GeoInformatica 6:3, 207–231, 2002

On the Generation of Time-Evolving Regional Data*

THEODOROS TZOURAMANIS
Laboratory of Data Engineering, Department of Informatics, Aristotle University, Thessaloniki 54006, Greece
E-mail: theo@delab.csd.auth.gr

MICHAEL VASSILAKOPOULOS
Laboratory of Data Engineering, Department of Informatics, Aristotle University, Thessaloniki 54006, Greece
E-mail: mvass@computer.org

YANNIS MANOLOPOULOS†
Department of Computer Science, University of Cyprus, Nicosia 1678, Cyprus
E-mail: manolopo@ucy.ac.cy

Received June 26, 2001; Accepted May 3, 2002

Abstract

Benchmarking of spatio-temporal databases is an issue of growing importance. In case large real data sets are not available, benchmarking requires the generation of artificial data sets following the real-world behavior of spatial objects that change their locations, shapes and sizes over time. Only a few innovative papers have recently addressed the topic of spatio-temporal data generators. However, all existing approaches do not consider several important aspects of continuously changing regional data. In this report, a new generator, called generator of time-evolving regional data (G-TERD), for this class of data is presented. The basic concepts that determine the function of G-TERD are the structure of complex 2-D regional objects, their color, maximum speed, zoom and rotation-angle per time slot, the influence of other moving or static objects on the speed and on the moving direction of an object, the position and movement of the scene-observer, the statistical distribution of each changing factor and finally, time. Apart from these concepts, the operation and basic algorithmic issues of G-TERD are presented. In the framework developed, the user can control the generator response by setting several parameters values. To demonstrate the use of G-TERD, the generation of a number of sample data sets is presented and commented. The source code and a visualization tool for using and testing the new generator are available on the Web.[1] Thus, it is easy for the user to manipulate the generator according to specific application requirements and at the same time to examine the reliability of the underlying generalized data model.

Keywords: spatio-temporal databases, benchmarking, synthetic data generators, regional data, access methods

1. Introduction

Spatio-temporal databases (STDBs) provide a framework for the efficient storage and retrieval of all states of a spatial database over time. This includes the current and past states and the support of spatial queries that refer to present and past time points, as well. Nowadays, new spatio-temporal applications that require spatio-temporal data to be stored

* Research performed under the European Union's TMR Chorochronos project, contract number ERBFMRX-CT96-0056 (DG12-BDCN)
† On sabbatical leave from the Department of Informatics, Aristotle University, Thessaloniki, 54006 Greece.

on-line are emerging. Image and multimedia databases, mobile communications, geographical information systems (GIS), urban planning, computer-aided design (CAD), medical databases are among these applications. Commercial database management systems (DBMSs) do not support the special characteristics of data objects that change their spatial locations, directions, shapes and sizes over time, because of their inability to implement a total ordering of objects in space and time domains and preserve proximity, at the same time. In many cases, the design and implementation of a STDB has been left as an extension of an established commercial DBMS. The long-term goal is to provide concepts and techniques that support STDB applications. Towards this goal, during the last years many efforts have focused on spatio-temporal formalism, data models, query languages, visualization and access methods [1], [2], [11], [17]. However, little work has appeared on benchmarks for STDBs. In this paper we will focus on the problem of generating benchmark data for STDBs.

The goal of benchmarks in STDBs is to compare the performance of different implementation alternatives. An example of a benchmark is the comparison of space requirements and query execution time of spatio-temporal access methods (STAMs). In order to evaluate such STAMs, extensive experimentation using real and synthetic data is required. A good benchmark must correspond to a recognizable, comprehensible real-life situation. It is important that the results hold not only for a specific environment but in more general settings, as well. Thus, the user is able to repeat the experiments and come to similar conclusions [26]. Along these guidelines, the present research aims at the creation of realistic benchmark data that enable the comparison of different access methods and algorithms for STDB applications.

For the comparison of the performance behavior of different access methods, the selection of data is of high importance as large spatio-temporal data sets are needed. We can distinguish between real data, which come from real-world applications and synthetic data generated according to statistical distributions. Very often either real data sets are not available or they cannot be useful for testing extreme conditions. In both cases synthetic data sets can be generated by some artificial specifications rather than by obeying a real-world behavior.

The use of synthetic data allows testing the behavior of access methods in exactly specified or extreme situations. Furthermore, different statistical distributions and parameter settings correspond to different scenarios. They also correspond to completely different synthetic data sets and applications on which an efficient access method should be evaluated too. Although, it is difficult to derive conclusions with respect to the performance in realistic situations by using synthetic data, in this paper it is shown that, at least in some cases, one can generate spatio-temporal data sets that simulate real-world behavior.

Much work has been done towards data generation for benchmarks in prototype and commercial non-spatio-temporal databases [3], [6], [7], [9], [10], [14], [16], [19]. These generated data can be used to test different implementations under various operating conditions. In STDBs, the work on the generation of test data is limited and only a few innovative papers have appeared in the literature. In Theodoridis et al. [21] the authors present a general approach for the generation of synthetic scenes of moving points, or

rectangular objects. This approach is parameter-driven and does not support the interaction between objects. In Pfoser and Theodoridis [15], the previous work is extended by introducing new features to the generation process. In Saglio and Moreira [18] a specialized spatio-temporal data generator, motivated by an application modeling fishing boats, is proposed, where there exist no, or very few restrictions for the motion: e.g., objects of one type may be attracted, or repulsed by objects of other classes. Recently also, another approach for the generation of test data, which is motivated from applications in the context of traffic telematics, has been presented in Brinkhoff [4], [5].

None of these approaches which are more thoroughly presented in Section 7 are suitable for benchmarking STAMs for regional data (especially quadtree-based STAMs). In this paper, we introduce a novel approach for data generation, which is specifically designed for applications stemming from the field of time-evolving regional data. The new generator, therefore, is called generator of time-evolving regional data (G-TERD). However, it is not the paper's goal to demonstrate (e.g., as in a case study) that the data computed by the presented generator fulfill only the requirements of a specific application from this field. Instead, our software tool is highly parameterized so that different parameter values may produce spatio-temporal data set distributions with different characteristics. In the presented framework, the user can control the behavior of the generator by defining parameters and statistical models. Such an approach considers the characteristics of a wide range of applications. The source code, also, is available on the Web, making it easy for the user to adjust the generator according to different classes of STDB application requirements.

The remaining part of the paper is organized as follows. Section 2 discusses the spatio-temporal benchmarking environment. Section 3 describes the basic concepts which were considered during the generator design for continuously changing synthetic regional data. Section 4 describes the operation of the new generator and some basic programming issues. Section 5 provides a description of the visualization demo, whereas Section 6 presents example setups and the sequences of scenes generated. Section 7 summarizes previous proposals for spatio-temporal data generation and finally, the last section concludes the paper and discusses future work issues.

2. Benchmarking and data generation in STDBs

The major effort in STDBs focuses on the design and implementation of efficient indexing techniques aiming at reduced query execution time. The proposed indexing schemes behave differently according to certain settings, like the spatial objects format (i.e., raster vs. vector), the spatial objects nature (e.g., points, lines, areas, volumes etc.), the data set distribution, the buffering strategies, the system caches, the disk page size and the query types (e.g., contain, overlap, nearest neighbor etc.). Moreover, in most cases, each indexing scheme has been evaluated separately. It is therefore difficult to compare several indexing schemes under a common framework. To the authors' knowledge the only extensive, though not exhaustive, experimental comparison of STAMs has been done in [12].

In order to compare the behavior of different indexing schemes under the same settings,

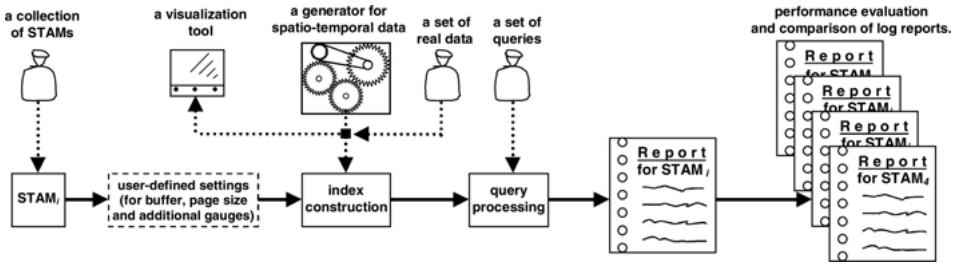


Figure 1. A simplified benchmarking system environment for STDBs.

there must be a flexible performance benchmarking environment. The general architecture of a benchmarking environment for STAMs is presented in figure 1 and according to [21] it consists of the following components:

- a collection of access methods,
- a module that generates synthetic data sets which cover a variety of real life applications,
- a set of real data that also represents various real life examples,
- a query processor capable to handle a large set of queries for extensive experimentation purposes,
- a reporter to collect all relevant output report logs, and
- a visualization tool that could be able to visualize data sets for illustrative purposes.

In this paper we concentrate on the development of a novel synthetic raster data generator for STDBs and a visualization tool to facilitate its use. The new generator is called generator of time-evolving regional data (G-TERD). A framework is also proposed for defining parameters and statistical models of G-TERD by the users. Test runs have shown that the new generator is possible to produce data sets with characteristics for a wide range of STDB applications. The Web site of G-TERD, http://delab.csd.auth.gr/stdbs/g-terd.html, allows downloading the generator, its documentation and source code, for experimentation purposes. The visualization tool of G-TERD runs under MS-Windows and it is also provided by the Web site so that in connection the user can visualize the computed time-evolving images.

3. Fundamental concepts

We assume a 2-D workspace, where real coordinates are used (float numbers in C) and that the space extent on the x- and the y-axis is set by the user. In the sequel, in order to simplify presentation, the values of coordinates are counted in ''units''. With respect to the time domain, we assume that the changing scene lasts for a period of time $T = [0, t_{\max})$. This period is divided in *num_timeslots* time slots. We further assume that the scene remains unchanged during a time slot. In other words, time is digitized.

3.1. Objects and sub-objects

The basic data structure of G-TERD is the time-evolving 2-D regional object. Each object consists of a group of sub-objects which are quadrangles of the same, or different colors. This group of sub-objects, in general, changes at each time slot. An object, or sub-object may have a static shape, or it can change its spatial extension with time. The spatial extension of an object is determined by the smallest possible 2-D rectangle, called minimum bounding rectangle (MBR), that encloses all its sub-objects. The user sets the maximum size of the object MBR and the maximum size of the sub-object quadrangle.

In order to simulate real-world complex non-rectangular objects (e.g., cars, animals, airplanes, clouds, islands) by an object, some, or all of its sub-objects may be connected and remain connected, for the whole lifetime of the object.

The number of objects appearing in the scene and the number of sub-objects in an object are dynamic. An object, or a sub-object may exist over the whole scene lifetime T, or may be created at one time slot and disappear at a later time slot. The objects may be static, or moving towards any spatial direction. The square sub-objects may, also, expand, or shrink and rotate around their center.

For the sake of simplicity, both elastic shocks and plastic crushes between objects, are not allowed. Instead, in case that two objects come towards each other, there is a possibility that the one object will pass over the other, as if they were moving in different heights, or there is a possibility that both objects will change moving directions in order to avoid the crush. More details on the influence of a moving, or a static object on the speed and moving direction of another object, are given in Subsection 3.4.

3.2. The scene-observer

The scene-observer plays a central role in a time-evolving scene. In real-life, the observer may be a satellite monitoring the earth surface, a telescope watching the space universe, the field of vision of a video or a photo camera, the eyes of an onlooker, etc. The scene-observer in G-TERD is a virtual 2-D rectangular window that shifts, zooms and rotates over the scene and films it, by printing one snapshot image per time slot, for the whole lifetime of the evolution.

The observer's window side length is user-controlled and it should not be wider than the side length of the space universe, in the x- and y-dimensions. The observer's window digitizes a workspace portion. The dimensions of a quadrangle of the workspace that is represented by a pixel of the observer's window, or in other words, the size of such a pixel varies according to the zoom-in, or zoom-out factor. At the starting phase of G-TERD, every 1×1 square units region of the workspace corresponds to one pixel of the observer's window. The scene-observer may move towards any spatial direction, or he/she may be static for some periods of time or for the whole scene lifetime. The observer may shift (''travel'') in a random way over the surface of the predefined workspace, or may follow the movement of a specific ''live'' object. Other functions supported are the zoom in-out and the rotation around the center point of the observer's window. Using the concept of the

scene-observer's window, a sequence of multicolored images that can be saved in the disk are produced.

3.3. Change of spatial locations, size, shape and color of the objects

Data objects may change their spatial location, size and/or their shape at different time intervals, according to the values of their speed, zoom and rotation-angle fields. These fields are measured in units of the workspace coordinate system per time slot and their domains are bounded by minimum and maximum values set by the user. The distributions of the speed, zoom and rotation-angle values are described in Subsection 4.2. A positive speed value for the x- (y-) axis denotes that the direction of the specific object/sub-object on this axis is to the East (North). A positive zoom value denotes that the specific object/sub-object expands (zooms-in), and finally, a positive rotation-angle value means that the specific object/sub-object rotates clockwise.

There are static, slow and fast moving objects, whereas a special flag is used to distinguish between them. When an object is static (in slow movement) the speed, zoom and rotation of all its sub-objects is set at zero (at half the value of the speed, zoom and rotation of the object).

In real-life examples, objects appear gradually in our field of vision. Thus, when a new object is created, initially all its square sub-objects have a zero side length and a ''$+1$'' zoom value. This zoom value guides its expansion. The opposite holds at deletion time, when the object finally disappears because the surface of every sub-object becomes less than 1 square unit. A sub-object may, also, disappear if in the course of its lifetime, its surface is calculated below 1 square unit. During that period, the sub-object remains ''live'', but it cannot appear in the resulting data set. This may happen to an object, also, if its ''live'' sub-objects disappear all together for a while, during the same period of time.

In general, each sub-object has its own color. The number of colors in the color palette is a user-defined parameter. When a special flag of an object is true, then all its sub-objects have the same color. However, in real-life applications, the moving objects may have slightly different colors according to the intensity of light. In order to simulate this behavior, each object/sub-object may change its color after a period of time (determined by a specific field of its structure) to an adjacent color, assuming that the color palette of the changing scene follows the model of the rainbow colors.

3.4. Interaction movement

Real world objects do not change their spatial locations, or their shape in a completely random fashion. Instead, the surrounding environment affects them, and their behaviors are affected by some constraints. Therefore, a scene generation producing completely random data will not be a representative simulation of numerous classes of real world application examples.

For instance, a flying observer may watch that a ship cannot navigate over an island

(static object), or over a second ship, or over a cloud (moving objects). On the contrary, it is probable that a cloud may pass over the ship and over the island. In order to simulate such instances, we assume that each object has an attribute array of *true* or *false* flags (*obj.CanPassOver*[]), where each flag corresponds to another ''live'' object. If for an object obj_1, it holds that $obj_1.CanPassOver[k] = false$, where k is the identifier of another object obj_2 ($k = obj_2.id$), then the object obj_1 cannot ignore the existence of object obj_2 in the future and it cannot pass over the object obj_2, if these two objects come towards each other. On the one hand, this resembles the case of a ship and an island, whereas on the other hand, it resembles the case of a ship and a cloud. Additionally, if $obj_2.CanPassOver[obj_1.id] = false$ (or $= true$), then the obj_2 cannot ignore (can ignore) the existence of object obj_1 in the future and it cannot pass (can pass) over object obj_1, if these two objects come towards each other. This resembles the case of the second ship (or the cloud, respectively) and the first ship, of the previous example.

As mentioned earlier, both elastic shocks and plastic crushes and all their additional properties that influence the movement and the appearance of the objects, are not supported. In case of two objects that cannot pass over each other but are coming towards each other, or in case of an object coming towards a border of the predefined workspace, an alarm warns the object(s) about the immediate danger of a crush. Then, the sub-objects that are closer to the danger change their moving directions, or if this is not possible, they stop moving, at least for a while.

4. The operation of G-TERD

In this section, a number of topics related to the initialization, parameterization, key sub-functions and the main function of G-TERD are presented.

4.1. User-defined parameters

In G-TERD several parameters may be user-defined in order to let the user control the behavior of the generator. Table 1 specifies all the user-defined parameters of G-TERD that can be set according to different classes of STDB application requirements.

The appropriate definition of these parameters is the simplest technique to control the properties of the resulting data sets. For instance, by setting *speed_min* = *speed_max* for both the x- and y-axes, then all the objects are forced to move in a parallel fashion at the same speed as if they were one object. This is similar to the movement of birds or military aircrafts flying together, or to the movement of a group of soldiers. Further, if *speed_min* and *speed_max* are set at zero, the position of the center point of all the objects and all their sub-objects remains constant, and the sub-objects may only zoom in-out and rotate. If the user sets *rotation_min* = *rotation_max*, then all the objects are forced to rotate together following the same direction and the same rotation-angle per time slot. This is similar to people engaged in synchronized movement.

Table 1. The user-defined parameters of G-TERD.

Parameter	*Explanation*
X_{max}	Workspace length on the x-axis
Y_{max}	Workspace length on the y-axis
num_timeslots	Time duration of the changing scene ($T = [0, num\_timeslots)$)
max_num_objs	Maximum number of ''live'' objects per time slot
max_num_subobjs	Maximum number of ''live'' sub-objects in an object per time slot
num_colors	Number of colors of the color palette
observer_side	Length of the observer's window side
obj_side_max	Maximum side length of object MBR
subobj_side_max	Maximum side length of the rectangular sub-object
percent_objs_static	Percentage of static objects per time slot
percent_objs_slow	Percentage of objects in slow movement, zoom and rotation per time slot
percent_objs_fast	Percentage of objects in fast movement, zoom and rotation per time slot ($percent\_objs\_fast = 100 - percent\_objs\_static - percent\_objs\_slow$)
speed_min[]	Minimum object speed per time slot positive value on x- (y-) axis: movement to the East (North)
speed_max[]	Maximum object speed per time slot positive value on x- (y-) axis: movement to the East (North)
zoom_min	Minimum object in-out zoom per time slot ($-1 \leq zoom\_min \leq 1$)
zoom_max	Maximum object in-out zoom per time slot ($-1 \leq zoom\_max \leq 1$)
rotation_min	Minimum object rotation-angle per time slot
rotation_max	Maximum object rotation-angle per time slot
$t_{max\_duration}$	Maximum number of time slots that have to elapse before the next computation of a field value of an object, a sub-object or of scene-observer
live_objects at $t = 0$	Number of ''live'' objects at time slot $t = 0$

4.2. Data distributions

A benchmarking environment should support data obeying a variety of widely-used continuous and discrete data distributions. Some popular well-established statistical data distributions are the Uniform, Triangular, Normal, Exponential, Zipf, Poisson and Gamma distributions. In order to have a generalized tool that simulates classes of real-life applications, several quantities that determine the operation of this tool must be random variables that obey a specified distribution among the ones mentioned above. For G-TERD, these random variables are presented in table 2.

Through careful specification of different distributions for the variables of table 2, the user can simulate several interesting scenarios. For instance, by using the exponential distribution with small mean for the speed and the period of time before the re-computation of the object speed, most of the objects would move slowly and ''nervously'' on the workspace, since their speed would change direction very frequently. On the other hand, it is possible to use large $t_{max\_duration}$ and exponential distribution with large mean for the number of time slots that must elapse before the re-computation of the object speed. In this case large periods will be required before the subsequent computations of the speed and many objects will move only in one direction during their whole lifetime. To find an

Table 2. Variables that are controlled by statistical distributions.

Variable	*Distribution*	*Domain*
Number of ''live'' sub-objects *obj.live_subobjs* in an object *obj* at the time slot of its creation	User-defined	$[1, max\_num\_subobjs]$
Initial distribution of the center point of a new object	User-defined	The workspace
Number of new objects per time slot	User-defined	$[0, max\_num\_objs - live\_objects]$
Number of new sub-objects in an object per time slot	User-defined	$[0, max\_num\_subobjs - obj.live\_subobjs]$
Deletion time slot *obj.endtime* of an object	User-defined	[birth time of the object $+1, num\_timeslots$]
Deletion time slot *subobj.endtime* of a sub-object	User-defined	[birth time of the object in which it belongs $+1, obj.endtime$]
Color *.color* of an object or a sub-object	User-defined	$[1, num\_colors]$
Speed *subobj.speed* of a moving sub-object per time slot and spatial axis	User-defined	$[speed\_min, speed\_max]$
Zoom *subobj.zoom* of a moving sub-object per time slot	User-defined	$[zoom\_min, zoom\_max]$ with $-1 \leq zoom\_min \leq zoom\_max \leq 1$
Rotation-angle *subobj.rotation* of a moving sub-object per time slot	User-defined	$[rotation\_min, rotation\_max]$
Number of time slots that must elapse before the next computation of an attribute such as the speed, zoom, color, etc.	User-defined	$[1, t_{max\_duration}]$

intermediate state, i.e., objects having directed movements for longer periods of time, we have to use intermediate values for the periods of time during which the speed is kept constant.

Evidently, by properly adjusting the domain value of each variable of table 2, the user may limit the data generated from the chosen distribution. For instance, we can consider setting the domain of the zoom value equal to $[-1, 0]$. This will lead to a scenario where every created object would expand for a while, during the initialization phase of its creation, and afterwards it will be deleted (''die'') in a very short time. In the same way, the experimenter may prefer the objects to move towards some specific orientation but each at a different speed. To achieve this, the experimenter can give non-positive, or non-negative values different from each other to both the lower and upper bounds of the sub-objects speed.

Among all the supported distributions, the Uniform $U(min, max)$ and the Triangular $Tr(min, max)$ distributions require setting only the minimum *min* and maximum *max* values of the domain value, whereas the rest require extra user-adjusted parameters. For instance, the Exponential $E(\mu, min, max)$ distribution also needs the mean μ parameter as input, the Normal $N(\mu, \sigma, min, max)$ also needs the mean μ and the mean square deviation σ, whereas the Zipf distribution needs setting of a parameter that controls the distribution skewedness.

G-TERD can be utilized to generate benchmark data for many application domains, given that their data distributions are known. The generator currently supports the Uniform, Triangular, Normal, Exponential, Zipf and Poisson distributions.

4.3. Creation of a new object

Initially, the new object location is selected so that its center point is randomly placed in the workspace, according to a predefined statistical distribution. The acceptable placement of an object obj_1 is controlled by a function, which checks if the selected workspace area is occupied by another object obj_2. If this happens and $obj_1.CanPassOver[obj_2.id] = obj_2.CanPassOver[obj_1.id] = false$, then no object can be positioned over the other. Therefore, another position for the object obj_1 must be found.

Afterwards, a decision is made about the number of sub-objects that each object will initially have, about their color (the same for all sub-objects, or not) and about their speed and rotation-angle per time slot. The surface of each rectangular sub-object is set at zero and a ''$+1$'' zoom value undertakes its expansion. The speed and rotation-angle of each sub-object are set randomly, following their domain value and the properties of the related user-defined distributions. The instance of a sub-object for the next time slot is calculated, for each speed, zoom and rotation-angle candidate value. If the sub-object comes towards a crush with a sub-object of another object, then the algorithm checks the *.CanPassOver*[] field of both objects that own these sub-objects. Based on the result of this check, the algorithm will determine if the sub-object being processed has to change its speed, zoom, or rotation-angle. The same will happen, if the sub-object being processed is expected to have invalid coordinates (coordinates located outside the workspace) at the next time slot.

The scenario is not very realistic at the beginning of the data generation. All the sub-objects of the newly created ''live'' objects cover a surface of zero size and expand by 1 square unit per time slot. Therefore, to obtain satisfactory results, the generated data should not be used during a warm-up phase [18]. In practice, the warm-up phase may hold until at least the 1 of the 1 square units region of the workspace becomes colored. If the output of G-TERD is expected to be a sequence of *num_timeslots* multicolored images, then in the course of the data generation process, G-TERD computes and saves in the disk $k + num\_timeslots$ images, where k is the sequence of images belonging to the warm-up phase. G-TERD names the output images as $-k, -k+1, \ldots, -1, 0, 1, \ldots, num\_timeslots - 1$ and the experimenter may ignore the k images corresponding to scenes produced during the warm-up phase of the evolution.

4.4. Update the instance of an object

The procedure starts with the calculation of the new object location and MBR. If all the sub-objects of the object have the same color, and the period of time that this color remains unchanged has expired, then a new color for all its sub-objects is selected. Otherwise, if its sub-objects do not have all the same color, each of them may change its color,

independently. If the period of time during which the speed, zoom and/or rotation-angle of each sub-object has expired, then a new value is set to the corresponding field. The instance of the sub-object for the next time slot is calculated, for each speed, zoom and rotation-angle value, and a procedure similar to the one that was described at the end of the previous paragraph, is followed, to avoid an undesirable crush.

4.5. Positioning of the scene observer

The shift, zoom and rotation functions of the scene-observer are similar to the corresponding functions of an object. If the scene-observer follows the evolution of a specific object, then the observer's speed is proportional to the distance of the center point of its window from the center point of the object followed. In this case, the zoom in-out function of the observer takes care of keeping the whole object inside the observer's window.

4.6. The output

The output of the scene-observer is a sequence of *num_timeslots* multicolored images that can be saved in the disk. For each pixel in the observer's window, the algorithm checks if there is any sub-object covering the pixel. In this case, the sub-object color is recorded in the output. If sub-objects of different objects cover the same pixel of the observer's window, then the color of the object, which can pass over any other of the involved objects, appears in the output image.

Since the number of colors, *num_colors*, supported by the time-evolving scene is defined by the user, each pixel of the observer's window needs $\log_2(num\_colors)$ bits in the resulting file. Thus, if necessary, the user may subsequently transform each image to grayscale or black and white.

4.7. The main routine

The main part of the algorithm is illustrated in figure 2. The input of the algorithm (lines 1–2) consists of the values and statistical distributions of all the parameters and variables that appear in table 1 and table 2, respectively. During the scene initialization phase (lines 3–5) at time slot $t = 0$, a user-defined number of *live_objects* ''live'' objects is created and located in the workspace. The initialization of the observer's window and the output of the first snapshot image of the time-evolving scene follow.

During the main loop phase (lines 8–16), new object instances are generated. If the deletion time slot of an object has already been reached and the surface of each of its sub-objects is less than 1 square unit, the object is deleted from the scene. A random number of new objects at each time slot is also created and placed in the workspace (line 14). The random number of new objects follows a predefined statistical distribution, such as the ones discussed in Subsection 4.2. Finally, the observer's window is located at another

```
line   ROUTINE main( )

       BEGIN
 1     The user must define all the parameters of Table 1;
 2     The user must select the statistical distributions of all variables of Table 2;
 3     The time slot being processed is t = 0;
 4     FOR each one of the live_objects "live" objects at time slot t DO
 5           Initialize the new object and place it in the workspace;
 6     Initialize the scene-observer and place it over the workspace;
 7     Output the first snapshot image (time slot t) of the time-evolving scene;

 8     WHILE t < num_timeslots DO
           BEGIN
 9         The time slot being processed is t = t + 1;
10         FOR each one of the live_objects "live" objects at time slot t DO
               BEGIN
11             Update the instance of the time-evolving object in the workspace;
12             IF the deletion time slot of the object has already occurred and
                   the surface of each sub-object has became less than 1 square unit THEN
13                 Delete this object and update the number of live_objects "live" objects
                       in the workspace, for the time slot t;
               END
14         Create a random number of new objects from the domain value [0, max_num_objs – live_objects],
               initialize and place them in the workspace;
15         Update the position of the scene-observer;
16         Output the snapshot image of the time-evolving scene for the time slot t;
           End

       END OF ROUTINE main( )
```

Figure 2. The main routine of G-TERD.

position over the workspace and the output function prints the snapshot image of the current state of the continuously changing scene.

5. A visualization tool

In order to make G-TERD available to the user for experimentation purposes, a Web site has been created, which allows downloading the MS-DOS executable file of the generator, the user documentation and the source code in C language. The Web site is at http://delab.csd.au,th.gr/stdbs/g-terd.html and provides, also, a visualization tool that runs under MS-Windows.

The executable file of G-TERD allows creating time-evolving data sets based on both the user-defined parameters of table 1 and the statistical distributions described in Subsection 4.2. The user may use these data sets as input to the visualization tool. Different parameter values and statistical models may produce time-evolving regional data set distributions with different characteristics, corresponding thus to different classes of STDB applications. The visualization tool permits to both visualize the synthetic data sets and simplify the definition of the user-defined parameters.

The visualization tool is based on Gnuplot 3.7 and it is easily installed. It is offered with sample data sets from different settings of the user-defined parameters and the random

variables obeying specific distributions. Thus, by using the sample data sets as tutorial, the user can immediately get an impression whether G-TERD may be suitable for his/her applications. The reason for adopting Gnuplot as a basis is that it supports all the different MS-Windows platforms and it does not require on-line Internet connection to be executed as happens to other similar tools [8], [19]. Besides, through Gnuplot the experimenters can easily visualize the generated data with no need for specific Web browsers and support for specific Java versions, as also happens with other tools [4], [8], [18], [20]. However, the development of an interactive Web-based environment, to both generate and visualize time-evolving synthetic regional data, is an activity in progress.

6. Test runs

In the following, we give an example of the use of the new generator for time-evolving regional data sets, by simulating several realistic scenarios. The scenarios correspond to different definitions of the parameters of table 1 and the statistical distributions that appear in table 2. The percentage of static and moving objects, their moving direction, the statistical model that should be followed, the number of new objects per time slot and the movements of the scene-observer, are only some of the parameters that lead to in completely different application examples.

For all scenarios, the side length of the scene-observer's window is set at 1024 units, the maximum number of live objects per time slot is 70 and the lifetime of the evolution is 101 time slots ($T = [0, 101]$). The snapshots illustrated in figures 3–7[2] correspond to time slots $t = 10$, 30, 50, 70, 90 and 100 for each scenario. The random number of new sub-objects per time slot and their deletion time slot are Uniform within their domain value. The same holds for the speed, the color and the period of time that must elapse before the next computation of an attribute such as the speed, zoom, etc. Finally, the change of the size and shape of the objects/sub-objects are controlled by the zoom and rotation-angle values per time slot, which are generated by the Triangular distribution. Table 3 depicts all the non-fixed user-defined parameters of the presented scenarios in this section. Note that, in order to make the snapshot more appropriate for black and white printing devices, only six dark colors have been allowed. There is no such color restriction in the Web version of G-TERD.

Note, also that in our experimentation, G-TERD required in average some seconds for the generation of a $1{,}024 \times 1{,}024$ image snapshot of each synthetic data set. Generally, the running-time of the generator is affected by the parameter settings of the user. The parameters that act decisively on the running-time are the workspace size, the time domain, the average number of new objects and sub-objects per time slot, and their average deletion time slot.

The first scenario in figure 3 illustrates static objects that are uniformly distributed in a workspace of $2{,}000 \times 2{,}000$ units. The speed domain value is $[10, 10]$ for each axis. Therefore, the scene-observer shifts over the workspace surface at a constant speed and diagonal orientation, from south-west to north-east. The objects are all created at the initial time slot, whereas in the sequel the creation of a new object is not allowed, since the

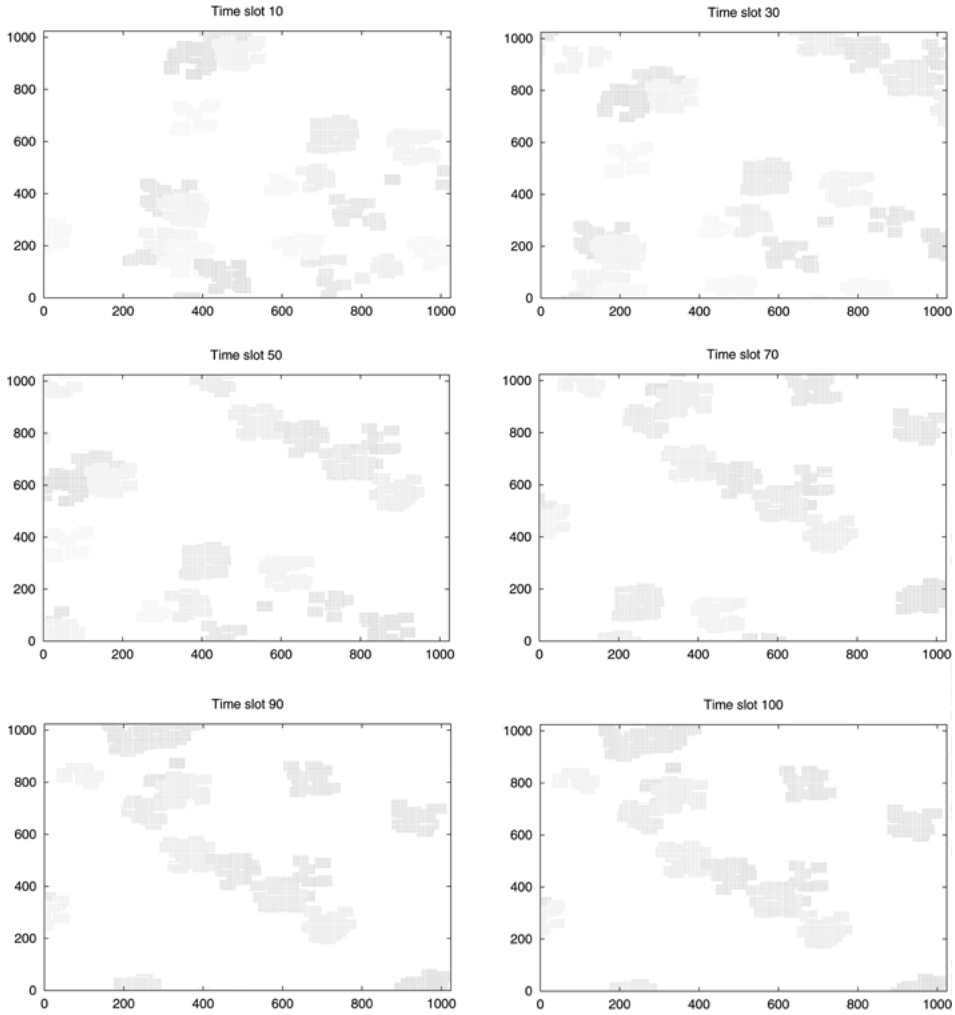


Figure 3. Static objects and a scene-observer moving from south-west to north-east. The workspace is 2000×2000 units and the length of the side of the observer's window is 1024 units.

distribution of new objects per time slot was Normal with mean $\mu = 0$ and mean square deviation $\sigma = 0$. No objects are deleted during the scene lifetime, since the distribution of their deletion time was also Normal with mean $\mu = 100$ and mean square deviation $\sigma = 0$.

The influence of the statistical distributions at the resulting data sets is demonstrated more clearly in the scenarios presented in figures 4 and 5. Both illustrate the initially normal distribution of moving objects, which are generated near the workspace center. The objects are moving, zooming in and out and rotating randomly. The randomness of the

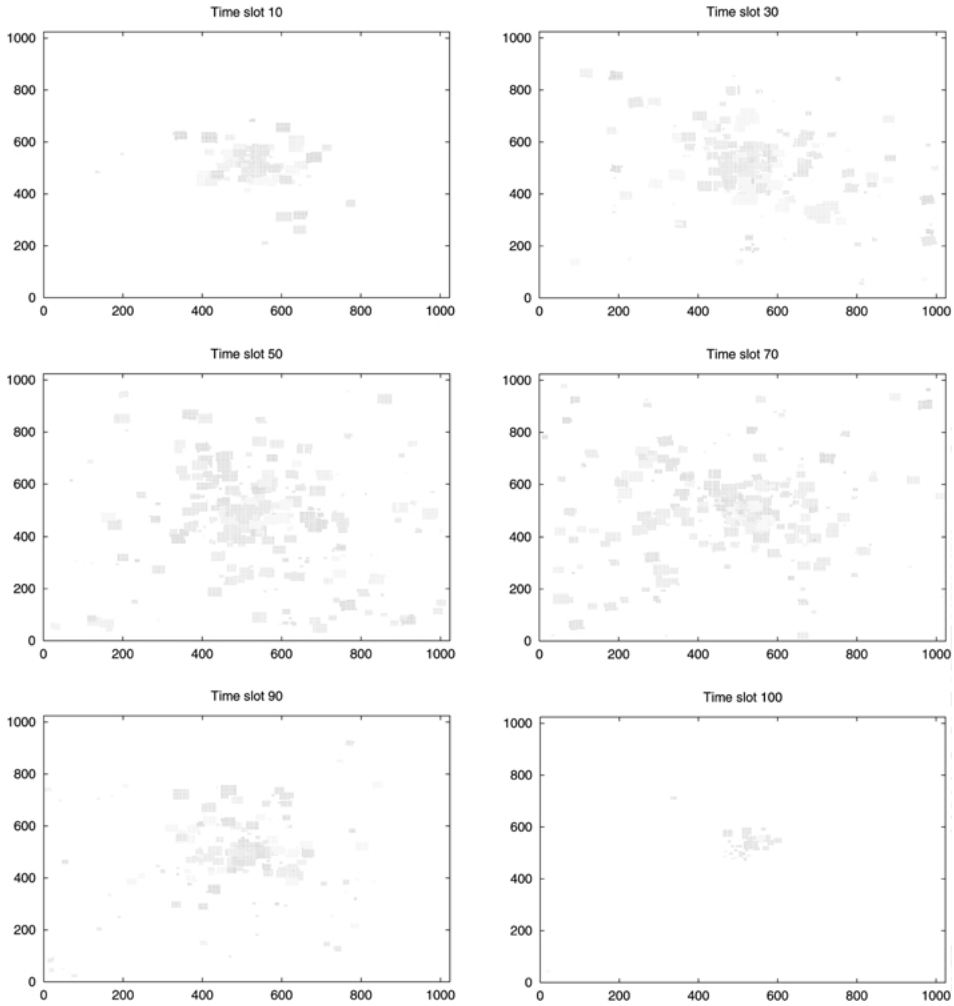


Figure 4. Moving objects that are generated by the workspace center and the deletion time distribution is $N(\mu = 10, \sigma = 90, min = \text{creation time of each object} + 1, max = num\_timeslots - 1)$.

speed, zoom and rotation-angle values results from the definition of their domain which allows positive and negative values with the same probability. The lifetime of the objects in the scenario of figure 4 is much smaller to the corresponding lifetime in the scenario of figure 5. This is a result of the Normal distribution obeyed by the variable of the object deletion time, which has mean $\mu = 10$ and mean square deviation $\sigma = 90$, while in the other scenario it has $\mu = 90$ and $\sigma = 10$.

The scene-observer's window covers the whole workspace. Thus, initially the scene-

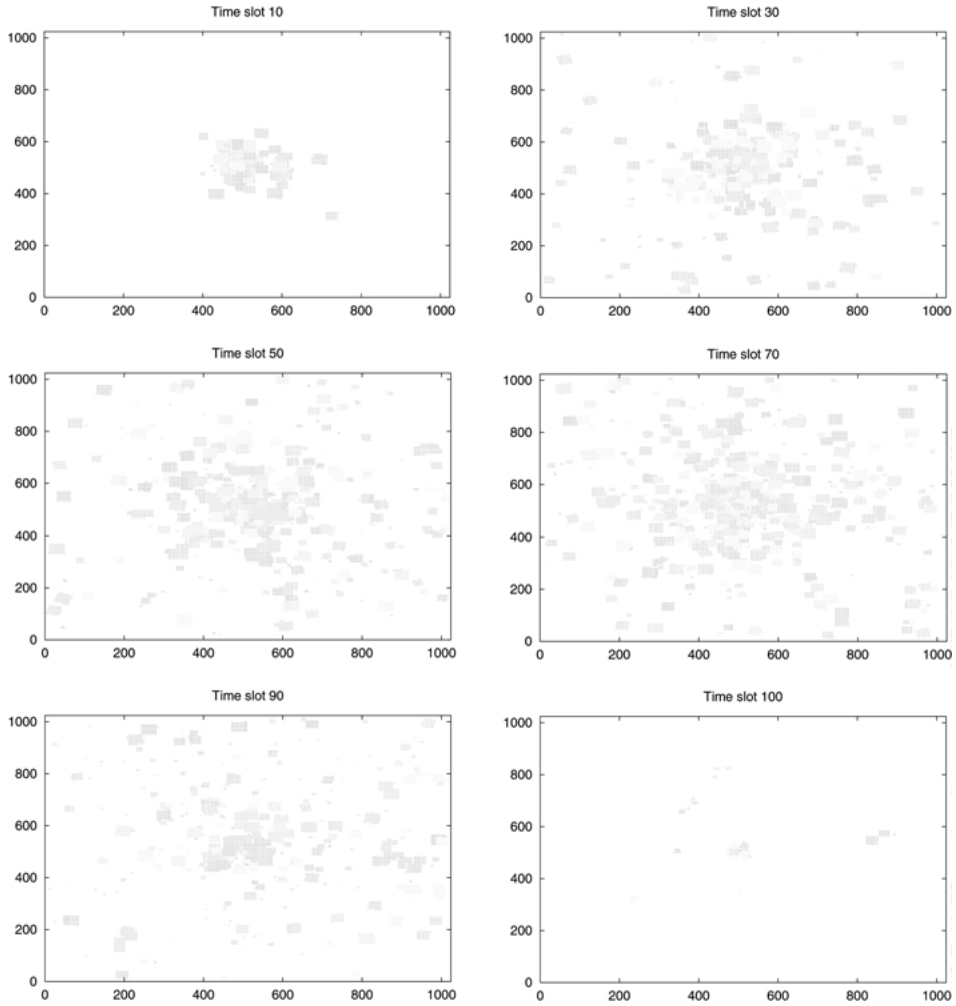


Figure 5. Moving objects that are generated by the workspace center and the deletion time distribution is $N(\mu = 90, \sigma = 10, min = \text{creation time of each object} + 1, max = num\_timeslots - 1)$.

observer may only zoom in. In order to make the evolution of moving objects more evident to the user, the scene-observer is forced to be static.

In figure 6, a fourth scenario is presented, where moving objects appear from the workspace bottom. The orientation of the objects on y-axis is South to North, since the domain of the speed for the y-axis is $[0, 20]$ and only positive values can be generated. The speed on the x-axis is random and uniformly distributed in the domain $[-10, 10]$. Many new objects are created at each time slot and they remain alive for a long time period. Therefore the objects have sufficient time to cross the whole workspace. When they arrive

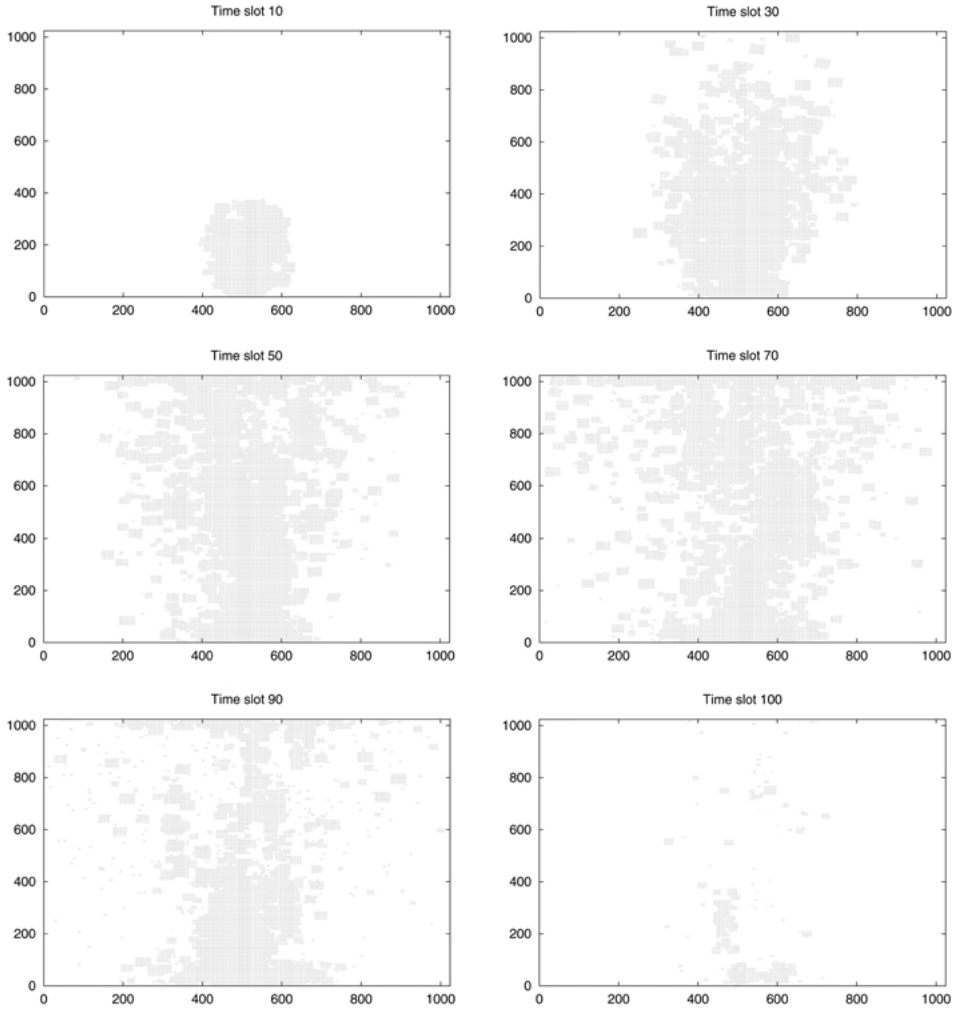


Figure 6. Moving objects that appear at the bottom and cross the workspace.

at the upper edge, G-TERD prevents them from leaving the workspace and thus they are scattered over the entire upper part of the workspace, until their deletion time comes.

The last scenario is a complex scenario and figure 7 depicts some of its snapshots. This scenario is based on two sub-scenarios and merges their data sets. The first sub-scenario produces static objects that live for the whole scene lifetime. It is similar to the scenario of figure 3 with the difference that the scene observer is static. The second sub-scenario produces moving objects that are generated along the y-axis and cross the workspace but each at a different speed. The non-fixed input parameters are similar to the corresponding

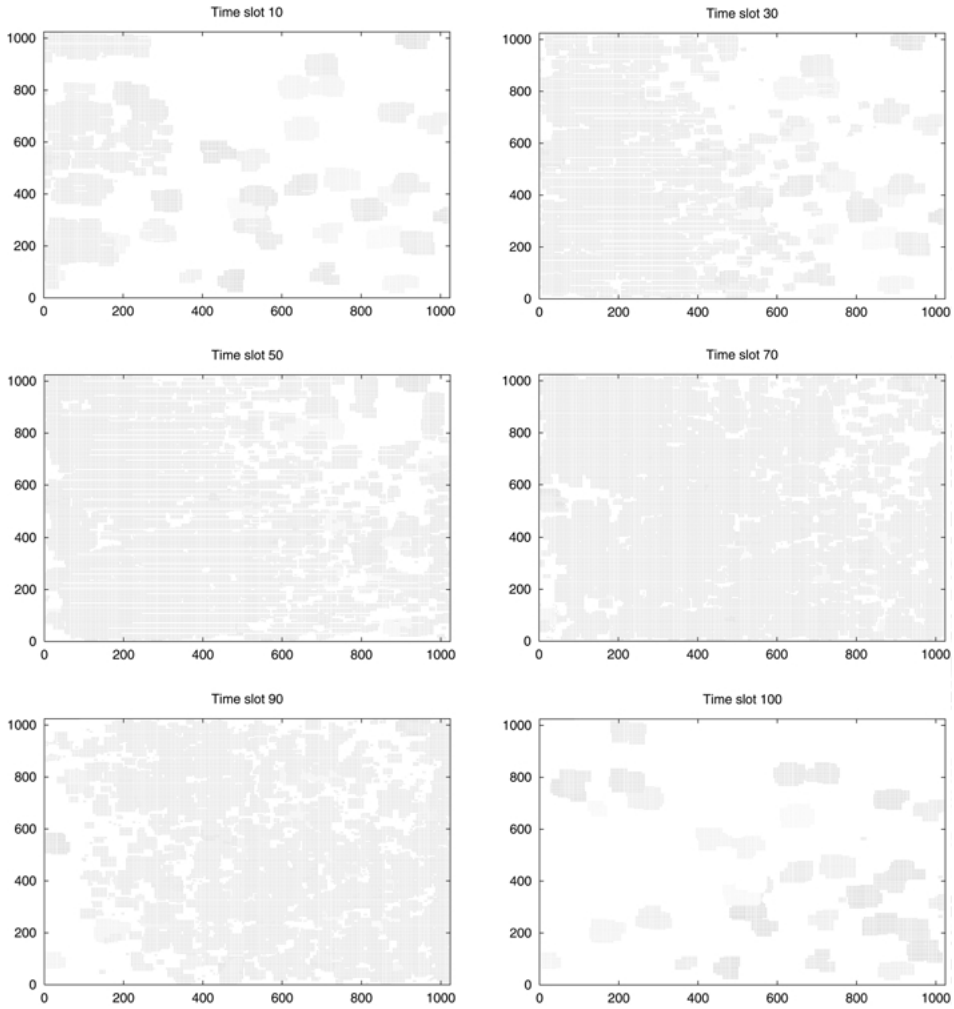


Figure 7. A complex scenario that combines two sub-scenarios.

inputs of the fourth scenario. There are only two differences. The first one is that the orientation of the moving objects is west to east and not south to north as it was in the fourth scenario. The second difference is that the distributions of the initial coordinates of the generated objects are $N(\mu = 0, \sigma = 0)$ on the x-axis and Uniform on the y-axis. In its current implementation, G-TERD allows the user to compose up to ten different individual scenarios and it can conjunct them by merging the resulting data sets.

Table 3. The non-fixed input parameters of the presented scenarios.

Parameter	*Scenario 1*	*Scenarios 2 and 3*	*Scenario 4*
X_max, *Y_max*	2000, 2000	1024, 1024	1024, 1024
max_num_subobjs	100	35	70
live_objs at $t = 0$	70	0	0
num_colors	16	16	1
subobj_side_max	100	70	70
percent_objs_static	100	0	0
percent_objs_slow	0	50	50
percent_objs_fast	0	50	50
[*speed_min*, *speed_max*] on each axis	$[10, 10]$ on x-axis $[10, 10]$ on y-axis	$[-10, 10]$ on x-axis $[-10, 10]$ on y-axis	$[-10, 10]$ on x-axis $[0, 20]$ on y-axis
[*zoom_min*, *zoom_max*]	$[0, 0]$	$[-1, 1]$	$[-1, 1]$
[*rotation_min*, *rotation_max*]	$[0, 0]$	$[-1, 1]$	$[-1, 1]$
$t_{max\_duration}$	5	20	20
distribution of the center of generated objects	U on both axes	$N(\mu = 512, \sigma = 10)$ on both axes	$N(\mu = 512, \sigma = 10)$ on x-axis $N(\mu = 0, \sigma = 10)$ on y-axis
deletion time of objects	$N(\mu = 100, \sigma = 0)$	$N(\mu = 10, \sigma = 90)$ for Scen.2 $N(\mu = 90, \sigma = 10)$ for Scen.3	$N(\mu = 100, \sigma = 5)$
new objects per time slot	$N(\mu = 0, \sigma = 0)$	$N(\mu = 3, \sigma = 1)$	$N(\mu = 3, \sigma = 1)$

7. Related work

In STDBs, the work on the generation of synthetic data is limited and only a few pioneering papers have recently addressed the topic of spatio-temporal data generators. However, the approach discussed in this paper is novel since all existing generators do not consider several important aspects of continuously changing regional data.

A spatio-temporal data set generator, called *Generate_Spatio_Temporal_Data* (GSTD), has been proposed in Theodoridis et al. [21]. It can generate moving points or rectangles, and starts by distributing their centers in the workspace according to certain distributions. After the initialization phase, there are three main parameters to control the evolution of the objects throughout time, according to a desired distribution. These parameters are: (a) the duration of object instances, which involves timestamp changes between consecutive instances; (b) the shift of the objects, which involves changes of spatial locations of the object centers; and (c) the resizing of objects, which involves changes of object sizes (only applicable to rectangular objects).

GSTD supports three alternative approaches for the manipulation of invalid object instances in cases where an object leaves the spatial data space. However, a limitation of GSTD approach is that the objects are moving almost freely in the workspace without taking into consideration the interaction between other objects, or any potential restrictions. In particular, it could be argued that in any possible scenario, the objects

are scattered all over the data space, or are moving in groups and the whole scene has the appearance of an unobstructed polymorphic cloud movement.

In order to create more realistic scenarios, Pfoser and Theodoridis [15] have extended the latter approach. They introduced an additional GSTD parameter to control the change of direction and they used static rectangles for simulating an infrastructure, where the scenario indicates that each moving object has to be outside of these rectangles.

In Saglio and Moreira [18] a generator for time-evolving points and rectangles which uses the modeling of fishing ships as a motivation is proposed. Ships are attracted by shoals of fish, while at the same time they are repulsed by storm areas. Fishes themselves are attracted by plankton areas. Ships are moving points, whereas shoals, plankton, and storm areas are moving regions. Although useful for testing of access methods, the original algorithm is highly specialized and turns out to be of limited use with respect to the demands of other real-world applications.

Finally, in Brinkhoff [4] and, in more detail, in Brinkhoff [5] a spatio-temporal generator for ''network-based'' moving objects is demonstrated. It combines a real network with user-defined properties of the resulting data set. The driving application is the field of traffic telematics and the presented generator satisfies exactly the requirements of this field. Important concepts of the generator are the maximum speed and the maximum edge capacity, the maximum speed of the object classes, the interaction between objects, the different approaches for determining the starting and the destination point of a moving object, and the re-computation of a route initiated by a reduced speed on an edge and by external events. A framework for preparing the network, for defining functions and parameters by the users, for data generation reporting are also presented. However, a disadvantage of the method is that it requires significant execution time for the generation of moving objects. Nevertheless, although the author claims that the approach allows considering the characteristics of a wide range of applications (mostly network-based), it is clear that many other interesting real world application examples such as meteorological phenomena, faunal phenomena, natural catastrophes, etc. cannot be simulated by using this tool.

On the other hand, various algorithms to generate spatio-temporal data sets have been presented, in order to facilitate the performance comparison of different access methods. Most of these algorithms are combinations, or specialized extensions of the previously cited generators, or do not imply any restrictions on the movement of the spatial objects. These approaches are not referred to any further.

Generating synthetic data sets has also been an active research field the last few years in the area of spatial databases. Several algorithms have been proposed to generate static spatial data (for instance, point, or rectangular), following a predefined distribution in the workspace. In order to compare different spatial join strategies, Guenther et al. [8] have proposed the exploitation of a spatial data generator, which produces data sets of 2-D rectangles based on user-defined parameters, such as cardinality, rectangle size, and coordinates' distributions. The target is to simulate real-life data sets (e.g., biotopes, forests, cities, and continents) by using synthetic ones. However, the suggested approach can generate only static spatial data sets.

8. Conclusion and future work

In this paper we investigated the issue of benchmarking STDBs. The first approach for data generation specifically designed for applications stemming from the field of time-evolving regional data is presented. The new generator, is called generator of time-evolving regional data (G-TERD). The basic concepts involved in the development, operation and use of G-TERD have been examined. These include the structure of complex 2-D regional objects, their color, maximum speed, zoom and rotation-angle per time slot, the influence of other moving or static objects on the speed and on the moving direction of an object, the position and movement of the scene-observer, the statistical distribution of each changing factor and finally, time.

G-TERD is very parametric and as the sample runs have demonstrated, it is very flexible in the simulation of a variety of real-world scenarios. Users can specify the time domain and the workspace extent, the maximum number of objects in the sample, their minimum and maximum speed, zoom and rotation-angle per time slot, the percentage of static and moving objects, the scene-observer's window size and, finally, the number of colors supported. Users may also decide about the statistical model of the cardinality of the new objects and sub-objects per time slot, the deletion time slot of an object or sub-object, their speed, zoom and rotation-angle, and finally the time period that must elapse before the next computation of an attribute (speed, zoom, color of an object or sub-object, etc.). Various statistical distributions are supported for that purpose.

The Web site of G-TERD provides access to the generator, its source code and some illustrative examples. G-TERD offers a framework for creating user-defined synthetic time-evolving regional data sets that can be used as a basic component for the experimental comparison of different STAMs. Also, G-TERD is an open source software [13]. The user is allowed to modify, improve or adapt the generator source code according to specific spatio-temporal application requirements. Thus, it is easy for the user to both examine the reliability of the underlying generalized data model and to repeat the experiments presented in this paper, in order to verify the claimed results.

The ultimate objective of this research is to provide a complete benchmarking environment system for quadtree-based STAMs. This general benchmarking framework is currently under construction and includes:

- A collection of recently proposed access methods: overlapping linear quadtrees (OLQs) [24], multiversion linear quadtree (MVLQ) [23], time-split linear quadtree (TSLQ) [25] and other quadtree-based STAMs which are under implementation,
- G-TERD for generating synthetic data sets that cover a variety of real life applications,
- A set of real time-evolving regional data acquired from Sequoia project that represent meteorological views,
- A set of spatio-temporal queries, including the five temporal window queries elaborated in Tzouramanis et al. [24], for extensive experimentation purposes,
- The visualization tool presented in Section 5 to visualize regional data sets, for illustrative purposes,

- A collection of all relevant output report logs from the efficient comparison of space requirements and query execution time of all the above STAMs.

In the future, we plan enhancing G-TERD to support a greater variety of distributions, such as skewed distributions or correlative 2-D distributions. Besides, the Web-based interface will be developed in order to both generate and make it possible to visualize time-evolving synthetic regional data.

Appendix

The basic C structures of G-TERD are presented in the following.

```
data structure obj
 BEGIN
 .id;                        // the object identification number
 .endtime;                   // the deletion time slot of the object
 .color_uniform;             // a flag denoting if the object has uniform color or not
 .color;                     // the object color if obj.uniform_color=true
 .color_duration;            // the period of time that must elapse before
                             // the next computation of the .color field
 .connected_parts            // a flag denoting if the object has connected and
                             // tangential sub-objects that have to remain connected
 .motion_type;               // a flag denoting if the object is static,
                             // or in either slow or fast movement
 .live_subobjs               // the number of ''live'' sub-objects per time slot
 .subobjs[ ]                 // the array of the obj.live_subobj sub-objects
 END
data structure subobj
 BEGIN
 .id;                        // the sub-object identification number
 .endtime;                   // the deletion time slot of the sub-object
 .loc[ ];                    // the location of its center point (it is an array
                             // consisting by two numbers, one for each spatial axis)
 .side;                      // the side length of the rectangular sub-object
 .color;                     // the sub-object color
 .color_duration;            // the period of time that must elapse before
                             // the next computation of the .color field
 .speed[ ];                  // the sub-object speed per time slot
                             // (it is an array consisting by two decimal numbers
                             // that correspond to the x- and y-axis)
 .speed_duration[ ];         // the period of time must elapse before
                             // the next computation of the .speed field
 .zoom;                      // the zoom of the sub-object per time slot
 .zoom_duration;             // similar use with the .speed_duration attribute
```

```
  .rotation;                  // the rotation-angle of the sub-object per time slot
  .rotation_duration;         // similar use with the .speed_duration attribute
  END
data structure observer
  BEGIN
  .loc[ ];                    // the same as in subobj
  .side;                      // the same as in subobj
  .follow_object_id           // the identification number of the object that the observer
                              // follows (if its value is null, non object is followed)
  .follow_object_id_duration  // the period of time which must elapse before
                              // the next computation of the .follow_object_id field
  .speed[ ];                  // the same as in subobj
  .speed_duration[ ];         // the same as in subobj
  .zoom;                      // the same as in subobj
  .zoom_duration;             // the same as in subobj
  .rotation;                  // the same as in subobj
  .rotation_duration;         // the same as in subobj
END
```

Notes

1. The generator Web site is: http://delab.csd.auth.gr/stdbs/g-terd.html
2. All figures in this paper appear in gray scale. Color versions can be found on the generator site.

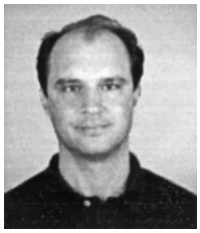
References

1. T. Abraham and J.F. Roddick. ''Survey of spatio-temporal databases,'' *Geoinformatica*, Vol. 3(1):61–99, 1999.
2. K.K. Al-Taha, R.T. Snodgrass, and M.D. Soo. ''Bibliography on spatiotemporal databases,'' *International Journal of Geographical Information Science*, Vol. 8(1):95–103, 1994.
3. D. Bitton, D.J. DeWitt, and C. Turbyfill. ''Benchmarking database systems: a systematic approach,'' *Proceedings 9th VLDB Conference*, 8–19, Florence, Italy, 1983.
4. T. Brinkhoff. ''Generating network-based moving objects,'' *Proceedings 12th International Conference on Scientific and Statistical Database Management (SSDBM'00)*, 253–255, Berlin, Germany, 2000.
5. T. Brinkhoff. ''A Framework for generating network-based moving objects,'' *Geoinformatica*, Vol. 6(2):153–180, 2002.
6. M.J. Carey, D.J. DeWitt, and J.F. Naughton. ''The OO7 benchmark,'' *Proceedings 1993 ACM SIGMOD Conference*, 12–21, Washington, DC, 1993.
7. M.H. Dunham, R. Elmasri, M.A. Nascimento, and M. Sobol. ''Benchmarking temporal databases a research agenda,'' Technical Report 95-CSE-20, Department of Computer Science and Engineering, Southern Methodist University, December 1995.
8. O. Guenther, P. Picouet, J.-M. Saglio, M. Scholl, and V. Oria. ''Benchmarking spatial joins a la carte,'' *International Journal of Geographical Information Science*, Vol. 13(7):639–655, 1999.
9. C. Gurret, Y. Manolopoulos, A. Papadopoulos, and P. Rigaux. ''BASIS: a benchmarking approach for spatial index structures,'' *Proceedings Workshop on Spatiotemporal Database Management (STDBM'99)*, 152–170, Edinburgh, Scotland, 1999.

10. J. Gray (ed.). The Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufmann, 2nd edition, 1993.
11. Y. Manolopoulos, Y. Theodoridis, and V.J. Tsotras: *Advanced Database Indexing* (Chapter 7: Spatiotemporal Access Methods), Boston, Kluwer Academic Publishers, 1999.
12. M.A. Nascimento, J.R.O Silva, and Y. Theodoridis. ''Evaluation for access structures for discretely moving points,'' *Proceedings of the International Workshop on Spatio-Temporal Database Management (STDBM'99)*, 171–188, 1999.
13. The Open Source Initiative. http://www.opensource.org, valid as of June 2001.
14. J. Pei, R. Mao, K. Hu, and H. Zhu. ''Towards data mining benchmarking: a testbed for performance study of frequent pattern mining,'' *Proceedings 2000 ACM SIGMOD Conference*, 592, Dallas, TX, 2000.
15. D. Pfoser and Y. Theodoridis. ''Generating semantics-based trajectories of moving objects,'' *Proceedings Workshop on Emerging Technologies for Geo-Based Applications*, Ascona, Italy, 2000.
16. M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. ''The Sequoia 2000 Benchmark,'' *Proceedings 1993 ACM SIGMOD Conference*, 2–11, Washington, DC, 1993.
17. T. Sellis, M. Koubarakis et al. (Eds.). *Spatiotemporal Databases—the Chorochronos Project*, Springer Verlag (LNCS series), in print.
18. J.-M. Saglio and J. Moreira. ''Oporto: a realistic scenario generator for moving objects,'' *Geoinformatica*, Vol. 5(1) 71–93, 2001.
19. M.D. Soo. ''Generating temporal data: a white paper,'' Technical Report, Department of Computer Science and Engineering, University of South Florida, 1997.
20. Y. Theodoridis and M.A. Nascimento. ''Generating spatiotemporal datasets on the WWW,'' *ACM SIGMOD Record*, Vol. 29(3):39–43, 2000.
21. Y. Theodoridis, J.R.O. Silva, and M.A. Nascimento. ''On the generation of spatiotemporal datasets,'' *Proceedings 6th Symposium on Spatial Databases (SSD'99)*, 147–164, Hong Kong, China, 1999.
22. Y. Theodoridis, T. Sellis, A. Papadopoulos, and Y. Manolopoulos. ''Specifications for efficient indexing in spatiotemporal databases,'' *Proceedings 7th Conference on Statistical and Scientific Database Management Systems (SSDBM'98)*, 123–132, Capri, Italy, 1998.
23. T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos: ''Multiversion linear quadtree for spatio-temporal Data,'' *Proceedings 4th East-European Conference on Advanced Databases and Information Systems (ADBIS-DASFAA'00)*, 279–292, Prague, Czech Republic, 2000.
24. T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos: ''Overlapping linear quadtrees and spatio-temporal query processing,'' *The Computer Journal*, Vol. 43(4):325–343, 2000.
25. T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos: ''Time split linear quadtree for indexing and querying image databases,'' *Proceedings 8th International Conference on Image Processing (ICIP'01)*, Thessaloniki, Greece, 2001.
26. J. Zobel, A. Moffat, and K. Ramamohanarao: ''Guidelines for presentation and comparison of indexing techniques,'' *ACM SIGMOD Record*, Vol. 25(3):10–15, 1996.

Theodoros Tzouramanis received his 5-year B.Eng. (1996) in Electrical and Computer Engineering and his Ph.D. (2002) in Informatics from the Aristotle University of Thessaloniki. His main research interests are access methods and query processing for spatio-temporal databases. Currently, he is serving in the Greek army.

Michael Gr. Vassilakopoulos was born in Thessaloniki, Greece in 1967. He received a B.Eng. (1990) in Computer Engineering and Informatics from the University of Patras and a Ph.D. (1995) in Electrical and Computer Engineering from the Aristotle University of Thessaloniki. His B.Eng. thesis deals with algorithms of computational geometry and his doctoral thesis deals with spatial databases. Apart from the above, his research interests include access methods, spatio-temporal databases, WWW databases, multimedia and GIS. He has published 30 papers in refereed scientific journals and conference proceedings. He has been with the Department of Applied Informatics of the University of Macedonia, the Department of Informatics of the Aristotle University of Thessaloniki and the Department of Informatics of the Technological Educational Institute of Thessaloniki. Currently, he is an informatics engineer at the Greek Public Administration.



Yannis Manolopoulos was born in Thessaloniki, Greece in 1957. He received a B.Eng (1981) in Electrical Engineering and a Ph.D. (1986) in Computer Engineering, both from the Aristotle University of Thessaloniki. Currently, he is Professor at the Department of Informatics of the latter university. He has been with the Department of Computer Science of the University of Toronto, the Department of Computer Science of the University of Maryland at College Park and the University of Cyprus. He has published over 100 papers in refereed scientific journals and conference proceedings. He is co-author of a book on Advanced Database Indexing by Kluwer. He is also author of two textbooks on Data Structures and File Structures, which are recommended in the vast majority of the computer science/engineering departments in Greece. He served/serves as PC Co-chair of the 8th National Computer Conference (2001), the 6th ADBIS Conference (2002) and the 8th SSTD Symposium (2003). Also, currently he is vice-chairman of the Greek Computer Society. His research interests include access methods and query processing for databases, data mining, and performance evaluation of storage subsystems. Further information can be found at http://delab.csd.auth.gr