the *Least Recently Used (LRU) Page Replacement Algorithm* described by Tanenbaum (2001, pages 218–222).

An additional advantage of the blocks, well known from molecular dynamics simulations, is that one can use them to cut off the short-range interaction between the pedestrians. Agents which are not in the same or one of the eight adjacent blocks are ignored. This implies that there needs to be some data structure where agents are registered to the block. Agents that move from one block to another need to unregister in the first block and register in the next one. In this way, an agent searching for its neighbors only needs to go through the registered agents in the relevant blocks. This brings the computation complexity from O(N^2) down to O(NM), where N is the number of all agents in the simulation, and M is the number of agents in a single block. M is a reasonably small number when compared to the number N of all agents in a real-world scenario.

For a testing scenario, of size $12\,km \times 15\,km$, we would need approx. 2.9×10^9 cells or 4500 blocks, resulting in 9 GByte memory requirement. The result of the lazy initialization together with the caching mechanism is that 50 Byte are enough for the scenario shown in Fig. 11. The computational speed for that simulation, with 500 hikers, was about 100 times faster than real time.

4 Mental Layer and Learning

4.1 How learning works (Mental modules and their interaction)

Every agent is created individually and treated microscopically. This means that we can assign *demographical data* to every agent. An agent knows, for example, its age, its physical fitness etc. Also it has an *expectation* of what it wants to experience, and what it likes most. This can be static, taken from the demographical data, or based on previous visits to this (or even another) hiking area.

Initially, every agent starts with a plan that, in its opinion, fulfills its expectations. For example, if the period of interest is a day, then such an initial plan might refer to a specific hike. To do this, the agent chooses *activity locations* it wants to visit, like hotel, peak of mountain, restaurant etc.

This chain of activity locations is then handed over to the *routing* module, which calculates the routes between

activities according to the information available. This information can be static and global, like shortest path information based on the street network graph. Also information that is local to the agents memory and might be uncertain can be used.

The mobility simulation then executes the routes. The agent experiences the environment and sends its perception as events (see later) to the other modules.

From here on, the system enters the *replanning* or *learning loop*. The idea, as mentioned before, is that the agents go through the same period (e.g. day) over and over again. During these iterations, they try to accumulate additional information, and to improve their plan.

The two critical questions are (1) how to accumulate, store, and classify that information, and (2) how to come up with new plans. Both questions are related to (artificial) intelligence, and we are certainly far away from answering them in their entirety. Nevertheless, our system contains the following elements which makes it able to learn:

- As one can see in Fig. 3, there are "agent databases" associated with each level of the planning hierarchy (e.g. activities, routes). The task of these agent databases is to store plans and to accord scores to them. That is, every time an agent comes up with a new plan, that plan is added to the repertoire of plans. In addition, the agent databases listen to the events emitted by the mobility simulation, and use these events to calculate a score for each plan once it has completed. If an agent database module assigns a bad score to a simulated hike, it tries to avoid its elements in future hikes. If it gets good feedback, however, it will try to reuse the elements of a hike, and combine these into a new hike, which will be simulated and scored again.

- Also the *route generator module* listens to the stream of events. However, rather than scoring complete plans, it constructs a graph-based mental map of the spatial environment. From the times when agents enter and leave links, the router learns how long it takes for an agent to walk along each link in the network. Also, the router cumulates all the other events that occur on every link. Using these values for each link in the network, the router is able to return the best route for each agent, based on its expectation and demographic data. For example, a physically fit person might prefer a route that is steeper, while a physically less fit person might get a route that includes more possibilities to rest.

- Similarly, the *activity generation module* takes the stream of events and generates another mental map, this time not based on a graph. In this mental map, it stores possible locations for activities, and their corresponding attributes.

- It is possible to make the mental layer module dynamic. In this case, it observes the agent on its path through the virtual environment. As soon as it detects an option that might yield a better score that the current plan, e.g. using a shorter path, or entering a restaurant, it notifies the agent in the simulation. Using this mechanism, agents are able to react to unpredicted changes in the environment, like weather changes or congestion.

It should be noted that the distinctions between these modules are not sharp. For example, an agent database may run out of memory if it memorizes as separate entities plans that differ only in small details; in that case, the agent database might have to start to build a mental map of the world in which case it becomes similar to the activity generation module as described above.

As said before, these aspects of the simulation concern the modeling of human intelligence, which is an unsolved (and maybe unsolvable) problem. Yet, one should recognize that for our simulations it is not necessary to model individual people correctly, but it is sufficient to obtain correct *distributions* of behavior. Our approach should be considered as a first step into that direction.

4.2 Modeling the Visual Landscape

As modeling agents' reaction to the visual qualities of the landscape is a key part of our project, it is necessary to model what the individual agent "sees" and interpret how what the agent sees reflects their expectations. This concerns the "scores" as mentioned above, which are necessary both for the agent database and for the mental maps.

There have been many attempts to model visual quality using GIS-based approaches. These approaches have distilled the overall 'attractiveness' of a particular place (usually modeled as a raster cell) into a single numerical factor, based on available GIS data. These analyses tend to be highly specific to a particular question (such as appropriateness for camping; Meitner and Daniel, 1997), and while useful for classifying huge areas of seldom visited land, their coarseness makes them less than appropriate for modeling smaller scale landscapes such as our test case. In particular, the fact that the existing models assign values to specific places, rather than on sequential

experiences, mean that they are not able to easily model concepts such as “landscape variety”, which, as mentioned above, was identified as one of the key points in attracting tourists in the Swiss Alps.

Rather than using a single visual quality model, our approach has been to give the agents the ability to “see” the landscape, and integrate their visual experience into the factors that are evaluated by the agent database modules. This allows us to model sequences of views, and provides a lot of flexibility in terms of exploring the importance of various visual parameters.

The view module exploits the capabilities of modern 3D graphics hardware to quickly perform visibility calculation. Using a similar technique as that described by Bishop et al. (2001), objects are rendered in perspective using false colors. These colors are assigned based either on unique objects or on logical groupings (such as stands of trees.) As the agents move through the landscape, the scene is rendered from the viewpoint of each individual agent. The rendering process produces a color image and a depth buffer, which is a natural byproduct of the rendering process used in current graphics hardware and describes how far away an object is from the viewpoint. As these visibility calculations are performed on specialized hardware, the process is able to scale well to complex scenes with little effort from the user perspectives.

While the process is considerably faster than other visibility approaches, it is not quick enough for our purposes. At a frame rate of 15+ frames per second, it quickly becomes the bottleneck for the entire simulation system.

We are exploring two different approaches to eliminate this bottleneck:

- pre-rendering the landscape using a grid of viewpoints. At each viewpoint, a series of view slices (each comprising a view angle of 15 degrees) is computed and analyzed. During the simulation, the nearest viewpoint to a given agent is selected, and these slices are reassembled to reflect the view direction of the agent.
- distributing the view modules across a cluster of rendering machines, with each one being responsible for a subset of the agents.

Both of these approaches offer considerable opportunity for speed improvement, and both are facilitated by the modular structure and communication strategies of the entire simulation system. It is very easy to swap the module that calculates the views for every agent at every time step with the pre-rendered implementation.

5 Communication between the modules

5.1 Introduction

Traditional implementations of transportation planning software, even when microscopic, are monolithic software packages – e.g. PAMINA (Rickert, 1998), EMME/2 (Babin et al., 1982), or VISSIM (PTV www page, accessed 2003). By this we do not dispute that these packages may use advanced modular software engineering techniques; we are rather referring to the user view, which is that one has to start *one* executable on one CPU and then all functionality is available from there. The disadvantage of that approach is twofold:

- All the different modules add up in terms of memory and CPU consumption, limiting the size of the problem.
- Although the approach is helpful when starting as *one* software project, it is not amenable to the coupling of different software modules, developed by different teams on possible/different operating systems.

A first step to overcome these problems is to make all modules completely stand-alone, and to couple them via files. Such an approach is for example used by TRANSIMS (TRANSIMS www page, accessed 2003). The two disadvantages of that approach are:

- The computational performance is limited by the file I/O performance.
- Modules typically need to be run sequentially. Each module needs to be run until completion before starting the next module. The system is unable to learn and adapt within a single module run. For example, the routing module can only be run *before* or *after* the mobility simulation. This implies that agents cannot change their routes while the mobility simulation is running.

The approach taking in this project is to couple the modules by messages rather than via files. In this way, each module can run on a different computer using different CPU and memory resources. It is even possible (as is with file-based interfaces) to make the modules themselves distributed; for example, we have mobility simulations (for traffic) which run on parallel Myrinet-equipped clusters of workstations.

It is clear that this now allows real-time interaction between the modules: for example, if an agent is blocked in congestion, the mental layer modules can react to this new situation and submit new routes or activities during a simulation run.

As our system is made up of many different module types and is designed to scale up to very large simulations, it is important to carefully consider how the modules communicate with each other. On simulations with tens of millions of agents, issues such as bandwidth usage, packet loss, and latency become increasingly important. As a result, we use different network protocols and implementations tailored to specific requirements of inter-module communication.

Our general design goals are to:

- Keep it simple. If someone develops a new module, it should be simple to implement the functions needed to communicate to other modules.
- Only use function libraries that are standard in every Linux distribution.
- Make it possible to port a module to another platform, for example a viewer to Windows or a simulation to a vector computer.
- Facilitate attaching existing modules from other simulation systems to our framework (e.g. by using a simple wrapper function).

5.2 Events

On their way from the starting location (e.g. hotel) to their individual destination (e.g. restaurant, peak of mountain), a pedestrian encounters different situations which she/he might enjoy more or less. As mentioned earlier, these perceptions are, as "events", sent to the decision-making modules, which record those events and are now able to decide how much an pedestrian enjoyed his trip. Typical events are spatial ("How many mountains can I see from here?") or computed directly by the simulation ("How many agents are near me?").

The mental modules listen to all those events. Depending on their different functionalities, they extract different types of information from them, as described in Sec. 4.1.

It is important to note that the task of the simulation of the physical system is simply to send out events about what happens; all interpretation is left to the mental modules. In contrast to most other simulations in the area of mobility research, the simulation itself does not perform any kind of data aggregation. For example, link travel times are not aggregated into time bins, but instead link entry and link exit events are communicated every time they happen. If some external module, e.g. the router, wants to construct aggregated link travel times from this information, it is up to that module to perform the necessary aggregation. Other modules, however, may need different information, for example specific progress reports for individual agents, which they can extract from the same stream of events. This would no longer possible if the simulation had aggregated the link entry/exit information into link travel times.

Despite this relatively clean separation – the mobility simulation computes "events", all interpretation is left to mental modules – there are conceptual and computational limits to this approach. For example, reporting everything that an agent sees in every given time step would be computationally too slow to be useful. In consequence, some filtering has to take place "at the source" (i.e. in the simulation), which corresponds to some kind of preprocessing similar to what real people's brains do. This is once more related to human intelligence, which is not well understood. However, also once more it is possible to pragmatically make progress. For example, it is possible to report only a random fraction of the object that the agent "sees", or it is possible to delegate the analysis of the views to a separate module (Sec. 4.2). Calibration and validation of these approaches will be interesting future projects.

5.3 Possible Protocols

This section discusses different possible implementation techniques for the messages. For some of the more promising technologies, the technological limitations are discussed, and they are measured in practice with our particular application in mind.

Existing Message Passing Tools When a single module is distributed across multiple computational nodes, one often uses MPI (Message Passing Interface; MPI www page, accessed 2003) or PVM (www.epm.ornl.gov/pvm/pvm_home.html, accessed 2003, Parallel Virtual Machine;). For example, our traffic simulation module (not described in this paper) is distributed to 64 hosts or more. It is also possible to use MPI for the communication between the modules, as

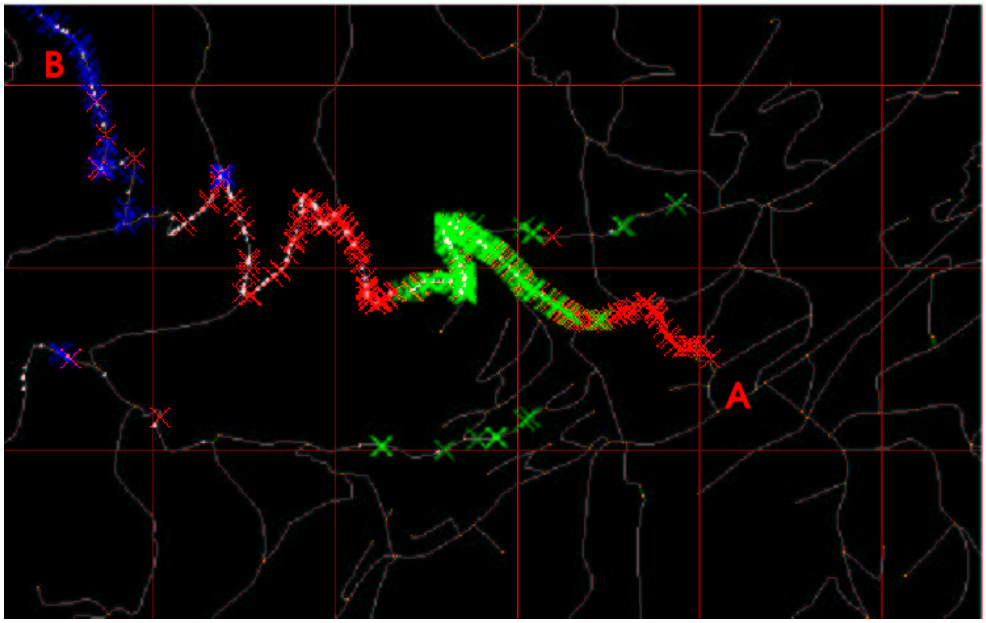

Figure 8: Agents hiking from a hotel to the top of a mountain. They report what they see during the hike. These *events* are rendered in different colors here; green, for example, means that the agents enjoyed a forest.

described in this paper. That approach has, however, the disadvantage that one is bound to the relatively inflexible options that MPI offers. For example, options to add or remove modules have only recently been added to the MPI standard, and multicast (see below) is not possible at all.

Reliable Data Streams TCP (Transmission Control Protocol) is a connection based, reliable IP (Internet Packet) protocol. Initially, a connection from the sender to the receiver must be opened. With this connection, both sides can send their messages as the connection is symmetric. TCP guarantees that the messages arrive in correct order, without errors. – The main disadvantage of TCP is that a connection must be opened for every host pair. If one side closes the connection (due to a program crash or a system reboot), the other side has to reopen the connection. This requires close attention from the programmer to handle all cases in a large system gracefully.

Unreliable Data Packets UDP offers, in comparison to TCP, no control for packet loss. UDP can be used to transmit single packets, but there is no guarantee that the sent packets will arrive. This is not always a disadvantage when compared to TCP's overhead for arrival checking. A message that arrives, however, is guaranteed to be error free, since the Ethernet layer includes a checksum.

The amount of packet loss is strongly dependent on the overall number of packets in the network. In state-of-the-art networks, which today are often 1 Gbit Ethernet, there is hardly any packet loss in the network itself. Losses occur mainly in the sending and receiving network cards, due to overflowing buffers. This is the case, for example, if the CPU is busy so that it cannot read the packets from the buffer in time. The more packets that are sent, the higher the chance that one is lost.

With Gbit Ethernet communication, up to 160'000 packets can be sent per second without any losses (Fig. 9 right). This means that we are able to distribute that many events per second from the simulation to other modules. Since the mobility simulation runs more than 100 times faster than wallclock-time, this results in 1'600 events per simulated second. This is not that much, however, if you keep in mind that there are 500 or more agents in the simulation which report their perception.

On a cluster with 100 Mbit Ethernet communication, the number drops to 100'000 packets per second, resulting in 1'000 events per simulated seconds. It is at this point unclear why the difference in bandwidth is not a factor of 10, as the difference between 100 Mbit and 1 Gbit would imply.

There are situations in which there is no need to retransmit lost packets. For example, if an agent reports that he is blocked in unexpected congestion (e.g. waiting for a cablecar), he needs a new route instantly. If its request is

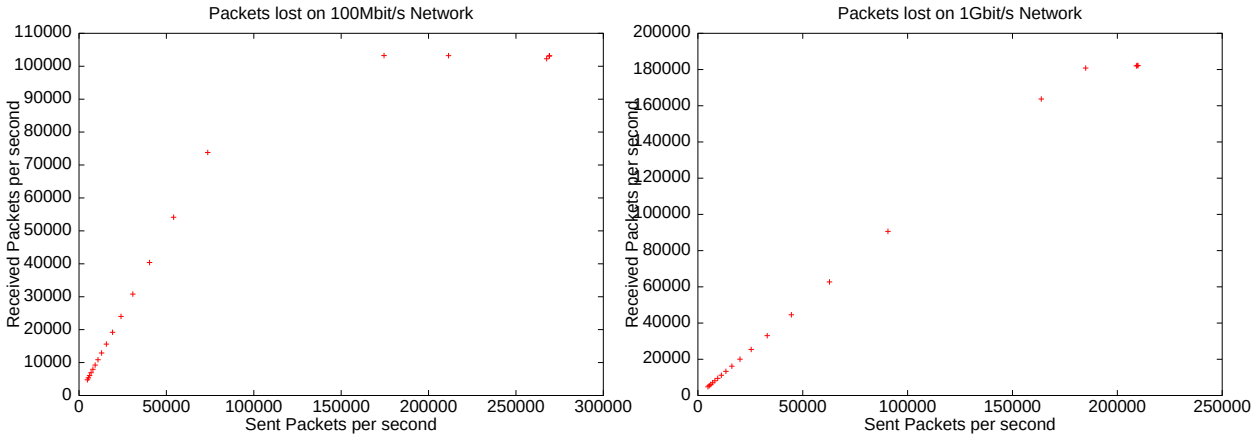

Figure 9: UDP packets sent at different rates using 100 Mbit/s (left) and 1 Gbit/s (right) networks, plotted as sent vs. received messages. Some packets are lost in the network, but most of them are lost in the overflowing buffers of the NICs. Each UDP packet can carry one event from the mobility simulation, e.g. a new position of an agent.

lost or delayed, it makes no sense for the system to buffer its request, since the agent has moved on, and the location in the original request might now be invalid. A new route computed based on the old information will be invalid as well. It is the agent's responsibility to restate its position again if it does not receive a new route after a certain time has elapsed (Gloor, 2001).

It is easier to deal with error conditions using UDP. We use UDP to transmit the agent positions from the recorder/player module to the visualizers. If the network is down for a few seconds, the simulation does not need to slow down because of lost packets. Once the viewer is back on line, it will receive the latest positions.

Multicasting Often there is a need for sending the same packet to more than one receiver. This can be achieved by opening multiple TCP connections or, more easily, by sending multiple UDP packets to the receivers. However, on large simulations, the network interface card (NIC) of the sending host quickly becomes the bottleneck, as it is unable to send out enough packets to keep the receivers fully occupied.

There is the possibility to use multicasting to send packets to every host on the local network. Multicasting is a relatively recent addition to the network standards; it is particularly useful for any kind of streaming data such as radio or television broadcasts over the network. Its advantage is that the multiplication of the packets for multiple receivers is not done by the NIC but by the network itself. This avoids the NIC bottleneck.

A drawback with multicasting is that it has, similar to UDP, no arrival control. That is, there is no feedback to the sender if all packets arrived at all destinations, or even at any destination at all. In consequence, this is not useful when message arrival needs to be guaranteed.

To live with this problem, it often is possible to implement some sort of flow control into the application. This seems to be a hard task and might introduce performance issues. But often there is no need for the full flow control available in TCP, and a lightweight solution can increase the performance substantially. This task is simplified by the fact that in most local networks packet loss is almost zero if you do not saturate the network.

Multicasting provides groups of hosts, that are referenced using special IP addresses (higher than 224.0.0.0). The sender chooses one of these groups and sends a single packet to this IP address. A receiver must explicitly join a group first, telling its NIC and the operating system to listen for packets sent to this group.

An advantage of this addressing scheme is that the sender does not need to know the IP address of the receiver. This simplifies the configuration of the system substantially. The Internet routers ensure that the packets find their way from the sender to the receivers, once they are registered to the multicast group.

Sending agent data to the viewers is an instance where multicasting is extremely effective. With our current pedestrian datasets, where 500 agents are simulated, each viewer requires 1.5MBit/s of bandwidth. By using multicasting, this bandwidth can be effectively shared between viewers, especially when they are viewing approximately the same location. As we build our datasets to a realistic scale (thousands of pedestrians), this bandwidth saving will become increasingly important.

The project presented here is a collaboration between two institutes at ETH Zürich. One of them is located more than 5 km away from where our computational cluster is. For every viewer that is connected to the simulation, extra

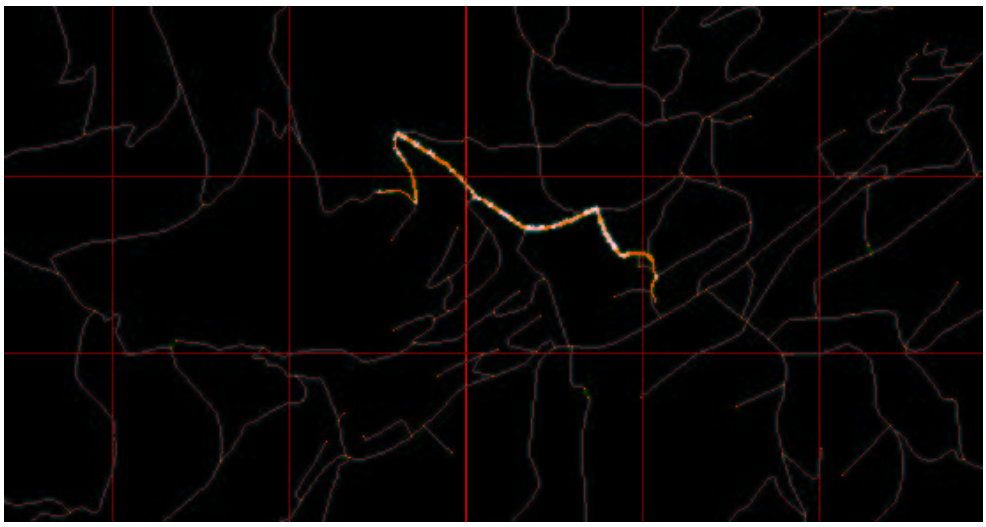

Figure 10: First run. All agents have the same destination, and at this stage, the same route.

bandwidth is needed. Using multicast, we cannot reduce the bandwidth used for one viewer. But as soon as there are multiple viewers looking at the same general area, the bandwidth remains almost constant.

File Based Communication When integrating existing modules from other simulation systems, it is often not possible or desirable to make modifications to their code. Typically, these systems use input and output files to pass data between modules.

Since we do not want to change our implementation for every "foreign" module, we use a wrapper around such modules. This wrapper reads the input from our modules as messages over the network, converts it into files understandable by the foreign module, and executes the foreign module. The corresponding output file is then read by the wrapper and reintegrated into our message based system.

Another use of such a wrapper is as a substitute for modules that have not yet been implemented. As we have not completed the activities generator module yet, we use a simple file, which contains "handmade" activity chains. A wrapper reads this file and sends the chains to the activity database module.

6 Preliminary results

A small proof-of-principle run is documented in Fig. 10 and 11. It is assumed that all agents leave in the morning from the hotel and hike to the same mountain peak. They however want to avoid each other because they want to hike in solitude. Fig. 10 shows the first run, where no hiker knew about the other hikers' intentions. Fig. 11 shows the situation after 50 iterations, where hikers have learned to spread out and avoid each other.

Fig. 12 shows a situation similar to Fig. 11, but in the 3D viewer. The hikers are visible as dark red figures in the background – to make them visible at all, they are drawn at ten times their natural size. In addition, Fig. 12 can be used to obtain an impression of the quality of the computer rendering of the scene, since it corresponds to the scene phographed in Fig. 1.

Considerably more work will be necessary to fill these elements with true real-world meaning. Progress will be reported in future papers.

Acknowledgments

We thank T. Stricker for making the CoPs cluster available to us, on which the tests with Gbit Ethernet were run. The Swiss National Science Foundation, under the program "Habitats and Landscapes of the Alps" (NFP 48) provides partial funding for C. Gloor and D. Cavens.

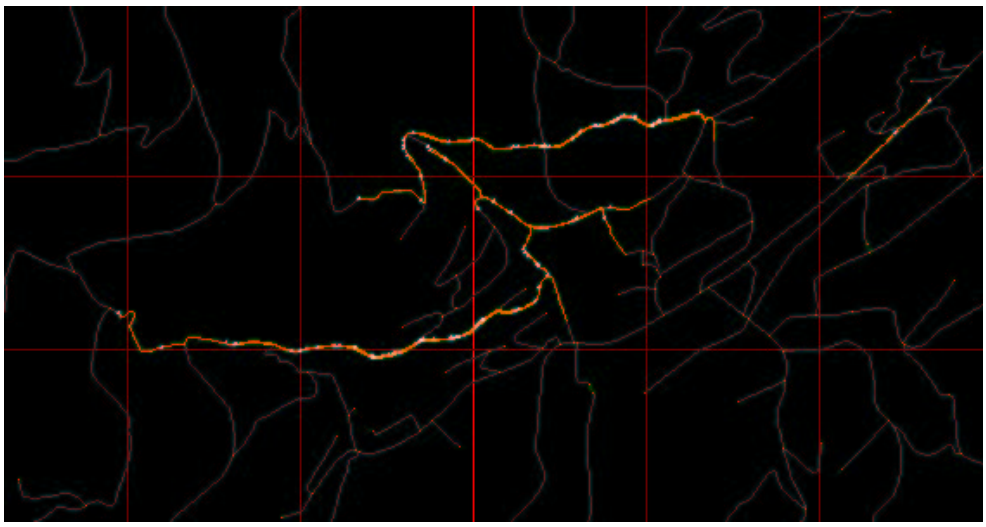

Figure 11: After 50 iterations

References

ALPSIM www page. www.sim.inf.ethz.ch/projects/alpsim/, accessed 2003. Planning with Virtual Alpine Landscapes and Autonomous Agents.

A. Babin, M. Florian, L. James-Lefebvre, and H. Spiess. EMME/2: Interactive graphic method for road and transit planning. *Transportation Research Record*, 866:1–9, 1982.

I. D. Bishop, W.-S. Ye, and C. Karadaglis. Experimental approaches to perception response in virtual worlds. *Landscape and Urban Planning*, (54):119–127, 2001.

D. Cavens and E. Lange. Hiking in real an virtual worlds. In Schrezenmayr et al, editor, *The Real and the Virtual World of Planning*, pages 145–165. 2003.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Pattern: Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series. Addison-Wesley, 2001.

R. Gimblett, editor. *Integrating Geographic Information Systems and Agent -Based Modeling Techniques*. Oxford Univerity Press, Oxford, 2002.

R. Gingold and J. Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Royal Astronomical Society, Monthly Notices*, 181, 1977.

C. Gloor. *Modelling of autonomous agents in a realistic road network (in German)*. Diplomarbeit, Swiss Federal Institute of Technology ETH, Zürich, Switzerland, 2001.

C. Gloor and K. Nagel. A pedestrian simulation for hiking in the alps. In *Proceedings of Swiss Transport Research Conference (STRC)*. Monte Verita, CH, 2002. See www.strc.ch.

D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, (407):487–490, 2000.

L. Mauron. *Pedestrians simulation methods*. Diploma thesis, Swiss Federal Institute of Technology ETHZ, 2002.

M. Meitner and T. Daniel. The effects of animation and interactivity on the validity of human responses to forest data visualizations. In B. Borland, editor, *Proceedings of Data Visualization '97, St. Louis, MO*. 1997.

MPI www page. http://www-unix.mcs.anl.gov/mpi/, accessed 2003. MPI: Message Passing Interface.

H. Müller and A. Landes. Standortbestimmung Destination Gstaad-Saanenland: Gästebefragung Sommer 2001. Schlussbericht, Forschungsinstitut für Freizeit und Tourismus der Universität Bern, 2001.

Figure 12: This is a picture of our 3-dimensional visualizer, showing the same view as figure 1.

PTV www page. Planung Transport Verkehr. See www.ptv.de, accessed 2003.

M. Rickert. *Traffic simulation on distributed memory computers*. Ph.D. thesis, University of Cologne, Cologne, Germany, 1998. Available via www.zaik.uni-koeln.de/~paper.

A. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, Inc., second, international edition, 2001.

H. Timmermans. The saga of integrated land use-transport modeling: How many more dreams before we wake up? In *Proceedings of the meeting of the International Association for Travel Behavior Research (IATBR)*. Lucerne, Switzerland, 2003. See http://www.ivt.baum.ethz.ch/allgemein/iatbr2003.html.

TRANSIMS www page. TRansportation ANalysis and SIMulation System. transims.tsasa.lanl.gov, accessed 2003. Los Alamos National Laboratory, Los Alamos, NM.

www.epm.ornl.gov/pvm/pvm_home.html. PVM: Parallel Virtual Machine. accessed 2003.