Generating Semantics-Based Trajectories of Moving Objects

Dieter Pfoser[1], Yannis Theodoridis[2]

[1] Department of Computer Science, Aalborg University, Fred. Bajersvej 7E, DK-9220 Aalborg, Denmark, pfoser@cs.auc.dk
[2] Computer Technology Institute, Akteou 11 & Poulopoulou, GR-11851 Theseio, Athens, Greece, ytheod@cti.gr

Abstract. The domain of spatiotemporal applications is a treasure trove of new types of data as well as queries. For designing and testing novel data types and access methods that will support these applications, large spatiotemporal datasets are needed. As in many cases it is not possible to obtain real datasets, either they do not exist or they are not accessible, synthetic datasets are governed by artificial parameters rather than by the (sometimes invisible) rules governing real-world behavior. In this work we show how, at least for some cases, one can generate spatiotemporal datasets that simulate real-world behavior. We illustrate example cases and translate them into appropriate calls of GSTD, a spatiotemporal data generator. The generated data is illustrated using two-dimensional snapshot pictures as well as three-dimensional (two spatial plus one temporal dimension) trajectory images.

Keywords: data generation, spatiotemporal datasets, synthetic data, moving objects.

1 Introduction

Spatiotemporal databases are an emerging technology. Example applications are fleet management, traffic management, and navigational systems, as well as the thriving developments behind mobile computing [1]. The estimates are that by the year 2003 500 million people will use mobile terminals [7]. Most of these terminals will be equipped with a GPS device, and, thus, may make their positions available to the outside, digital, and geo-referenced world. Applications, here, include spatiotemporal data mining, as well as providing geo- and time-referenced content.

New techniques comprising data structures and access methods are necessary to handle this new kind of data and queries [13],[19]. The testing of these techniques, in turn, requires the existence of datasets, either real or synthetic. Since real datasets (i) are not accessible for some applications and (ii) may not be useful for stress testing conditions, a lot of work can be found in the literature on generating synthetic data, following some specifications (desirable cardinality, distribution, etc.).

Work in the area of data generation includes [5], which aims at generating billion-record SQL databases using C programs running on a shared-nothing computer system consisting of a hundred processors, with a thousand discs. The data generated here is alphanumeric. A generator for Web data, i.e., tables and Web pages, is introduced in [14]. Soo [18] presents a temporal data generator. This work can be

seen as the one-dimensional case of spatial data generators such as presented in [6]; a web-based spatial data generator that produces sets of rectangles. The generator uses statistical distributions to compute the size, shape, and location of the data. Examples of spatiotemporal data generators are GSTD [20] and Oporto [17]. The Oporto generator uses a driving application, the modeling of fishing ships. Ships are attracted by shoals of fish while by at the same time avoiding storm areas. Fish themselves are attracted by plankton areas. Ships are moving points; shoals, plankton, and storm areas are moving regions. The GSTD generator produces moving point or rectangular region objects according to several user-defined parameters. The generator uses various distributions to control the movement and shape of the objects. Theodoridis et al. [20] present six scenarios on generating spatiotemporal datasets with respect to the desirable heading of objects, speed, etc. Although useful for access methods testing purposes [8], the original algorithm turns out to be limiting with respect to specific real-world behavior, such as directed movement or movement obstructed by buildings or alike objects, which we term *infrastructure*.

In this paper, we extend GSTD to facilitate building such scenarios. In particular, Section 2 provides the background of this work, namely the issue of spatiotemporal objects and the rationale of the GSTD generator, Section 3 proposes appropriate extensions to the algorithm in order to support directed and obstructed movements, while some example scenarios are presented in Section 4. Finally, conclusions and future work are discussed in Section 5.

2 Background

Many different objects in the real world exhibit spatiotemporal characteristics. We can distinguish those objects by, either, their spatial extent or their temporal characteristics. By temporal characteristics we refer to how objects change their properties over time, i.e., continuous or discrete. As for the spatial extent, we distinguish point objects (no spatial extent) and aerial objects (spatial extent). In the real world we can identify point objects but also aerial objects whose extent does not matter in the given application context, and, thus, is reduced to a point; as an application, consider the tracking of the continuous movement of cars, planes, people, etc. Aerial objects, on the other hand, may have an extent that changes over time. Examples of spatially continuously changing objects are storms, troop formations, grasshopper populations, tribes, etc. while examples of spatially discretely changing objects are landparcels [11].

An important aspect of a continuous movement is that it cannot be recorded in its entirety. Let us consider the example of a moving object whose position is sampled at points in time, i.e., a moving vehicle or a person, a military unit in the battlefield, etc. In order to record its movement, we would have to know its position at any time, i.e., on a continuous basis. However, GPS and telecommunications technologies only allow us to sample an object's position, i.e., to obtain the position at discrete instances of time, such as every few seconds.

2.1 Trajectories of Moving Objects

A first approach to represent the movements of objects would be to simply store the position samples. This would mean that we could not answer queries about the objects' movements at times in-between sampled positions. Rather, to obtain the

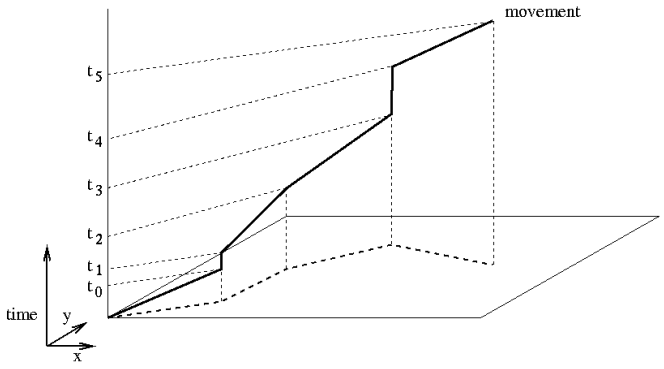


Fig. 1. A moving object trajectory

entire movement, interpolation is necessary. The simplest approach is to use linear interpolation, as opposed to other methods such as polynomial splines [2]. The sampled positions then become the end points of line segments of polylines, and the movement of an object is represented by an entire polyline in three-dimensional space. In geometrical terms, the movement of an object is termed a *trajectory* (we will use "movement" and "trajectory" interchangeably). The solid line in Fig. 1 represents the movement of a point object. Space and time are combined to form one (three-dimensional) coordinate system. The dashed line shows the projection of the movement on the plane of x- and y- coordinates [10]. The issue of interpolation becomes more complex when we consider aerial objects. In this case, interpolating instances means to interpolate the shape as well.

Interpolating trajectories raises questions about the uncertainty associated with a particular representation. Did Pfoser and Jensen [10] describe uncertainty as related to trajectory data, so do Pfoser and Tryfona [12] address the general issue of uncertainty and imprecision in temporal, spatial, and spatiotemporal contexts.

2.2 GSTD Overview

A fundamental issue in generating synthetic spatiotemporal datasets is the definition of a set of parameters that control the evolution of spatial objects. GSTD [20] addresses the following:

- *duration of an object instance*, which involves change of timestamps between consecutive instances,
- *shift of an object*, which involves change of spatial location (in terms of center point shift), and
- *resizing of an object*, involves change of an object's size (only applicable to aerial objects).

Since the evolution (or 'history') of a spatiotemporal object, identified by its *o_id*, is represented by a set of instances (*o_id*, s_i, t_i), where s_i is the location of object *o* at instant t_i (s_i and t_i are called *spacestamp* and *timestamp*, respectively), the goal to be reached is the calculation of its consecutive instances, starting from an initial instance (*o_id*, s_1, t_1). It is also assumed that the spatial workspace of interest is the unit square $[0,1)^2$ and that time varies from 0 to 1 (i.e., the unit interval). For

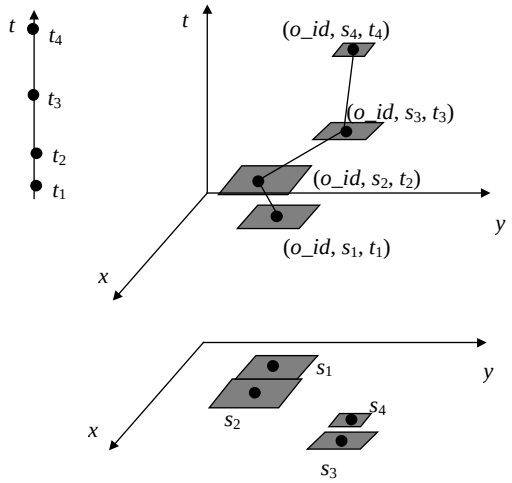


Fig. 2. Consecutive instances of a time-evolving object and the corresponding projections

illustration reasons, four instances of a moving object are shown in Fig. 2 together with the corresponding projections on the spatial plane and the temporal axis [20].

The *shift*, the *duration*, and the *resizing* of an object's instance are represented by the functions:

- **`duration`**`(o_id,interval,curr_tstamp,new_tstamp)`
- **`shift`**`(o_id,Δc[],curr_sstamp_c,new_sstamp_c)`
- **`resizing`**`(o_id,Δext[],curr_sstamp_ext[],new_sstamp_ext[])`

which calculate the new timestamp and the new spacestamp's center and extent, called `new_tstamp` (a numeric value), `new_sstamp_c` (a 2-dimensional point), and `new_sstamp_ext[]` (an array of 2 intervals), respectively, of an object identified by its `o_id`, as the sums of the respective current values and the respective parameters, namely `interval`, `Δc[]`, and `Δext[]`.[1]

Not all generated instances of objects are within the workspace. GSTD manipulates *invalid* instances according to one among three alternative approaches: (i) the '*radar*' approach, where coordinates remain unchanged, although falling beyond the workspace, (ii) the '*adjustment*' approach, where coordinates are adjusted (according to linear interpolation) to fit the workspace, and (iii) the '*toroid*' approach, which assumes a toroidal workspace, as such once an object traverses one edge of the workspace, it enters back in the 'opposite' edge.

In summary, GSTD gets several user-defined parameters as input:

- `N` and `D` correspond to the initial cardinality and density (i.e., the ratio of the sum of the areas of data rectangles over the workspace area) of the dataset,
- `starting_id` corresponds to the initial identification number of every object in the dataset,

[1] The format of the functions to compute the parameters interval, Δc[], and Δext[] are as follows: interval (distr_t(),min_t,max_t) , Δc[](distr_c(),min_c,max_c), Δext[](distr_ext(),min_ext,max_ext).

- `numsnapshots` corresponds to the time resolution of the workspace,
- `distr_init()`, the inital distribution of centers
- `distr_t()`, `min_t`, and `max_t` correspond distribution and the domain of the `interval` parameter,
- `distr_c()`, `min_c[]`, and `max_c[]` correspond to the distribution and the domain of the Δ`c[]` parameter,
- `distr_ext()`, `min_ext[]`, and `max_ext[]` correspond to the distribution and the domain of the Δ`ext[]` parameter.

Several GSTD parameters are controlled by statistical distributions. For example, the initial spatial distribution `distr_init` is a choice among *uniform*, *gaussian*, and *skewed*. For more details on GSTD, please refer to the original paper [20] and the GSTD Web interface [21].

3 Introducing Semantics in GSTD

Spatiotemporal movement characteristics make it necessary, to add new features to the current GSTD implementation. Introducing a *new parameter* allows us to create more realistic object movements. Additionally, the *infrastructure concept* permits the creation of trajectories stemming from objects moving in an obstructed environment, e.g., cities. We present an infrastructure generator as well as modifications to the GSTD algorithm to consider this information during trajectory generation.

3.1 Extending the GSTD Algorithm

The majority of objects in the real world do not move according to statistical parameters but, rather, move intentionally. For example, the movement of a car is guided by the will of its driver, which could be to reach many destinations consecutively. However, the path to such destinations is not linear but according to many factors such as the road network, traffic conditions etc. Overall, however, a movement towards a destination is directed. With the current GSTD implementation such a movement that is overall directed and locally undirected cannot be simulated. Currently, the spatial **`shift`** parameter, which is responsible for a change of direction, is changed for each new timestamp, i.e., after every `interval`. Thus, by using a broad Δ`c[]` parameter the change in **`shift`** is larger, and as a consequence the objects move "nervously". Alternatively, by using a narrow Δ`c[]` parameter all objects move in one direction during the whole time frame. To find a middle ground, i.e., objects having directed movements for longer periods of time, we introduce a new parameter, called `dinterval`. During a `dinterval`, Δ`c[]` is kept constant. Was the Δ`c[]` parameter so far only governed by a distribution function specified as input to GSTD, does it now become a more complex function with the following format:

- Δ`c[](dinterval,new_tstamp,distr_c(),min_c[],max_c[]`

The format of the new parameter is `dinterval(distr_dt(),min_dt,max_dt)`, thus introducing three extra input parameters, namely `distr_dt()`, `min_dt`, and `max_dt`, corresponding to the distribution and the domain of `dinterval`.

3.2 Clustered Movements

Different groups of moving objects exhibit different kinds of behavior. Let us consider the movements of cars in the morning hours. Parents are dropping their kids to schools in the suburbs, people drive to their jobs downtown, taxis move around in the city center, and people drive to shopping malls. Here, clusters of people exhibit similar movement characteristics.

GSTD indirectly supports clustered datasets. To achieve that, sets of objects moving with varying speed, agility and direction are just to be generated and then appropriately combined into a single dataset.

3.3 Infrastructure

An important assumption in GSTD was that the objects are unobstructed in their movements with the single exception of a workspace boundary. However, in real-world scenarios, there may exist spatial objects that hinder other objects in their movement. We term this set of objects *infrastructure*. Like moving object datasets, infrastructure can be composed by either real or synthetic data.

Focusing on synthetic data, a popular generator for spatial data (used so far in spatial database work) has been implemented. The input parameters of the generator are the following:

- `N`, i.e., the number of rectangles,
- `X` and `Y`, i.e., the (maximum) size of the rectangles in x- and y- dimension, respectively, and
- `extent_type`, i.e., whether the size of rectangle is constant and equal to `X` or `Y` (`extent_type=c`) or variable up to `X` or `Y` (`extent_type=v`).

The output consists of a set of non-overlapping rectangles with their centers obeying a uniform distribution (other popular distributions, such as gaussian or skewed, have been implemented as well).

Fig. 3 depicts an infrastructure file containing 20 rectangles, of a variable box size of up to 20% in both dimensions. The pseudo-code of the infrastructure generator is presented in the Appendix.

When generating moving objects data, each new instance has to be *outside* each of the *infrastructure* objects. To achieve that, we employ a similar approach to what the GSTD algorithm uses in treating objects that are outside the workspace (cf. Section 2.2). From the three originally proposed approaches, it is only the radar and the adjustment approach that can be applied to infrastructure. The application of the *radar* approach is straightforward; the coordinates of the new instance remain unchanged, although intersecting with infrastructure objects. Most important in this context is the '*adjustment*' approach, where coordinates are adjusted (according to linear interpolation) to be outside infrastructure objects.

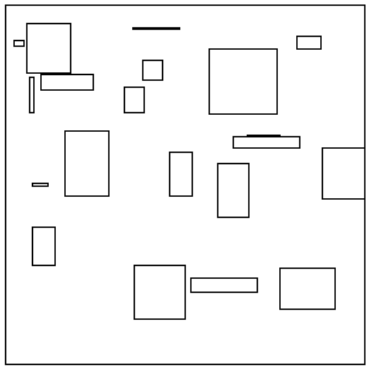


Fig. 3. Infrastructure (a set of rectangles)

In this context, do note that we are only dealing with static obstacles, i.e., the infrastructure. Examples of dynamic (or moving) obstacles are chunks of slow moving traffic, motor cades, special transport blocking roads, etc. Those objects could be generated using the GSTD tool itself. However, in such a case the whole environment becomes more complex[2].

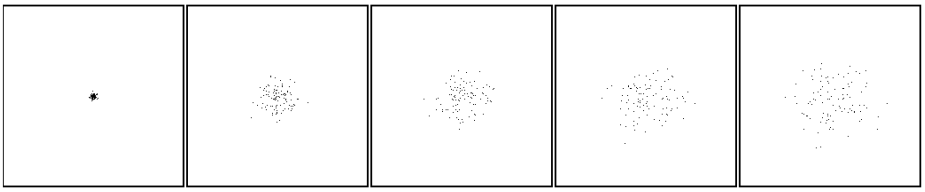
4 Complex Scenarios on Moving Object Trajectories

In this section we show how to generate trajectories of moving objects using GSTD to simulate real-world scenarios. We present various scenarios and depict a proper GSTD parameter setting. The generated data is illustrated using a two-dimensional temporal snapshot representation as well as a three-dimensional polyline representation.

An initial goal of data generators is the control of basic object properties, such as speed, heading, and agility of objects. Subsequently, we combine the groups of objects showing similar movements characteristics to form clusters. The last set of movement examples is characterized by infrastructure hindering the free movement of objects. Here, we use a random set of rectangles to represent static real-world objects, e.g., the movements of cars would be obstructed by buildings, rivers, parks, etc.

For all the examples to follow, we made some assumptions: (a) The GSTD generator can generate both, point objects and aerial objects. The extent of an object is of no importance in the present context, thus in the sequel we will only deal with *point objects*. (b) The GSTD generator employs several approaches of how to adjust the positions of objects leaving the workspace. As already mentioned, we have chosen to experiment with the *adjustment* approach. (c) We assume that the spatial workspace of interest is the unit square $[0,1)^2$ and that time varies from 0 to 1 (i.e., the unit interval). Finally, (d) the format in which the distributions are specified in the following examples is identical to the actual input stream used in GSTD, i.e., `r` stands for a *uniform* distribution, `g_0.5_0.1` stands for a *gaussion* distribution with a mean = 0.5 and a standard deviation = 0.1, and `s(1)` stands for a *skewed* distribution.

[2] Algorithms determining whether a moving object intersects a moving obstacle fall in the category of a *spatiotemporal join*, which deserves careful investigation [19].



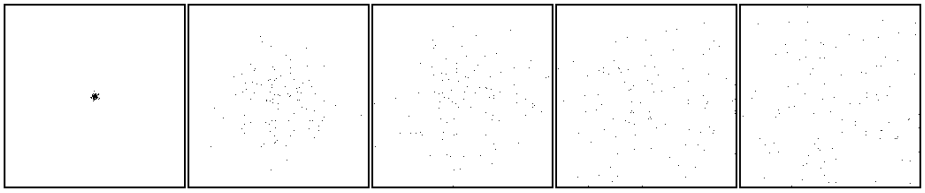
(a) 2D snapshots

(b) 3D trajectories

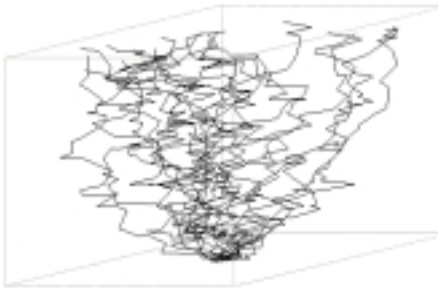
```
2D: N = 100; 3D: N = 20
D = 0
starting_id = 1
numsnapshots = 1000
distr_init()= g_0.5_0.01

distr_t() =      distr_dt() =
g_0_0.03         g_0_0
min_t = 0        min_dt =0
max_t = 1        max_dt = 0

distr_c() = r    distr_ext() = r
min_c[] =        min_ext[] = 0,0
-0.03,-0.03      max_ext[] = 0,0
max_c[] =
0.03,0.03
```

(c) GSTD parameters

Fig. 4. Slow moving objects



(a) 2D snapshots



(b) 3D trajectories

```
2D: N = 100; 3D: N = 20
D = 0
starting_id = 1
numsnapshots = 1000
distr_init()= g_0.5_0.01

distr_t() =      distr_dt() =
g_0_0.03         g_0_0
min_t = 0        min_dt =0
max_t = 1        max_dt = 0

distr_c() = r    distr_ext() = r
min_c[] =        min_ext[] = 0,0
-0.08,-0.08      max_ext[] = 0,0
max_c[] =
0.08,0.08
```

(c) GSTD parameters

Fig. 5. Fast moving objects

4.1 Movement Parameters

The first set of examples demonstrates of how basic characteristics of objects such as speed, agility, and direction can be manipulated using the parameters of the GSTD generator.

Varying objects speed

Objects do not move with constant speed, e.g., cars are parked, stopped at traffic lights, move in cities, or on highways. Further, different kinds of objects have different speed capabilities. Cyclists can not move as fast as cars, different kinds of ships move at different speeds, e.g., high-speed ferries and fishing boats. The scope of this section is to illustrate the connection between the real-world *speed* characteristic and the relevant GSTD parameters.

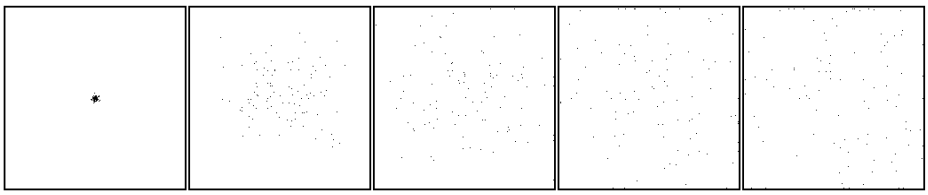
A varying objects speed can be achieved by varying the `distr_c()`, `min_c[]`, and `max_c[]` parameters in GSTD. This means that during a temporal interval the center of an object can move by as much as the distribution permits. The distribution can be any one supported by GSTD. In the example shown in Fig. 4(a), we chose `distr_c()` to be uniform, with `min_c[x,y]` = (-0.3, -0.3) and `max_c[x,y]` = (0.3, 0.3). However, do note that these parameters have to be seen in connection with the given temporal interval determined by `distr_t()`, `min_t`, and `max_t`.

The snapshots of Fig. 4(a) are taken at timestamps 0 (initial distribution), 0.25, 0.50, 0.75, and 1 (final distribution). Fig. 4(b) gives a 3D representation of the trajectories. The two spatial dimensions are in the horizontal plane, whereas the temporal dimension is in the vertical direction. The cube represents the extents of the workspace. To give a better visual impression of the movement, the spacestamps are connected using linear interpolation to form polylines.

Fig. 5 illustrates faster moving objects. As before, we chose `distr_c()` to be uniform, but with `min_c[x,y]` = (-0.8, -0.8) and `max_c[x,y]` = (0.8, 0.8). The possible range of deviation is larger. One can see that after an initial dispersion phase, the distribution of objects is almost uniform throughout the workspace. This is in contrast to the previous example for slower movement, where the initial distribution highly affects the overall positioning of the objects. Fig. 5(b) shows a rather nervous movement of objects, i.e., a frequent change of heading. With the next set of examples we show how to smoothen directional movement.

Agility of objects

To generate trajectories for more directed movements, an additional GSTD parameter, `distr_dt()`, was introduced in Section 3.1. For the example shown in Fig. 6 we used `distr_dt()` = g_0.2_0.1, i.e., a gaussian distribution with a mean = 0.2 and a standard deviation = 0.1, with `min_dt` = 0 and `max_dt` = 1. In general, the larger a `dinterval`, the longer an object moves into a specified direction, i.e., if the average `dinterval` = 0.2, there are five changes of the general direction during the whole time frame. The other parameters are the same as in the example in Fig. 4, depicting slow movement.



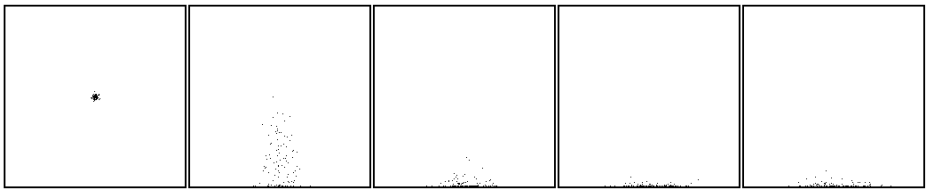
(a) 2D snapshots

(b) 3D trajectories

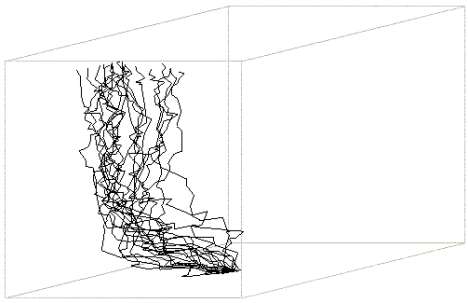
```
2D: N = 100; 3D: N = 20
D = 0
starting_id = 1
numsnapshots = 1000
distr_init()= g_0.5_0.01

distr_t()=      distr_dt()=
g_0_0.03        g_0.2_0.1
min_t = 0       min_dt = 0
max_t = 1       max_dt = 1

distr_c()= r    distr_ext()= r
min_c[] =       min_ext[] =
0.03,-0.03      0,0
max_c[] =       max_ext[] =
0.03,0.03       0,0
```

(c) GSTD parameters

Fig. 6. Smoothly moving objects



(a) 2D snapshots



(b) 3D trajectories

```
2D: N = 100; 3D: N = 20
D = 0
starting_id = 1
numsnapshots = 1000
distr_init()= g_0.5_0.01

distr_t()=      distr_dt()=
g_0_0.03        g_0_0
min_t = 0       min_dt = 0
max_t = 1       max_dt = 0

distr_c()= r    distr_ext()= r
min_c[] =       min_ext[] = 0,0
-0.03,-0.1      max_ext[] = 0,0
max_c[] =
0.03,0.03
```

(c) GSTD parameters

Fig. 7. A directed movement towards “south”

The overall effect of the dinterval parameter is that a chosen direction is kept for a larger time interval. Thus, in comparing Fig. 4 and Fig. 6, one can easily see that, now, the trajectories appear less "nervous," and, at the same time, objects move away from their initial positions. In contrast to this, the objects in Fig. 4 move around their inital positions; the many changes of Δext[], with a mean = 0, cancel each other.

Preferred direction

Direction is an important characteristic of a movement. To achieve directional movement we, again, manipulate the min_c[] and max_c[] parameters. Consequently, we can adjust four parameters, -x, x, -y, and y, to bias movement. In Fig. 7, a directed movement is illustrated. The parameter setting is distr_c() to be uniform and min_c[x,y] = (-0.03, -0.1) and max_c[x,y] = (0.03, 0.03), i.e., the distribution in the y direction is biased towards the negative direction. General directional movements can be achieved by combining smaller/larger deviation values.

Summary

The objective of this section was to show the correspondence between movement characteristics and GSTD parameters (cf. Table 1).

Parameter/Characteristic	Speed	Agility	Direction
interval: distr_t(), min_t, max_t	√		
dinterval: distr_dt(), min_dt, max_dt		√	
Δc[]: distr_c(), min_c[], max_c[]	√		√

Table 1. GSTD parameters and their influence on movement characteristics

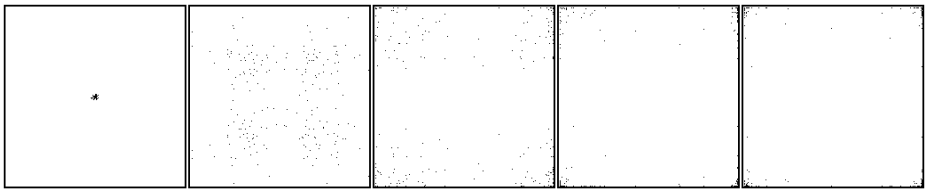
The speed of objects is influenced according to the fraction Δc[]/interval. Thus, in the examples, we specify ranges for both parameters, i.e., distributions. Slower and faster moving objects are characterized by allowing a smaller/larger range for Δc[] by at the same time keeping interval constant.

The agility of objects is governed by the new dinterval parameter; the larger the range, the fewer times the objects change their general direction. The immediate direction is governed by the Δc[] parameter; its values determine the direction, e.g., if Δc[x] = 0.03 and Δc[y] = 0.03 the object moves into a "Northeastern" direction.

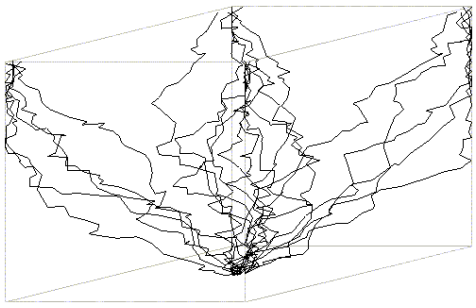
4.2 Clustered Movement

In the previous section we assumed that the movement of objects within one example is determined by one set of parameters. In this section, we show how to use the capabilities of GSTD to generate clustered movement of objects within one dataset.

The key parameter for this purpose is called starting_id. By generating various datasets with different movement characteristics and appropriate starting_id values, we can then use an simple external concatenation program to combine them. Fig. 8 visualizes the so generated dataset of four clusters of objects moving into four different directions. In Fig. 8(b) one can observe that the objects "crawl" up the



(a) 2D snapshots



(b) 3D trajectories

2D: **N** = 200; 3D: **N** = 20
D = 0
starting_id = 1/6/11/16
numsnapshots = 1000
distr_init()= g_0.5_0.01

distr_t()=	**distr_dt()**=
g_0_0.03	g_0_0
min_t = 0	**min_dt** = 0
max_t = 1	**max_dt** = 0

Distr_ext()= r
min_ext[] = 0,0
max_ext[] = 0,0

distr_c()= r
4 clusters:
1st **min_c[]** = -0.065,-0.065
max_c[] = 0.03,0.03
2nd **min_c[]** = -0.03,-0.065
max_c[] = 0.065,0.03
3rd **min_c[]** = -0.03,-0.03
max_c[] = 0.065,0.065
4th **min_c[]** = -0.065,-0.03
max_c[] = 0.03,0.065

(c) GSTD parameters

Fig. 8. Clustered movements towards the corners

corners. The movement parameters, listed in Fig. 8(c), direct the objects to the corners. Because of the adjustment approach they are forced "inside" when they try to leave the workspace, but try again to leave because of the parameter setting.

4.3 Obstructed Movement

In Sections 4.1 and 4.2 we assumed that the objects could move freely in the workspace. In this section, we introduce infrastructure, i.e., static spatial objects that hinder the objects in their movement (cf. Section 3.3).

Table 2 shows the parameters to generate two infrastructure datasets containing 20 and 40 objects, respectively. The size of the objects is limited to 20% of the size of the workspace per dimension.

For the first example of sparse infrastructure, we generated the datasets visualized in Fig. 9. In this example, we chose the "adjustment" approach, i.e., when GSTD computes a new position that lies inside an infrastructure object, "adjustment" moves

it “outside” the obstacle, where “outside” in this context means on the border of the obstacle (cf. the first snapshot in Fig. 9(a)).

The movement parameters in this example are the same as in Section 0. In comparing Fig. 9 with Fig. 6, we can observe that the infrastructure deflects the objects towards the upper right corner. Also, by comparing Fig. 9(b) and Fig. 6(b), we conclude that although the movements are similar (the trajectories resemble each other), the trajectories influenced by infrastructure are edgier.

In the visualization of the trajectories in Fig. 9(b) we use linear interpolation to connect the spacestamps, i.e., although it is guaranteed that all the spacestamps lie outside the infrastructure objects, this does not necessarily have to be the case for the interpolating line segments as outlined.

Fig. 10 visualizes a dataset with the same movement parameters as before, but using the dense infrastructure dataset containing 40 objects (cf. Table 2).

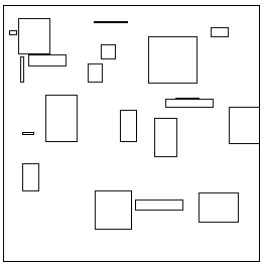
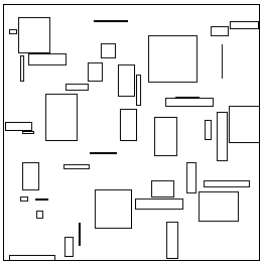
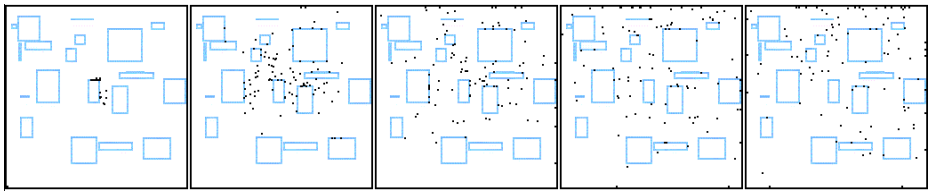
Example	Parameters	Illustration
Sparse infrastructure	**N** = 20 **X** = 0.2 **Y** = 0.2 **extent_type** = v	
Dense infrastructure	**N** = 40 **X** = 0.2 **Y** = 0.2 **extent_type** = v	

Table 2. Infrastructure parameters

5 Conclusions and Future Work

The paper presents several examples of how real-world movement characteristics can be translated into appropriate parameters of a spatiotemporal data generator. In this work, we used the GSTD tool to generate trajectories of moving point objects. To accommodate (a) different degrees of object agility and (b) clustered movement, we introduced additional GSTD parameters over the original implementation. Furthermore, movement in the real world is not free of obstacles, here termed *infrastructure*. Thus, as a third extension, we presented modifications to the GSTD tool to accommodate infrastructure during data generation.



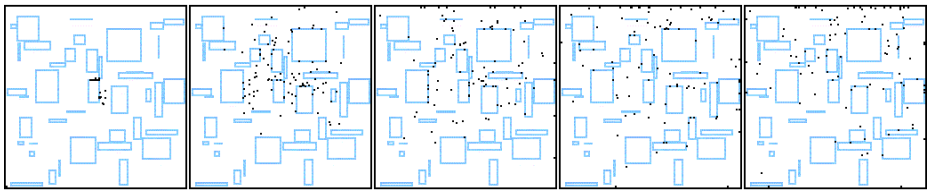
(a) 2D snapshots

(b) 3D trajectories

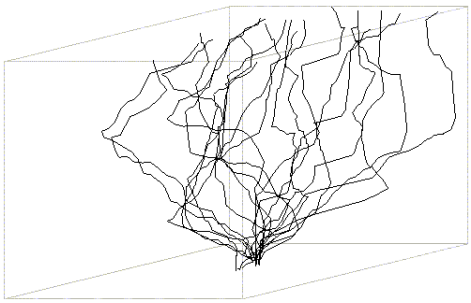
```
2D: N = 100; 3D: N = 20
D = 0
starting_id = 1
numsnapshots = 1000
distr_init()= g_0.5_0.01

distr_t()=      distr_dt()=
g_0_0.03        g_0.2_0.1
min_t = 0       min_dt = 0
max_t = 1       max_dt = 1

distr_c()= r    distr_ext()= r
min_c[] =       min_ext[] = 0,0
0.03,-0.03      max_ext[] = 0,0
max_c[] =
0.03,0.03
```

(c) GSTD parameters

Fig. 9. Obstructed movement, sparse infrastructure



(a) 2D snapshots



(b) 3D trajectories

```
2D: N = 100; 3D: N = 20
D = 0
starting_id = 1
numsnapshots = 1000
distr_init()= g_0.5_0.01

distr_t()=      distr_dt()=
g_0_0.03        g_0.2_0.1
min_t = 0       min_dt = 0
max_t = 1       max_dt = 1

distr_c()= r    distr_ext()= r
min_c[] =       min_ext[] = 0,0
0.03,-0.03      max_ext[] = 0,0
max_c[] =
0.03,0.03
```

(c) GSTD parameters

Fig. 10. Obstructed movement, dense infrastructure

A major part of the work is dedicated to showing different movement examples, such as varying speed, agility, direction, as well as clustered and obstructed movement. The various examples are visualized using a two-dimensional snapshot plus a three-dimensional representation of the trajectories.

This work points to several future research directions. An immediate goal is, to extend the GSTD tool provided through a Web interface [21] to accommodate the presented modifications.

Spatiotemporal data generators are parameterized using mathematical, better statistical concepts. On the other hand, several real-world movements are governed by human behavior. An interesting approach would be to parameterize these behavioral patterns as they can be used for data generation. A somewhat related approach to structuring space can be found in [15]. Particular semantic concepts that could be studied in this context are *closeness*, e.g., objects that lie close to each other or close to an obstacle are treated differently, and *attraction*, e.g., objects change direction when they approach a target. Since these concepts raise neighborhood issues, work in nearest-neighbor [16] as well as closest-pair queries [3] should be integrated into GSTD.

As we outlined in Section 2.1, one might use linear interpolation to estimate the position in between known instances. Consequently, one cannot assume that the "real" trajectory of a moving object coincides with the polyline interpolation. In other words, every new instance is adjusted with respect to the previous instance and the underlying infrastructure but it could be further adjusted such that even the interpolating line segment does not intersect with any infrastructure object. The adjustment algorithm in the current GSTD implementation does not consider this kind of semantics and this is a task of near future work. Also, the current GSTD implementation uses only static infrastructure. As outlined in Section 3.3, infrastructure can be dynamic and extending the GSTD tool towards this direction would be a logical step.

Overall, data generation initiated in [20] is just but one tool in the larger framework of benchmarking access methods and query processing techniques. In the area of spatiotemporal databases, this effort started within the CHOROCHRONOS project [4] and is still in progress [8],[9].

Acknowledgements

This research was supported by the CHOROCHRONOS project, funded by the European Commission DG XII Science, Research and Development, as a Network Activity of the Training and Mobility of Researchers Program, contract no. ERBFMRX-CT96-0056. The first author is further supported in part by the Danish Technical Research Council through grant 9700780, the Danish Natural Science Research Council through grant 9400911, and the Nykredit Corporation.

References

1. Barbará, D.: Mobile computing and databases – a survey. *IEEE Transactions of Knowledge and Data Engineering*, 11(1), pp. 108-117, 1999.
2. Bartels, R., Beatty, J., and Barsky, B.*: An Introduction to Splines for Use in Computer Graphics & Geometric Modeling*. Morgan Kaufmann Publishers, Inc., 1987.
3. Corral, A., Manolopoulos, Y., Theodoridis, Y., and Vassilakopoulos, M.: Closest Pair Queries in Spatial Databases. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, to appear, 2000.
4. Frank, A., Grumbach, S., Güting, R., Jensen, C., Koubarakis, M., Lorentzos, N, Manolopoulos, Y., Nardelli, E., Pernici, B., Schek, H.-J., Scholl, M., Sellis, T., Theodoulidis, B., and Widmayer, P.: Chorochronos: a research network for spatiotemporal database systems. *SIGMOD Record*, 28(1), pp. 110-114, 1999.
5. Gray, J., Sundaresan, P., Englert, S., Backlawski, K., and Weinberger, P.: Quickly generating billion-record synthetic databases. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pp. 243-252, 1994.
6. Günther, O., Oria, V., Picouet, P., Saglio, J.-M., and Scholl, M.: Benchmarking spatial joins *À La Carte*. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pp. 32-41, 1998.
7. Karppinen, J.: Wireless multimedia communications: a Nokia view. In *Proceedings of the Wireless Information Multimedia Communications Symposium*, Aalborg University, 1999.
8. Nascimento, M., Silva, R., and Theodoridis, Y.: Evaluation of access structures for discretely moving points. In *Proceedings of the International Workshop on Spatio-Temporal Database Management*, pp.171-188, 1999.
9. Papadopoulos, A., Rigaux, P., and Scholl, M.: A performance evaluation of spatial join processing strategies. In *Proceedings of the 6th International Symposium on Spatial Databases*, pp. 286-307, 1999.
10. Pfoser, D. and Jensen, C.: Capturing the uncertainty of moving-object representations, In *Proceedings of the 6th International Symposium on Spatial Databases*, pp. 111-132, 1999.
11. Pfoser, D. and Tryfona, N.: Requirements, definitions, and notations for spatiotemporal application environments. In *Proceedings of the 6th International Symposium on Advances in Geographic Information Systems*, pp. 124-130, 1998.
12. Pfoser, D. and Tryfona, N.: Fuzziness and uncertainty in spatiotemporal applications. Under submission, 2000.
13. Pfoser, D., Theodoridis, Y., and Jensen, C.: Novel approaches in query processing for moving objects data. Under submission, 2000.
14. Priyadarshini, P., Qin, F., Lim, E., and Ng, W.: WEDAGEN - a synthetic Web database generator. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, pp. 744-748, 1999.
15. Raubal, M., Egenhofer, M., Pfoser, D., and Tryfona, N.: Structuring space with image schemata: wayfinding in airports as a case study. In *Proceedings of the International Conference on Spatial Information Theory*, pp. 85-102, 1998.

16. Roussopoulos, N., Kelley, S., and Vincent, F.: Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD Conference on Management of Data*, pp. 71-79, 1995.
17. Saglio, J.-M. and Moreira, J.: Oporto: A realistic scenario generator for moving objects. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, pp. 426-432, 1999.
18. Soo, M.: Generating temporal data: a white paper. Department of Computer Science and Engineering, University of South Florida, 1997.
19. Theodoridis, Y., Sellis, T., Papadopoulos, A., and Manolopoulos, Y.: Specifications for efficient indexing in spatiotemporal databases", In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pp. 123-132, 1998.
20. Theodoridis, Y., Silva, R., and Nascimento, M.: On the generation of spatiotemporal datasets. In *Proceedings of the 6th International Symposium on Spatial Databases*, pp. 147-164, 1999.
21. Theodoridis, Y. and Nascimento, M.: Generating spatiotemporal datasets on the WWW. Under submission, 2000. GSTD Web interface in http://www.cti.gr/RD3/GSTD/. Valid as of January 2000.

Appendix

The Infrastructure Generator

```
Generate_Infrastructure algorithm
Input:    values N, flag constant
          arrays extent[]
Output:   instance(low[], high[])

begin
01 for N do
02    Set low[] = random(0,1)
03    if constant = 1 then //constant size
04       Set high[] = low[] + extent[]
05    else //variable size
06       Set delta[] = random(extent[])
07       Set high[] = low[] + delta[]
08    end-if
09    if low >= 0 and high <= 1 then //within workspace
10       output(low[],high[])
11    end-if
12 end-for
end.
```

The modified GSTD Algorithm [20]

```
Generate_Spatio_Temporal_Data algorithm
Input:    values N, starting_id, numsnapshots, D,
          min_t, max_t, min_dt,max_dt
          arrays min_c[], max_c[], min_ext[], max_ext[]
          distributions distr_init, distr_t, distr_dt, distr_c,
distr_ext
Output:   instance (id, t, l_l_point, u_r_point), validity_flag

begin
01  for each id in range [starting_id .. N+starting_id] do
02     /*initialization phase*/
03     Set c[] = RNG(distr_init(), 0, 1), ext[] = extent(N, D)
04     Set t = 0, active = TRUE
05  end-for
06  Set step = 1 / numsnapshots
07  for each id in range [starting_id .. N+starting_id] do
08     /*loop phase*/
09     Set next_snapshot = step
10     while active do
11        /* calculate delta-values and new instances */
12        Set interval = RNG(distr_t(), min_t, max_t)
13        if Σ interval > dinterval
14           Set dinterval = RNG(distr_dt(), min_dt, max_dt)
15           Set Δc[] = RNG(distr_c(), min_c[], max_c[])
16           Set Δext[] = RNG(distr_ext(), min_ext[], max_ext[])
17           reset Σ interval
18        end-if
19        Set old_instance = instance
20        update_instance(instance)
21        /* check instances and output */
22        if t > 1 then
23           active = FALSE
24           output(old_instance, current[i], next_snapshot)
25        else /*check instance validity and output*/
26           Set validity_flag = valid(instance)
27           if validity_flag = FALSE and approach ≠ 'radar'
   then
28              adjust_coords(instance, approach)
29           end-if
30           if t > next_snapshot then
31              output(old_instance, current[i], next_snapshot)
32           end-if
33        end-if
34     end-while
35  end-for
end.
```