# Generating Spatiotemporal Datasets on the WWW

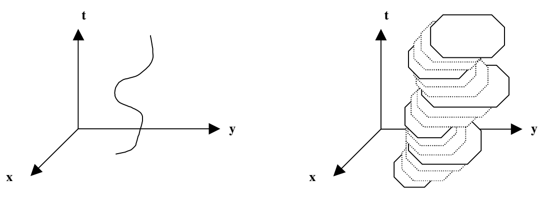
**Yannis Theodoridis**
Computer Technology Institute
Patras, Hellas
yannis.theodoridis@cti.gr

**Mario A. Nascimento**
Dept. of Computing Science
University of Alberta, Canada
mn@cs.ualberta.ca

**Abstract**. Efficient storage, indexing and retrieval of time-evolving spatial data are some of the tasks that a Spatiotemporal Database Management System (STDBMS) must support. Aiming at designers of indexing methods and access structures, in this article we review the GSTD algorithm for generating spatiotemporal datasets according to several user-defined parameters, and introduce a WWW-based environment for generating and visualizing such datasets. The GSTD interface is available at two main sites: `http://www.cti.gr/RD3/GSTD/` and `http://www.cs.ualberta.ca/~mn/GSTD/`.

## 1. Introduction

Emerging spatial applications deal with objects whose position, shape and size change over time, i.e., the so-called spatiotemporal objects. Real world examples include vehicle or human trajectories archival data, fire front monitoring, flight simulators, weather forecast, etc. According to [11], a *spatiotemporal object*, identified by its *o_id,* is a time-evolving spatial object, i.e., its evolution (or 'history') is represented by a set of instances (*o_id*, $s_i$, $t_i$), where $s_i$ is the location of the object at instant $t_i$ ($s_i$ and $t_i$ the object instance *spacestamp* and *timestamp*, respectively). Following that definition, a 2D time-evolving point (region) is represented by a line (respectively, solid) in 3D space (Figure 1).




(a) moving point (b) moving region

Figure 1. Example of 2D time-evolving spatial objects

For all the above applications, efficient indexing is a requirement due to the high volume (and complexity) of data involved. The solutions could either be extensions of existing spatial or temporal access methods [13, 5, 6] or novel proposals [4, 9]. In any case, designers should evaluate their techniques under extensive experimentation using real and synthetic data. Zobel et al. [14] suggest that "*experiments of indexing techniques should be based on benchmarks such as standard sets of data and queries*". Following that guideline, we are building a benchmarking environment for spatiotemporal access methods (STAMs) that includes the following:

a) *a module that generates synthetic data and query sets*, which would cover a variety of real life examples, plus *a repository of real datasets*,
b) *a visualization tool* that could be able to visualize datasets and structures, for illustrative purposes,
c) *a collection of STAMs* for experimentation purposes, and
d) *a database of experimental results* concerning evaluation of STAMs.

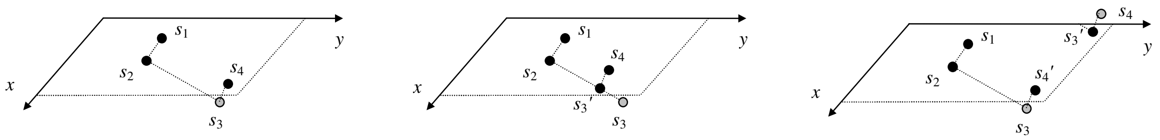
In this paper we focus on items (a) and (b). We discuss the GSTD algorithm for generating spatiotemporal data, originally proposed in [12], and a WWW-based interface recently developed to facilitate its use.

## 2. The GSTD Rationale

In order for the user of a benchmarking environment to conduct an extensive series of experiments under a variety of conditions, he/she should be able to generate datasets by tuning an appropriate set of parameters and distributions. A fundamental issue on generating synthetic spatiotemporal datasets is the definition of a *rich set of parameters* that control the evolution of spatial objects. Towards this goal, we have addressed the following three operations:

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

In [12] we proposed the GSTD algorithm for building sets of moving point or rectangular objects. For each object *o*, GSTD generates tuples of the




(a) 'radar' (b) 'adjustment' (c) 'toroid'

Figure 2. GSTD manipulation of invalid instances

format ($o\_id$, $t$, $p_l$, $p_u$, $f$), where $t$ is the instance's timestamp, $p_l$ ($p_u$) is the lower-left (upper-right) corner of the instance's spacestamp (an MBR – assuming a 2D scenario), and $f$ is a flag denoting whether this instance is (spatially) valid or not.

In the GSTD algorithm, the three operations (*duration*, *shift* and *resizing*) correspond to three formulae that calculate the new timestamp and the new spacestamp's center and extent, as functions of current values and the respective parameters, namely `interval`, `Δc[]` and `Δext[]`. Note that `Δc[]`and `Δext[]` are arrays whose dimension depend on the dimensionality of the dataset being generated. Since chronological databases are supported, `interval` can range from 0 to +1. Also, `Δc[]` and `Δext[]` can range from –1 to +1, since the spatial workspace is assumed to be the unit (hyper-) cube. To do that, GSTD allows several user-defined parameters as input, namely:

– `N` and `D` that 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` that corresponds to the initial identification number of every object in the dataset;
– `numsnapshots` that corresponds to the time resolution of the workspace;
– `min_t` and `max_t` that correspond to the domain of the `interval` parameter;
– `min_c[]` and `max_c[]` that correspond to the domain of the `Δc[]` parameter;
– `min_ext[]` and `max_ext[]` that correspond to the domain of the `Δext[]` parameter;

and generates a series of tuples for each object, according to the following procedure: "*each object is initially active and, for each one, new instances are generated as long as its current instance is active and timestamp t < 1; when all objects become inactive, the algorithm ends*".

Initially, all objects are given starting locations, such that their center points are distributed in the workspace with respect to a chosen (`distr_init`) distribution, and their extensions are either set to zero (in case of point data) or calculated according to an `extent(N,D)` routine with respect to the input `N` and `D` parameters (in case of rectangular data). After the initialization phase, each new instance of an object is generated as a function of the current instance and the three parameters (`interval`, `Δc[]` and `Δext[]`). In order for a new instance to be generated, the values of the three parameters are calculated by calling a `RNG(distr,min,max)` routine, i.e., a random number generator for numbers between `min` and `max` that follow a given (`distr`) distribution. Currently, three popular distributions (namely, *uniform*, *gaussian* and *skewed*) are supported.

Obviously, it is possible that a coordinate may fall outside the workspace; GSTD manipulates *invalid* instances according to one among three alternative approaches:

– the *'radar'* approach, where coordinates remain unchanged, although falling beyond the workspace,
– the '*adjustment*' approach, where coordinates are adjusted (according to linear interpolation) to fit the workspace, and
– the '*toroid*' approach, where the workspace is assumed to be toroidal, as such once an object traverses one edge of the workspace, it enters back in the "opposite" edge.

In the first case, the output instance is appropriately flagged ($f = 0$ in the generated tuple) to denote its invalidity, although the next instance is still based on that one. In the other two cases, it is the modified instance that is stored in the resulting data file and used for the generation of the next one. Notice that in the '*radar*' approach, the number of objects present (i.e., valid) at each time snapshot may vary.

The three alternative approaches are illustrated in Figure 2. For sake of simplicity, it is only the centers of spacestamps that are illustrated; black (grey) locations represent valid (invalid) instances. In the example of Figure 2a, '*radar*' fails to detect $s_3$, hence $s_3$ is not stored, although the next location $s_4$ is based on that. Unlike '*radar*', the other two approaches always calculate a valid instance $s_3$' to be stored in the data file which, in turn, is used by GSTD for the

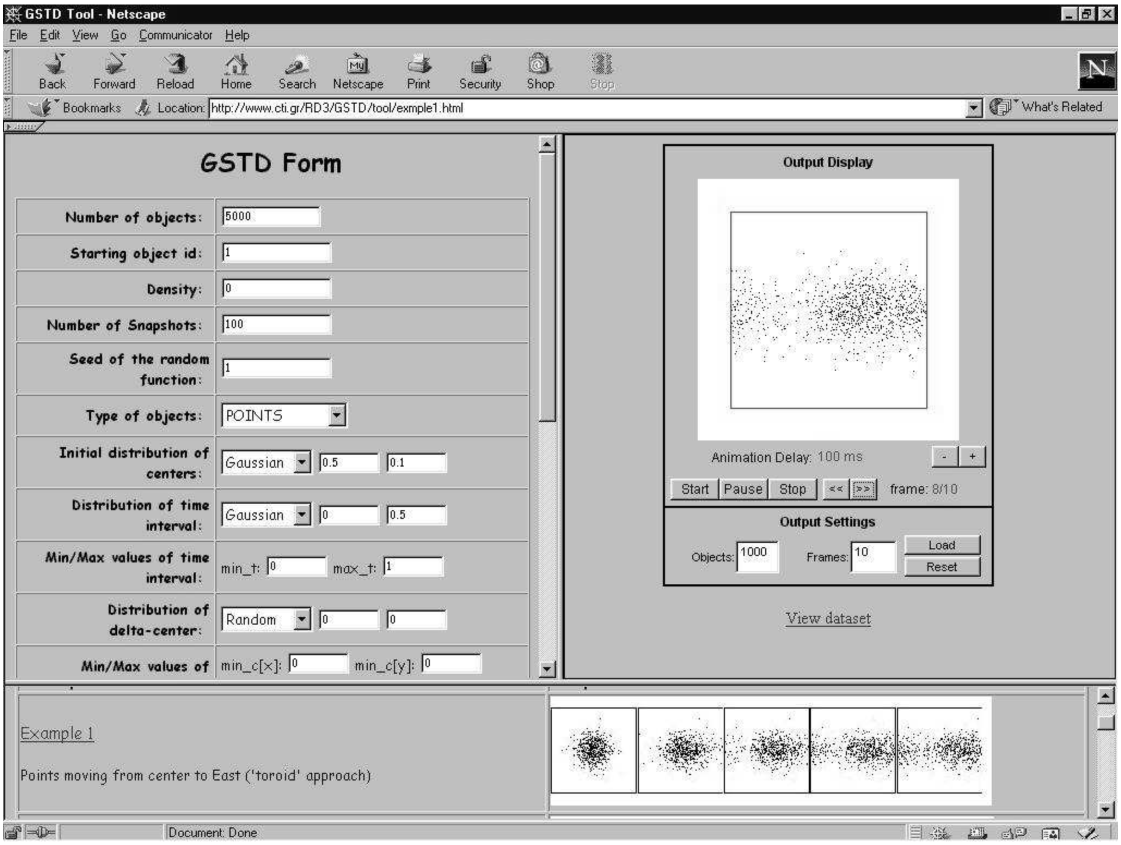



Figure 3. A screen dump of the web interface

generation of $s_4$. It is interesting to watch the behavior of $s_4$ in Figure 2c, where the calculated location finally stored ($s_4$') is actually identical to that in Figure 2a, as the effect of two consecutive calculations for $s_3$' and $s_4$'.

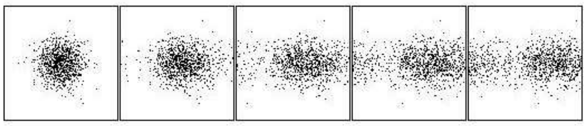
## 3. The GSTD Interface

As we mentioned earlier, real world examples of (point or region) spatiotemporal datasets abound. However, different motion scenarios correspond to different datasets, which a STAM should be evaluated on. For example, *random* versus *biased* heading, *fast* versus *slow* motion are some of the parameters that may lead to completely different applications and evaluation results.

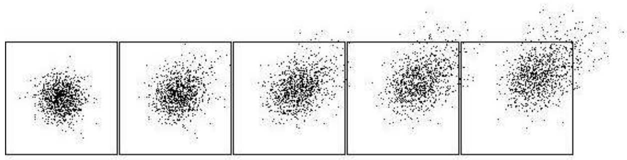
Through carefully using different distributions for the above parameters, the GSTD user can simulate several interesting scenarios; for instance, using a random distribution for Δc[i] and interval, all objects would move equally fast (or slow) and uniformly on the workspace; whereas using a skewed distribution for interval one would obtain a relatively large number of slow objects moving randomly; and so on. Also, by properly adjusting the distributions for each dimension of Δc[], one may control the heading of objects; for instance, by setting Δc[i]=Uniform(0,1) ∀i, one would obtain a scenario where the set of objects eventually converge to the upper-right corner of the unit workspace, irrespectively from the initial distributions; similarly, if one would prefer the objects to move towards some specific direction (e.g. East), he/she can simply adjust Δc by setting lower and upper bounds for each dimension, as discussed in detail below.

To facilitate use of the GSTD algorithm and provide a friendly tool to STAM designers and evaluators, we recently developed a WWW-based interface (Figure 3). Users simply set GSTD parameter values and choose the cardinality and resolution of the dataset to be generated (in the left-frame) and then the output is visualized on the screen (right-frame). Animation facilities, such as pause and frame by frame motion are also provided. Moreover, the generated tuples are downloadable for further use (e.g. for running experiments).

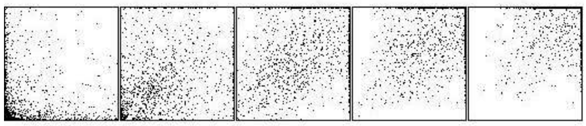
In order to provide some interesting scenarios, six example datasets, originally presented in [12], are ready for use (can be selected in the bottom-frame, where a series of snapshots are illustrated). Figure 4 presents the example datasets, where illustrated



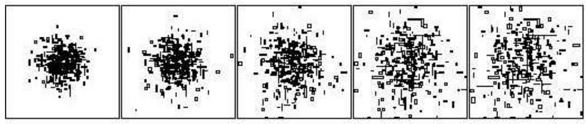
Scenario 1: points moving from center to East (‘*toroid*’)



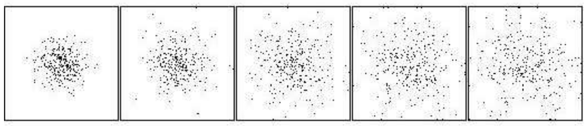
Scenario 2: points moving from center to NorthEast (‘*radar*’)



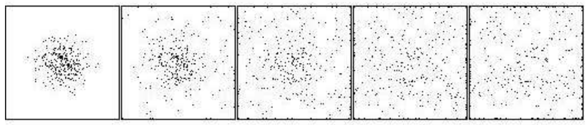
Scenario 3: points moving from SouthWest to NorthEast



Scenario 4: rectangles moving (and resizing) randomly



Scenario 5: points moving randomly (low speed)



Scenario 6: points moving randomly (high speed)

Figure 4. Example files generated by the GSTD algorithm

snapshots correspond to $t$ = 0, 0.25, 0.50, 0.75, and 1. Finally, the GSTD command line parameters appear, in case the user prefers to use the GSTD algorithm off-line (the source code is available via the GSTD home pages).

Scenarios 1 and 2 illustrate points with initial gaussian spatial distribution moving towards East and NorthEast, respectively. In the former case, where the ‘*toroid*’ approach was adopted, the points that traverse the right edge enter back in the left side of the workspace. In order to force the points to move strictly to the East we set Δ`c[y]` = 0 and Δ`c[x]` > 0. In the latter case, where the ‘*radar*’ approach is simulated, the points move towards NorthEast. As some of them fall beyond the upper-right corner (some quite early due to their high speed), they never reappear in the workspace since Δ`c[]` > 0. Scenario 3 illustrates an initially skewed distribution of points and their movement towards NorthEast. Since the ‘*adjustment*’ approach is the choice, the points concentrate around the upper-right corner. Scenario 4 includes rectangles initially located around the middle point of the workspace, which are moving and resizing randomly. The randomness of *shift* and *resizing* is guaranteed by a uniform distribution applied for Δ`c[]` and Δ`ext[]`. Finally, scenarios 5 and 6 exploit the *speed* of objects as a function of the GSTD input parameters. By increasing (in absolute values) the `min` and `max` values of Δ`c[]`, users can generate ‘faster’ objects while the same behavior could be achieved by decreasing the `max_t` input that affects `interval`. Similarly, the *heading* of objects can be controlled, as presented in scenarios 1 through 3.

It is worthwhile to stress that the above scenarios and respective parameters can be visualized (thus working as a tutorial) using the example datasets in the bottom-frame of the web interface (Figure 3).

## 4. Conclusion

In this article, we discussed the GSTD algorithm that generates sets of moving points or rectangles according to user requirements, thus providing a tool that could simulate a variety of possible scenarios. Moreover, we presented the WWW environment that we developed to facilitate the use of the GSTD algorithm.

Although extended related work is found in traditional database benchmarks and data generators (e.g. [1, 2]), in the field of spatial databases it is very limited [10, 3]. In particular, when motion is introduced to support spatiotemporal databases, to our knowledge the single related work is the Oporto generator [8] but for specific purposes (it actually generates scenarios with harbors, fishing ships, spots and shoals of fish).

An interesting issue that arises and deserves more attention is about the location of an object at time $t_j$, such that $t_i < t_j < t_{i+1}$, based on the knowledge of two instances that correspond to consecutive timestamps, $t_i$ and $t_{i+1}$. If *linear interpolation* is followed, the spacestamp is considered to be moving with respect to a start- (at time $t_i$) and an end- (at time $t_{i+1}$) location. Alternatively, *projection* means that the spacestamp is considered to be static and equal to the one at time

$t_i$. Both alternatives find applications in real world; navigational and cadastral systems, respectively, are good examples. In any case, detecting the status of object *o* at a time instance during ($t_1$, $t_2$) is an open issue (e.g. uncertainty may need to be captured [7]). As we argued in [12], the GSTD algorithm is independent of that issue; actually, it generates a series of instances regardless of that. Unlike the underlying data generator, it is a STAM construction algorithm or a visualization tool that needs to decide on the one or the other scenario to be supported.

## Acknowledgements

Yannis Theodoridis was partially supported by the EC funded TMR project "CHOROCHRONOS: A Research Network for Spatiotemporal Database Systems". Mario A. Nascimento was initially partially supported by CNPq and by the Pronex/FINEP SAI project, and is currently supported by a Startup Research Grant from the Univ. of Alberta. Many thanks to Jefferson R.O. Silva for implementing the GSTD algorithm and running a large set of series of experiments, and to Aggelos Kokorogiannis and Ioannis Poulakis for building the WWW-based interface.

## References

1. D. Bitton, D. J. DeWitt, C. Turbyfill, "Benchmarking Database Systems: A Systematic Approach", *Proceedings of the 9th Int'l Conference on Very Large Data Bases* (VLDB), 1983.
2. J. Gray, P. Sundaresan, S. Englert, K. Backlawski, P. J. Weinberger, "Quickly Generating Billion-Record Synthetic Databases", *Proceedings of ACM SIGMOD Conference on Management of Data*, 1994.
3. O. Günther, V. Oria, P. Picouet, J.-M. Saglio, M. Scholl, "Benchmarking Spatial Joins A La Carte", *Proceedings of the 10th Int'l Conference on Scientific and Statistical Database Management* (SSDBM), 1998.
4. G. Kollios, D. Gunopulos, V.J. Tsotras, "On Indexing Mobile Objects", *Proceedings of the 18th ACM Symposium on Principles of Database Systems* (PODS), 1999.
5. M. A. Nascimento, J. R. O. Silva, "Towards Historical R-trees", *Proceedings of ACM Symposium on Applied Computing* (ACM-SAC), 1998.
6. M. A. Nascimento, J. R. O. Silva, Y. Theodoridis, "Evaluation of Access Structures for Diecretely Moving Points", *Proceedings of Int'l Workshop on Spatio-Temporal Data Management* (STDBM), 1999.
7. D. Pfoser, C. S. Jensen, "Capturing the Uncertainty of Moving-Object Representations", *Proceedings of the 6th Int'l Symposium on Spatial Databases* (SSD), 1999.
8. J.-M. Saglio, J. Moreira, "Oporto: A Realistic Scenario Generator for Moving Objects", *Proceedings of the 10th Int'l Workshop on Database and Expert Systems Applications* (DEXA), 1999.
9. S. Saltenis, C.S. Jensen, "R-tree based Indexing of Spatio-Temporal data", TimeCenter Technical Report, TR-45, 1999.
10. M. Stonebraker, J. Frew, J. Dozier, "The SEQUOIA 2000 Project", *Proceedings of the 3rd Int'l Symposium on Spatial Databases* (SSD), 1993.
11. Y. Theodoridis, T. Sellis, A. Papadopoulos, Y. Manolopoulos, "Specifications for Efficient Indexing in Spatiotemporal Databases", *Proceedings of the 10th Int'l Conference on Scientific and Statistical Database Management* (SSDBM), 1998.
12. Y. Theodoridis, J. R. O. Silva, M. A. Nascimento, "On the Generation of Spatiotemporal Datasets", *Proceedings of the 6th Int'l Symposium on Spatial Databases* (SSD), 1999.
13. Y. Theodoridis, M. Vazirgiannis, T. Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications" *Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems* (ICMCS), 1996.
14. J. Zobel, A. Moffat, K. Ramamohanarao, "Guidelines for Presentation and Comparison of Indexing Techniques", *ACM SIGMOD Record*, 25(3): 10-15, 1996