Animating Spatiotemporal Constraint Databases

Jan Chomicki[1], Yuguo Liu[2], and Peter Revesz[2]

[1] Monmouth University, West Long Branch, NJ 07764, USA
chomicki@monmouth.ed,
WWW home page: http://www.monmouth.edu/~chomicki
[2] University of Nebraska-Lincoln, Lincoln, NE 68588, USA
revesz@cse.unl.edu,
WWW home page: http://cse.unl.edu/~revesz

Abstract. Constraint databases provide a very expressive framework for spatiotemporal database applications. However, animating such databases is difficult because of the cost of constructing a graphical representation of a single snapshot of a constraint database. We present a novel approach that makes the efficient animation of constraint databases possible. The approach is based on a new construct: *parametric polygon*. We present an algorithm to construct the set of parametric polygons that represent a given linear constraint database. We also show how to animate objects defined by parametric polygons, analyze the computational complexity of animation, and present empirical data to demonstrate the efficiency of our approach.

1 Introduction

Spatiotemporal databases have recently begun to attract broader interest [10, 12, 27]. While the temporal [4, 23, 24] and spatial [14, 28] database technologies are relatively mature, their combination is far from straightforward. In this context, the constraint database approach [15] appears to be very promising. Constraint databases provide a uniform framework for modeling arbitrarily high dimensional data, they are thus naturally suited for temporal, spatial, and spatiotemporal database applications. Constraint databases are similar to relational databases: They enjoy formal, model-theoretic semantics and support a variety of well-established query languages like relational algebra, relational calculus, and variants of Datalog.

Spatiotemporal databases applications, like modelling continuous evolution or change in natural and social phenomena, lead to a new mode of user interaction. The user would like to be able to *animate* the spatiotemporal objects present in the database by viewing their consecutive snapshots. The need to support animation efficiently adds new requirements to the underlying database engine. Snapshots of spatiotemporal objects have to be displayed quickly. That's where the constraint database technology falls somewhat short. In that approach, a snapshot of a spatiotemporal object is a spatial object, represented *implicitly* using a conjunction of inequalities (for polyhedral objects the inequalities are linear). Such a representation cannot be immediately displayed on computer screen: It has first to be converted to an *explicit* boundary representation. The conversion is relatively time-consuming. It has also to be repeated for every time instant in the animation. As a result, animation with a fine time granularity becomes slow and the ability to do real-time animation is limited.

To make the animation of spatiotemporal objects more efficient, we propose to use a separate data model just for the display purposes (the decoupling of retrieval and display was postulated in the context of spatial query languages by Egenhofer [8, 9]). For linear constraint databases, this model is a natural generalization of the spaghetti data model [18], called the *parametric spaghetti data model* (introduced in [5]). The basic modelling construct of the latter model is a *parametric polygon*. Using parametric polygons the conversion from an implicit constraint representation to an explicit display representation is broken into two stages: the construction of parametric polygons and their instantiation with consecutive time instants followed by display. The first stage, responsible for the bulk of the conversion, is now done only once, which leads to very substantial time savings and makes animation much more efficient. In this paper, we describe the mapping from linear constraint databases to parametric polygons and show using empirical data the superiority of this approach.

The plan of the paper is as follows. In section 2 we introduce the basic notions of linear constraint databases and the parametric spaghetti data model. In section 3 we present the mapping used for the construction of parametric polygons. In section 4 we show two basic animation algorithms for spatiotemporal objects and compare them empirically. We also discuss the issue of efficient display of snapshots. In section 5 we discuss related work and in section 6 we present conclusions and speculate about future work.

2 Basic Concepts

2.1 The Linear Constraint Data Model

In the *linear constraint data model* [15, 18, 25] each database consists of a finite set of *constraint relations*. Each constraint relation consists of a finite set of constraint tuples. Each constraint tuple is a conjunction of linear constraints over the attribute variables of the relation. For example, suppose that *desert* is the following constraint relation:

```
Desert(x, y, t) :- x >= 0, y >= 0,
                   x - t <= 10, x + y <= 20,
                   t >= 0, t <= 10.

Desert(x, y, t) :- x >= 0, y >= 0,
                   x + y + t <= 30,
                   t >= 10, t <= 20.
```

The intended meaning of each of the above constraint tuples is a set of polygons, with one polygon for each time instance satisfying the inequalities on

t. In this way, one can represent, for example, how the boundaries of a desert area are changing over time. For example, at times $0, 10$ and 20 the desert area would look as shown in Figure 2.1. At time 0 the area is the polygon shown in solid lines. Between time 0 and 10 the right side of the desert extends until at time 10 the area becomes a triangle. Then the shape remains a triangle but it expands further until at time 20 the hypotenuse becomes the dotted line shown in the figure.

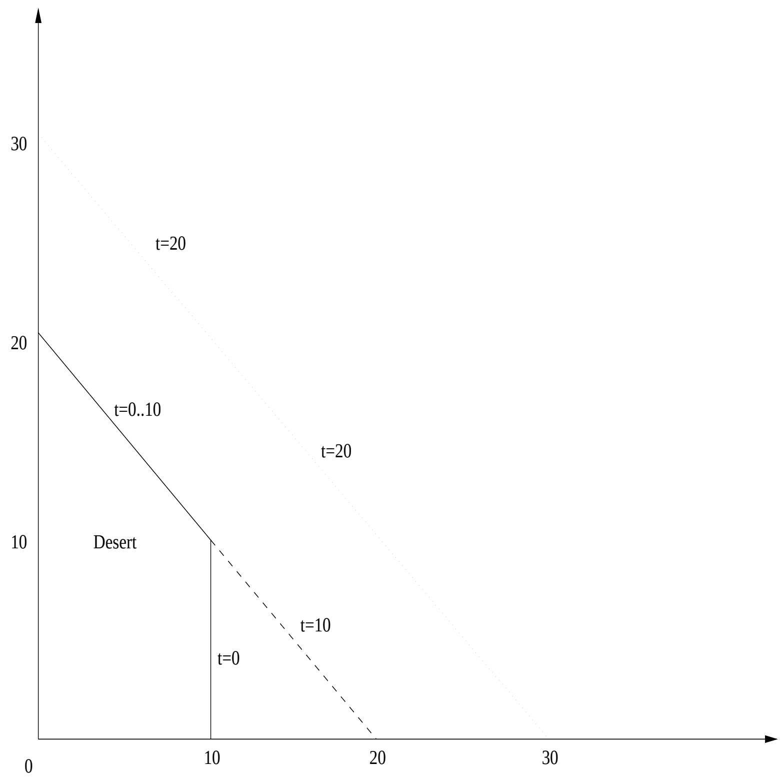

Fig. 1. An Example of Desert Area Changing

A more complex example with many constraint tuples is a relation $City$ that describes the expansion of the area of a city over time. Each constraint tuple represents one small region of the city. We created such a constraint relation that approximates the area of the city of Lincoln, Nebraska for the years between 1950 and 1990. The snapshot of the city for year 1990 is shown in Figure 2.1.

Although the linear constraint data model allows any number of attribute variables to be constrained by conjunctions of linear constraints, in this paper we assume that three distinguished variables are linearly dependent only on

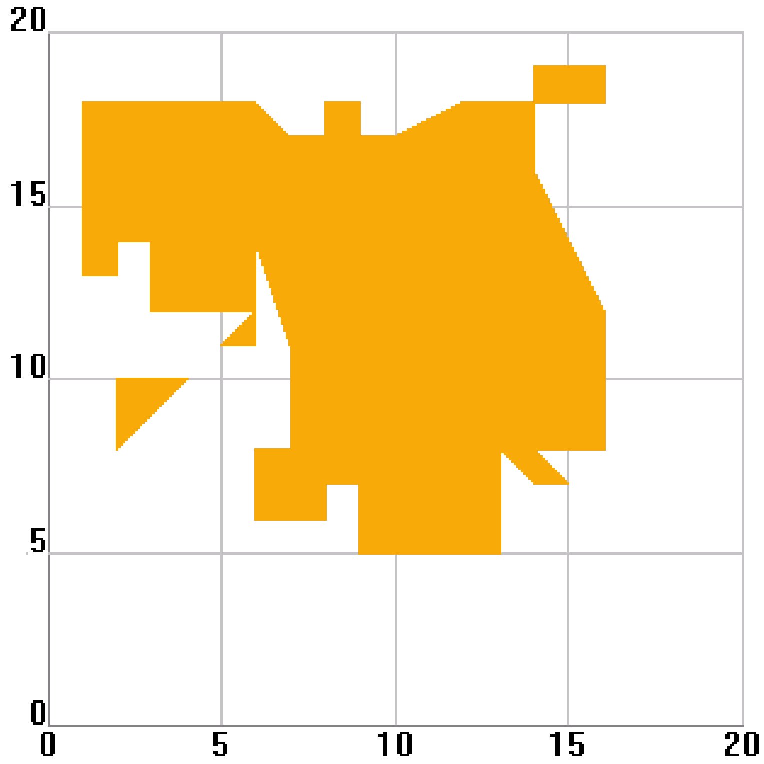

Fig. 2. City Area Snapshot at t = 1990

each other, namely x and y for the dimensions in the plane and t for time. Other attribute variables may be present and dependent on each other but must be independent of x, y and t. The animation algorithm refers only to those three variables and therefore, for simplicity, we assume that they are the only variables in constraint tuples. As we are dealing with continuous movement in the Euclidean plane, we fix the domain of all the variables to be the set of real numbers.

A *snapshot* of a constraint tuple over x, y and t is obtained by instantiating the variable t to some specific value t_0. A snapshot of a constraint relation consists of snapshots of all the tuples instantiated to the same value t_0.

2.2 The Spaghetti Data Model

The spaghetti data model [18] is a very popular model for representing spatial databases for CAD (Computer Aided Design) and GIS (Geographic Information Systems)[28]. Depending on the dimension K of the data it is possible to be more specific and to talk the K-spaghetti data model. In this paper we assume that $K = 2$, because in GIS applications the objects of interest are planar. Hence we in our paper spaghetti will mean 2-spaghetti unless otherwise specified.

In the spaghetti data model we can represent only spatial objects that are composed of a finite set of closed polygons. As a matter of fact, each spatial object can be decomposed into a set of triangles (some are degenerate triangles like line segments or points) where each triangle is represented by its three corners in a single relational database table. There are many good algorithms from computational geometry for triangulating polygons [1]. In this paper, we consider *only polygons which are triangles.*

Example 1. Let us consider the polygonal figure in Figure 1.

In the spaghetti model the figure in Figure 1 is represented by the relation in Table 1.

Table 1. Triangular Representation

ID	x	y	x'	y'	x''	y''
p1	10	4	10	4	10	4
l1	5	10	9	6	9	6
l1	9	6	9	3	9	3
t1	2	3	2	7	6	3
r1	1	2	1	11	11.5	11
r1	11.5	11	11.5	2	1	2
p2	3	5	3	8	4	9
p2	4	9	7	6	3	8
p2	3	5	7	6	3	8

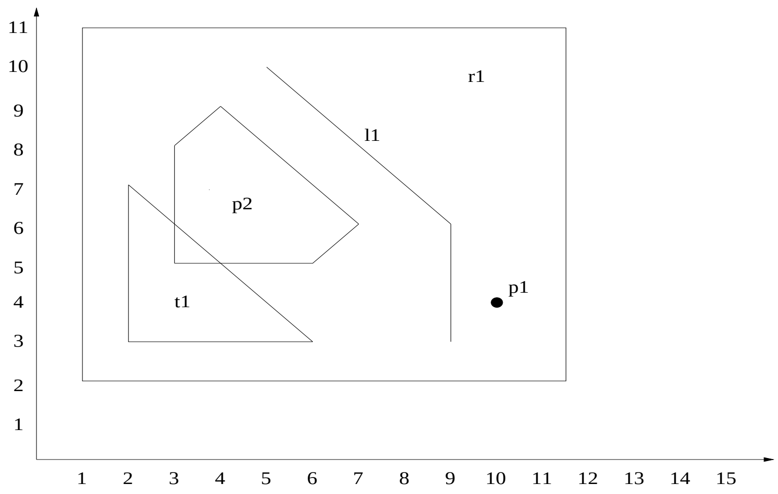

Fig. 3. A polygonal figure

Note that the rectangle is represented by two and the pentagon by three triangles.

The abstract semantics of a spaghetti data model is for each object the set of points (in two dimensions) that belong to the area of the plane that is within any of the triangles associated with that object.

2.3 The Parametric Spaghetti Data Model

The *parametric spaghetti data model* provides an alternative representation of linear constraint databases. The parametric spaghetti data model uses *parametric polygons*. For example, the constraint tuple for the *desert* relation (Figure 2.1) can be represented in the parametric spaghetti data model as the table in Table 2.

Table 2. Parametric Spaghetti Data Representation of the Desert Tuple

x	y	x'	y'	x''	y''	$FROM$	TO
0	0	0	20	10+t	10-t	0	10
0	0	10+t	0	10+t	10-t	0	10
0	0	10+t	0	0	10+t	10	20

The parametric spaghetti model extends with a temporal parameter t the spaghetti model. The range of the parameter t is given for each row of the above table. Each row is a parametric triangle. The vertices of this triangle are defined as linear functions of t. A parametric polygon can be represented by a number of parametric triangles. A snapshot of a parametric triangle is the triangle obtained by instantiating the variable t to some value t_0 that is in the interval $(FROM, TO)$. The meaning of a parametric triangle is the set of all its snapshots.

Using parametric polygons one can represent spatiotemporal objects that are defined using linear arithmetic constraints [5]. The latter objects need, however, to satisfy an additional condition: all of their snapshots have to be bounded. The objects may still be unbounded in time.

What kind of change can be represented using parametric polygons? Due to the restrictions to *linear* functions of t, only *fixed-speed* continuous transformations can be modeled. The transformations include translation and scaling, so continuously moving, growing, or shrinking polygons may be represented. Objects may appear and disappear finitely many times (each incarnation will be modeled as a different parametric polygon). They can also finitely many times change their attributes like color or shading. However, only spatial extents can change continuously in this model.

To implement parametric polygons, one does not need to extend the relational data model. Each linear function of t can be represented using exactly two coefficients. If *pair* is not available as a data type, one simply doubles the number of attributes holding spatial data.

3 Constructing the parametric representation

In this section we show how to construct a representation in the parametric spaghetti model of a spatiotemporal object defined using linear arithmetic constraints over x, y, and t.

Let $w(x,y,t)$ be a generalized tuple over x, y, and t. It represents a polyhedron P with finitely many extreme points. In order to be able to construct an equivalent parametric representation, we require that $w(x,y,t_0)$ describes a closed bounded polygon for every time instant t_0. Otherwise, the snapshot cannot be represented in the parametric spaghetti model. Note that a spatiotemporal object can be unbounded but only in one dimension: t.

Mapping Algorithm:

1. Determine the extreme points and the faces of P [22].
2. Determine all the intersections of the faces of P. Each such intersection is a line and can be described as a system of two linear equations in x, y, and t (the faces are described by single linear equations). From this system obtain the equation relating x and t (eliminate y) and the one relating y and t (eliminate x). We will call those equations the *characteristic equations* of the line. For each line, determine which extreme points lie on it (there may be 1 or 2).
3. Let $t_0, \ldots, t_k$ be the time coordinates of the extreme points of P, sorted in ascending order. Duplicates are ignored. If there is a point in P whose

t-coordinate is greater than t_k, then P is unbounded in the t-dimension and all extreme points with $t = t_k$ are marked "right special". Symmetrically, if there is a point in P whose t-coordinate is less than t_0, all extreme points with $t = t_0$ are called "left special".

4. For every left-open interval $I = (t_i, t_{i+1}]$, repeat the following:
   (a) Denote by P_I the slice of P that contains all the points of P whose time coordinates are in I. P_I is a polytope, possibly with some vertices that were not among the extreme points of P. The vertices of P_I are obtained by substituting t_i (or t_{i+1}) in the characteristic equations of each line of P to obtain the x and y coordinates and checking whether the resulting point is in P. The lines for which the check is positive for both t_i and t_{i+1} are marked as "relevant to P_I". All the vertices of P_I have t-coordinates equal to t_i or t_{i+1} (by construction).
   (b) Among the edges of P_I pick those that lie on lines that are relevant to P_I (those edges will connect a vertex with $t = t_i$ with one that has $t = t_{i+1}$). For each such edge the characteristic equations of the corresponding line give a representation of one vertex of a parametric polygon.
   (c) Triangulate the obtained parametric polygon, for example by picking an arbitrary order of parametric vertices $v_0, v_1, v_2, v_3, \ldots$ and forming the triangulation $(v_0, v_1, v_2), (v_0, v_2, v_3), \ldots$. No new triangulation points are created. Each obtained triangle defines a parametric row T with $T.FROM = t_i$ and $T.TO = t_{i+1}$.
5. If there are any right special points, define $I = [t_k, +\infty)$ and:
   (a) Denote by P_I the slice of P that contains all the points of P whose time coordinates are in I. This is still a polyhedron but no longer bounded. The vertices of P_I are obtained by substituting t_k in the characteristic equations of each edge of P to obtain the x and y coordinates and checking whether the resulting point is in P and whether the line contains a point with $t > t_k$ (to exclude lines connecting two vertices with the same t).The lines for which both checks are positive are marked as "relevant to P_I".
   (b) The relevant lines of P_I do not contain edges but rather extreme rays of P_I (they are also extreme rays of P). For each such ray the characteristic equation of the corresponding line gives a representation of one vertex of a parametric polygon.
   (c) Triangulation is done is in the previous case. The attributes $T.FROM = t_k$, $T.TO = +\infty$.
6. The left-special points are treated symmetrically to right-special points.

Example 2. We apply the above construction to the constraint relation representing Figure 2.1. We obtain the following extreme points (x, y, t):

$$(0,0,0), (10,0,0), (10,10,0), (0,20,0), (0,0,10), (20,0,10), (0,20,10), (0,0,20),$$
$$(30,0,20), (0,30,20).$$

None of them is special. We obtain two slices: $(0, 10]$ and $(10, 20]$. Some of the faces are: $x + t = 10$ and $x + y = 20$. From the intersection of those two faces we obtain two characteristic equations $x = 10 + t$ and $y = 10 - t$ that describe one vertex of a parametric polygon. The remaining vertices are determined in a similar way. For the slice $(0, 10]$ we obtain a parametric quadrangle which is then triangulated into two triangles (in a time-independent way). Those triangles are represented by the first two rows in Table 2. The last row in this table represents the second slice $(10, 20]$.

4 Naive and Parametric Animation

By the *animation* of a linear constraint database relation R we mean the sequential display of its snapshots or spatial extents (the set of (x, y) points) at times $t_0, t_1, \ldots, t_n$ at the user's request. The user specifies the initial time t_0, the time period Δ and the number of steps n, with the implicit condition that $t_i = t_{i-1} + \Delta$ for each $1 \leq i \leq n$.

In this section we describe and compare two methods for the animation of linear constraint databases. These two methods are called the naive and the parametric animation methods.

4.1 Animation Methods

Naive animation method. The naive animation method works directly on linear constraint databases. It finds for each time instance t_i, by instantiating the variable t to t_i, a linear constraint database relation that has only two spatial variables, namely x and y. Each constraint tuple of this relation defines a convex polygon. The naive method finds a triangulation of each polygon and the vertices of each triangle. Finally, it displays the set of triangles.

Parametric animation method. The parametric animation method has a *preprocessing* step and a *display* step. In the preprocessing step it constructs a parametric spaghetti database representation of the linear constraint database using the algorithm outlined in section 3. This construction needs to be done only once, before any user requests. The construction also finds for each parametric triangle p a beginning time $t_{p.from}$ and an ending time $t_{p.to}$. Before $t_{p.from}$ or after $t_{p.to}$ the parametric triangle has no spatial extent and does not need to be displayed.

During the display step, which is done at the user's request, for each consecutive time instant t_i and for each parametric triangle it is checked first that t_i is between $t_{p.from}$ and $t_{p.to}$. Corresponding to the parametric triangles whose range includes t_i, the parametric method finds, by instantiating the variable t to t_i, a set of triangles defined by their corner vertices. Then it displays the set of triangles.

Both animation methods assume that the computer system used provides in its graphics library as a primitive a display module for triangles defined by their corner vertices. Such a primitive is common in computer systems. The two animation methods are summarized in Figure 4.

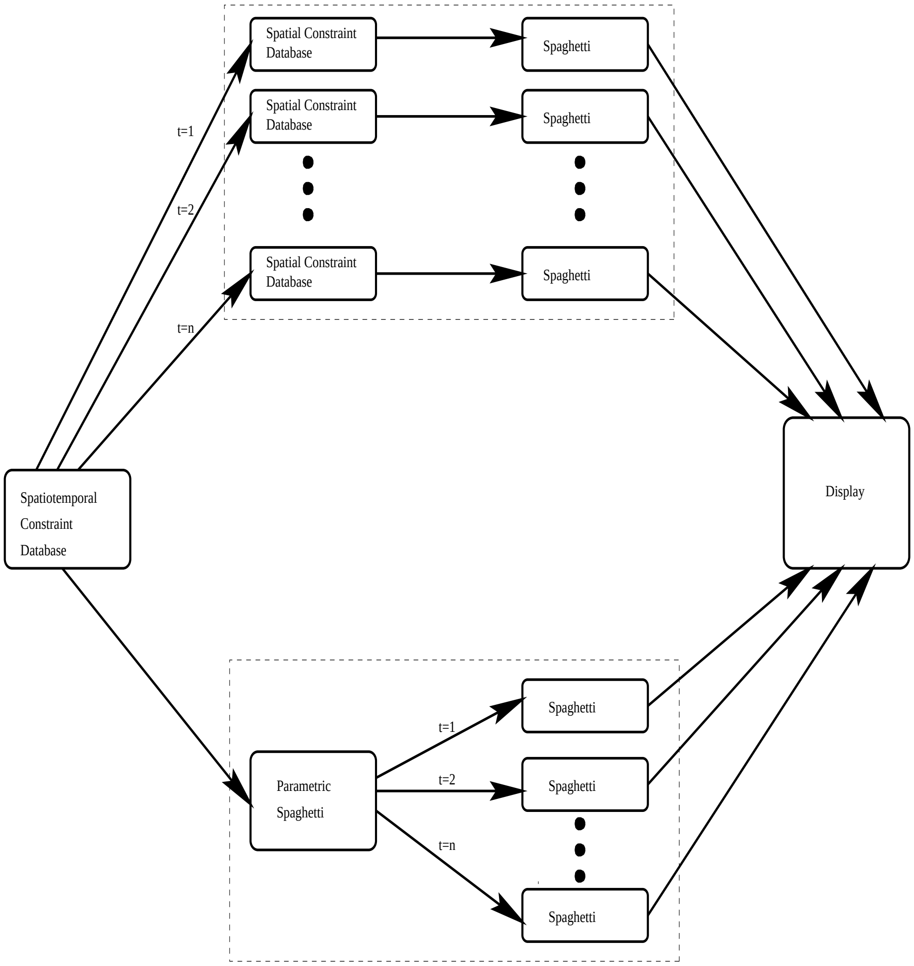

Fig. 4. Two Methods for the Animation of Spatiotemporal Databases

4.2 Computational Complexity

To compare the time complexity of both animation methods, we restrict ourselves to constraint relations consisting of single generalized tuples and their parametric representations. As each tuple is processed independently, the time of animating a constraint relation is then obtained in a straightforward way. Also, we assume that all processing is done in main memory.

We define first the following parameters:

- m - the number of constraints in the generalized tuple,
- k - the number of parametric tuples representing the generalized tuple,
- n - the number of animation steps.

Lemma 1. *$k \in \theta(m^2)$.*

Proof. From the description of the mapping in section 3, it is clear that $k \in O(m^2)$. To see that this bound is actually achieved, consider a family of pyramids whose "bottom" side is an m-gon and the remaining sides are isosceles triangles. Now take the intersection of this solid with a half-space whose bounding plane cuts across the solid by meeting its bottom side at an angle α, $0 < \alpha < \pi/2$. The result can be represented as a constraint tuple with $m + 2$ linear arithmetic constraints but the conversion produces a representation with $O(m^2)$ parametric tuples.

Lemma 2. *For any given time t, the number of parametric tuples containing t is in $O(m)$.*

The naive approach requires no preprocessing. Each snapshot can be constructed and displayed in $O(m \log m)$ time [19]. Thus the entire animation requires $O(nm \log m)$ time. In the parametric approach, a polyhedral representation of the generalized tuple (step 1 of the mapping) can be constructed in $O(m \log m)$ time [19]. The entire preprocessing thus takes $O(\max(k, m \log m))$ time, due to the size of the parametric representation. Displaying all the snapshots takes $O(nk)$ time. In the worst case ($k \in O(m^2)$), the total time required by parametric animation is $O(nm^2)$, compared to $O(nm \log m)$ for the naive one. However, this was a rather artificial example. We have found that in practice $k \in O(m)$, and then the parametric approach requires $O(m \log m + nm)$ time, compared to $O(nm \log m)$ for the naive one.

Parametric animation can be further improved. For example, relations with parametric triangles can be indexed using one of the interval-indexing methods like interval or priority search trees [19]. Triangles to be displayed at a given time can then be quickly retrieved (in time $O(\log k + m)$ which by Lemma 1 is $O(m)$). In this approach, the display time can thus be reduced to $O(nm)$ even when $k \in O(m^2)$. This is optimal. However, building of the index takes $O(k \log k)$ time. This cost can be amortized over multiple runs if the parametric representation is built once and animated several times. (Notice that such amortization is not possible in the naive approach.)

Another possible optimization that does not use an index consists of keeping track during the animation of the set of "live" parametric triangles (those whose interval (FROM,TO) contains the current time). If the number of "live" triangles at any given time is small (and thus one avoids having to look at the many triangles that are not "live"), the savings may be significant. Lemma 2 says that the size of the set of live triangles is in $O(m)$. This approach could be implemented by creating two copies of the parametric table: one sorted on the FROM column (the *FROM table*) and the other on the TO column (the *TO* table). The copies are then merged. When a *FROM* tuple is encountered during the merge, the parametric triangle is added to "live" set, otherwise (a *TO* tuple) the corresponding triangle is removed from this set. In addition, one needs to keep track of the current time and at every consecutive time instant instantiate all the "live" parametric triangles. This approach adds $O(k \log k)$ time to preprocessing. Now display consists of n display events and $O(k)$ interval endpoint events. Each of the latter involves insertion or deletion in a structure of size $O(m)$. Each of the former involves retrieving all objects in this structure. Thus display now takes $O(nm + k \log m)$ time. If the number of "live" triangles at any time can be bounded by a constant, this approach takes $O(n + k)$ time, as compared to $O(n \log k)$ using interval indexing.

4.3 Implementation Results

We implemented both animation methods in Microsoft Visual C++ on top of MLPQ (Management of Linear Programming Queries), a system developed and used at the University of Nebraska for querying linear constraint databases [21, 17]. The animation system is named ASTD (Animation of Spatiotemporal Databases).

The main library routines that we implemented include an $O(m^2)$-time routine to convert a linear constraint database with only x, y special variables into a set of triangles, and a separate routine to convert linear constraint databases with x, y and t special variables into a set of parametric triangles. No optimizations of the parametric approach were used. We implemented the animation routines that use the conversion routines and the Visual C++ MFC class library function for displaying triangles given their corner vertices.

We also implemented a graphical user interface module through which a user can call the ASTD system and specify the following parameters: the animation method to be used, the name of the linear constraint database relation to be animated, the initial time, the time period, and the number of time steps. ASTD also allows the setting of another parameter, namely the minimum delay time $d_{\min}$. The minimum delay time controls the speed of the animation in the sense that there must be at least $d_{\min}$ time between the issue of two commands to display a set of triangles. In our execution experiments the minimum delay time was set to be very small. Hence each snapshot was essentially displayed as soon as the animation algorithm calculated the corner vertices of the set of triangles to be displayed.

We measured the execution times for the animation of the two examples (desert, city area) described in section 2 using the same parameters. In addition, we have experimented with an even more complex example: a spatiotemporal representation of the geographic range of California gull. Two snapshots of this representation are shown in Figure 4.3.

The ASTD system ran on a 266 MHz Pentium II with 64M memory in a Windows NT environment. Table 3 compares the execution times (in milliseconds) of the two animation methods on the above three examples. For all examples, the parametric animation method was much faster. The missing entry in the last row is due to the fact that the computation ran out of available memory.

Table 3. Naive vs. Parametric Animation (Time in Millisecond)

Example	Number of Tuples/ Number of Time Points	Without Parametric Representations	With Parametric Representations
Desert	2 / 20	2,980	480
City	17 /20	17,860	1,680
Gull	131 / 60	-	12,470

4.4 Time Series

For completeness, we mention another possible approach to the problem of animating spatiotemporal databases. In this approach the time series of all the snapshots is precomputed and stored in a relational database. During the animation, consecutive snapshots are retrieved and displayed. This approach suffers from the fact that time granularity has to be fixed in advance. More significantly, the amount of data required by this approach is considerably higher than in the approaches described in the present paper. As a result, the animation data may have to be stored on disk, significantly increasing animation time.

5 Related work

Spatiotemporal data models and query languages. Spatiotemporal data models and query languages are a topic of growing interest. The paper [27] presents one of the first such models. In [10] the authors define in an abstract way moving points and regions. Apart from moving points, no other classes of concrete, database-representable spatiotemporal objects are defined. In that approach continuous movement (but not growth or shrinking) can be modeled using linear interpolation functions. In [12] the authors propose a formal spatiotemporal data model based on constraints in which, like in [27], only discrete change can be modeled. An SQL-based query language is also presented. We have proposed

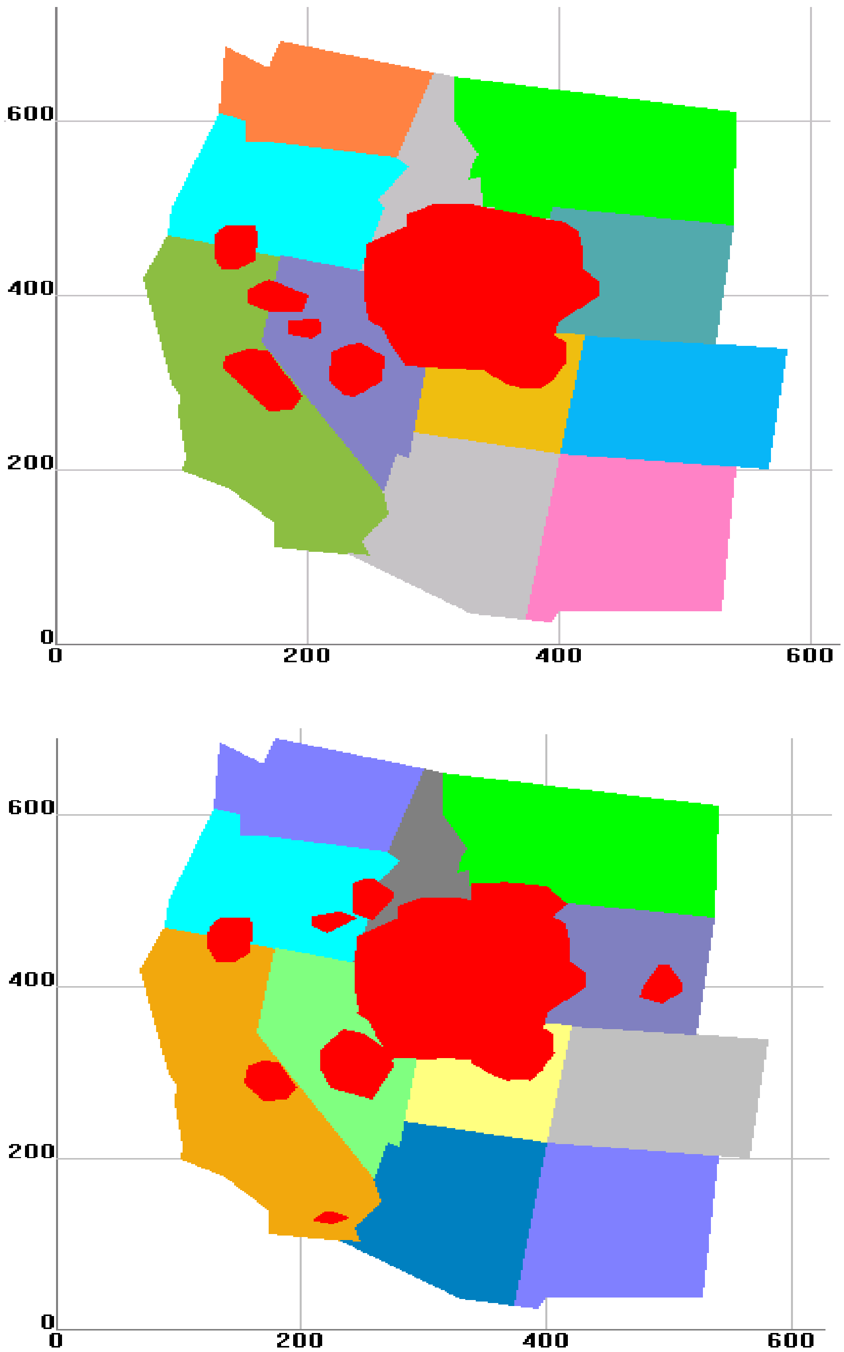

Fig. 5. California gull at t=0 (top) and t=31 (bottom).

elsewhere [7] a spatiotemporal data model based on parametric mappings of geometric objects. This model is also capable of modeling continuous change and avoids some of the closure problems of the parametric spaghetti model.

Computer animation. In this area many formalisms and systems for the specification of graphical animations have been proposed, including scripting languages [13]. The issues include, among others, producing realistic effects, animation language design, and making animation efficient and easy to build. The animations can typically be specified in several different ways (the following list is based on the system Maya (`http://www.aw.sgi.com`): by key frames with interpolation between them, procedurally, by inter-object constraints or by motion path (the path is specified by the user as a NURBS curve). To our knowledge, the work on computer animation has so far concentrated solely on the display level and the need to have a separate database representation of animations has not been identified.

Robotics and vision. Spatiotemporal applications abound in those areas. However, the emphasis is on finding robot trajectories (robot motion planning) or reconstructing object trajectories from a sequence of images (computer vision). Trajectories are not treated as objects which can be stored in a database, queried, or animated.

User interface design. Linear arithmetic constraints have been proposed as a language for user interface specification [2]. The main emphasis of this work is on dealing with constraint relationships (constraint hierachies) and efficient constraint solving. Constraints are not treated as database objects. Also, constraints are used to specify a single state of the interface, not a sequence of such states.

6 Conclusions and Future Work

We have shown that defining a separate display data model enhances the usability of spatiotemporal constraint databases by making their animation more efficient. This work opens many avenues for further research.

Computer animation. For more serious animation projects, it may be necessary to look into constraint languages that are more expressive than linear arithmetic constraints, e.g., polynomial constraints. For such constraints, new parametric representations have to be developed. We also believe that our approach can be further enhanced using techniques from computer graphics, animation, and computational geometry (more efficient construction of the parametric representation, more efficient display). In computer graphics, it is common to animate scenes with thousands of polygons per scene. If such animations are represented using constraints and stored in the database, it may become infeasible to perform preprocessing and display entirely in main memory. Then external memory algorithms [26] and index structures [16] need to be used.

Two-tiered data model. If the parametric 2-spaghetti data model is more suitable than the linear constraint database model, then perhaps the latter may be abandoned altogether? In fact, the parametric model can express some spatiotemporal objects which are not definable using linear constraints [5, 6]. However, the parametric model lacks important closure properties, e.g., it is not closed with respect to intersection [6]. This means that relations containing parametric polygons cannot be joined, although other operations like selection or projection can easily be supported. Therefore, we believe that the constraint model remains more suitable for querying. However, one should explore the option of storing the parametric polygons in a database and providing necessary indexing mechanisms. In this way *content-based* retrieval of animations can be supported.

Higher dimension. In this paper, we focused on two dimensional spatiotemporal problems which only had x, y and t as linearly dependent variables. However, many real-life spatiotemporal problems are three dimensional, that is, involve three spatial and one temporal variables that are linearly dependent on each other [20]. We need to develop a three-dimensional parametric representation and study translations of three-dimensional constraint databases to this representation. Also, the display of three dimensional parametric spaghetti databases on two dimensional computer screens has to be studied.

Multiple time granularities. It is natural to allow multiple time granularities: years, months, days, etc. To represent them in a constraint database one can use complex values in the time domain or multiple temporal attributes. The parametric representation would have to be suitably extended and the mapping from constraint databases to the parametric representation generalized.

Rotation. Neither the linear constraint data model nor the parametric spaghetti data model can describe rotation of objects. Since rotation is a fairly common movement of spatiotemporal objects, an extension has to be found of both data models that can describe rotational movements. We are currently investigating data models that add rotation without having to deal with general polynomial constraint databases.

General continuous change. It is natural to allow other attributes, apart from the spatial ones, to change continuously. For an example, consider continuous picture shading. To represent it, one would define the shade attribute as a (linear) function of time. No extension of the constraint data model is necessary to represent such an attribute. The parametric spaghetti model is easy to extend in this direction by allowing linear functions of time in non-spatial attributes. The change to the display algorithm is also minor. However, the mapping (section 3) from linear constraint databases to the parametric representation continues to work only under the assumption that spatial and non-spatial attributes are independent [3]. In practical terms, it means that all the points of a polygon have to change in exactly the same way.

Acknowledgments

The work of the first author was supported by NSF grant IRI-9632870. The work of the second and third authors was supported by NSF grants IRI-9632871 and IRI-9625055, and by a Gallup Research Professorship. The first author thanks

Anna Lipka of Alias/Wavefront for the information about Maya and Vassilis Tsotras for the references to the literature on interval indexing. The authors are grateful to the anonymous reviewers for their helpful comments.

References

1. M. Bern. Triangulations. In Goodman and O'Rourke [11], chapter 22, pages 413–428.
2. A. Borning, K. Marriott, P. Stuckey, and Y. Xiao. Solving linear arithmetic constraints for user interface applications. In *ACM Symposium on User Interface Software and Technology*, 1997.
3. J. Chomicki, D. Goldin, and G. Kuper. Variable Independence and Aggregation Closure. In *ACM Symposium on Principles of Database Systems*, pages 40–48, Montréal, Canada, June 1996.
4. J. Chomicki. Temporal Query Languages: A Survey. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic, First International Conference*, pages 506–534. Springer-Verlag, LNAI 827, 1994.
5. J. Chomicki and P. Z. Revesz. Constraint-Based Interoperability of Spatiotemporal Databases. In *International Symposium on Large Spatial Databases*, pages 142–161, Berlin, Germany, July 1997. Springer-Verlag, LNCS 1262.
6. J. Chomicki and P. Z.. Revesz. Constraint-Based Interoperability of Spatiotemporal Databases. *Geoinformatica*, 3(3), September 1999.
7. J. Chomicki and P. Z. Revesz. A Geometric Framework for Specifying Spatiotemporal Objects. In *International Workshop on Time Representation and Reasoning*, Orlando, Florida, May 1999.
8. M. Egenhofer. Why not SQL! *International Journal of Geographic Information Systems*, 6(2):71–85, 1992.
9. M. Egenhofer. Spatial SQL: A Query and Presentation Language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1), 1994.
10. M. Erwig, R.H. Güting, M. M. Schneider, and M. Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. In *ACM Symposium on Geographic Information Systems*, November 1998.
11. Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press, 1997.
12. S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-Temporal Data Handling with Constraints. In *ACM Symposium on Geographic Information Systems*, November 1998.
13. M. Gervautz and D. Schmalstieg. Integrating a scripting language into an interactive animation system. In *Computer Animation*, pages 156–166, Geneva, Switzerland, 1994.
14. R. H. Güting. An Introduction to Spatial Database Systems. *VLDB Journal*, 3(4):357–400, October 1994.
15. P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995.
16. P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, and J. S. Vitter. Indexing for Data Models with Constraints and Classes. *Journal of Computer and System Sciences*, 52(3):589–612, 1996.

17. P. Kanjamala, P.Z. Revesz, and Y. Wang. MLPQ/GIS: A Geographic Information System using Linear Constraint Databases. In *9th COMAD International Conference on Management of Data*, pages 389–393, Hyderabad, India, December 1998. Tata McGraw Hill.
18. J. Paredaens. Spatial Databases, The Final Frontier. In *International Conference on Database Theory*, pages 14–32, Prague, Czech Republic, January 1995. Springer-Verlag, LNCS 893.
19. F. Preparata and M. Shamos. *Computational Geometry*. Springer-Verlag, 1985.
20. J.. Raper. *Three Dimensional Applications in Geographical Information Systems*. Taylor & Francis, 1989.
21. P. Z. Revesz and Y. Li. MLPQ: A Linear Constraint Database System with Aggregate Operators. In *International Database Engineering and Applications Symposium*, pages 132–137. IEEE Press, 1997.
22. R. Seidel. Convex Hull Computations. In Goodman and O'Rourke [11], chapter 19, pages 361–375.
23. R. T. Snodgrass. Temporal Databases. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 22–64. Springer-Verlag, LNCS 639, 1992.
24. A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.
25. L. Vandeurzen, M. Gyssens, and D. Van Gucht. On the Desirability and Limitations of Linear Spatial Database Models. In *International Symposium on Large Spatial Databases*, pages 14–28, 1995.
26. J.S. Vitter. External Memory Algorithms and Data Structures. In J. Abello and J.S. Vitter, editors, *External Memory Algorithms and Visualization*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1999.
27. M. F. Worboys. A Unified Model for Spatial and Temporal Information. *Computer Journal*, 37(1):26–34, 1994.
28. Michael F. Worboys. *GIS: A Computing Perspective*. Taylor&Francis, 1995.