Interpolating Probability Values or Fuzzy Set Values for Uncertain Spatiotemporal Objects

Erlend Tøssebro
Department of Computer Science and Electrical Engineering
University of Stavanger
Stavanger, Norway
erlend.tossebro@uis.no

Abstract - **This paper looks at ways of quickly interpolating probability values or fuzzy set values for uncertain spatiotemporal objects that may change continuously over time. The paper starts with presenting a way to compute a tetrahedralization of an uncertain spatiotemporal object and using that to compute consistent interpolations. This approach also turns out to be able to create fairly good interpolations of the shape of the spatiotemporal object without needing an extra algorithm for this purpose. However, a naïve use of any tetrahedralization turns out to create interpolation artifacts in those objects that become significantly more or less uncertain with time. The paper then presents a way to overcome this issue at the cost of more processing.**

Keywords: Interpolation, Spatio-temporal uncertainty, Tetrahedralization

I. INTRODUCTION

In [17] and [18], a method for storing uncertainty in spatiotemporal data was presented. The method is based around storing a region in which the object might be, called the support (a term from fuzzy set theory [20]). For regions an additional region is stored that indicated where the object certainly is, called the core. [19] presents a method for quickly interpolating the probability functions for non-temporal two-dimensional spatial data using a Delaunay triangulation of the support. This method does not assume a particular probability function but is used to quickly generate approximate probability values where the function is unknown or very time-consuming to calculate.

This paper presents an algorithm for quickly interpolating probabilities or fuzzy set values that works for spatiotemporal objects as well. It is based on the uncertain time slice model for uncertain spatiotemporal objects as presented in [18] and described in section II.B and II.C. A time slice of a spatiotemporal object is a special case of a 3D object, and creating something like a Delaunay triangulation in three dimensions is significantly more complex than for the two-dimensional case in [19].

Two example applications of such a system are given below:

Example 1: If you are trying to determine the habitat of a species of plant or animal, you have to rely on observations of that plant or animal. Secondly, species often have areas in which they may appear but are not common. Such a habitat may be represented by a fuzzy set in which the degree of membership indicates whether the species is common or rare there. For animal habitats a value of 1.0 might indicate areas in which the animals usually live while a value close to 0 might indicate an area in which the animals may occasionally wander but do not remain for long. Because of both natural factors and human activities the habitats of both plants and animals change continuously. For instance the king crab is steadily spreading southwards along the northern European Atlantic coastline from where it was first introduced by the Russians.

Example 2: [9] mentions an example of uncertain spatiotemporal regions: The extent of kingdoms in the distant past. The evidence of the extent of past kingdoms and empires must be based on archaeological evidence and what textual evidence remains. This textual evidence was often written to glorify particular kings or emperors and may have exaggerated the size of their empires. These borders are often highly uncertain and changed with time. While the changes may have been discrete (conquests) when they occurred, we do not know exactly when they occurred and it is therefore better to visualize the extents as continuously changing, and use color shading to represent the estimated likelihood of the empire extending that far.

Another use for the method is as an interpolator for uncertain spatiotemporal objects. Like the algorithm in [16], this method produces an interpolation of an object between two snapshots. The original intention when this work was begun was to use the interpolation method from [16] to create the core and support boundaries and then tetrahedralize the support. However, it was discovered that the tetrahedralization algorithm itself was capable of producing a good interpolation, so interpolating the boundaries in advance was not necessary.

A third use of the algorithm is to create temporal version of a triangulated irregular network (TIN). This can be used to create a spatiotemporal field. If the entire TIN is updated at the same time the tetrahedralization generated by the algorithm in this paper can be used to create a spatiotemporal field model. The algorithm requires that the entire TIN is updated at the same time but does not require that the same measurement points are used each time. If the TIN is not updated at the same time, there are several algorithms already published that can be used (See section II.A). These have a longer running time but are more general.

II. RELATED WORK

This section is divided into three subsections because this paper draws on work in three different directions. 3D Triangulations is work on creating tetrahedralizations (the 3D equivalent of a triangulation) of 3D objects. This is used in this paper to create a representation of an uncertain time slice of the object as a 3D object. Spatiotemporal

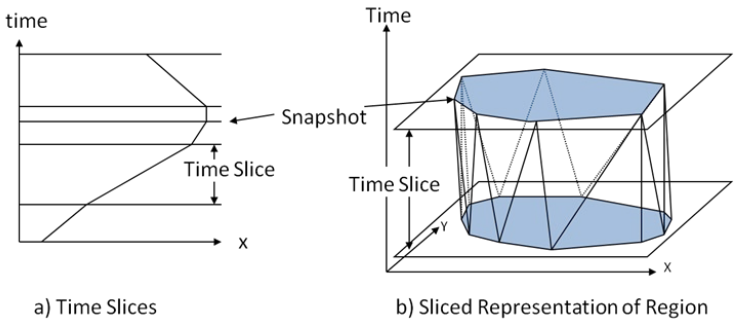


Figure 1: Sliced Representation of spatiotemporal objects

databases and uncertainty in spatiotemporal databases are the remaining two sections.

A. *3D Triangulation (tetrahedralization)*

The 3D Delaunay triangulation is the inverse of the 3D Voronoi diagram in the same manner as for 2D. This 3D Voronoi diagram can be computed in $O(n^2)$ time according to [3] page 161. According to [13], the Delaunay triangulation of any dimension can be computed in O(n*t) time where n is the number of points and t is the number of simplices in the final result. In most cases this is O(n2 + t*log(n)) according to [14]. The topic of generating a Delaunay tetrahedralization of general three-dimensional shapes has been extensively studied in the computational geometry community. [2] and [15] are other examples of papers studying variants of this problem. The problem in [2] resembles the problem in this paper as it looks on three-dimensional regions that have been split into slices so that the only vertices are in the top and bottom planes. It differs in that the boundary of the object is specified in advance whereas the algorithm proposed in this paper is free to set the boundary of the interpolation. Using tetrahedralizations for shape blending has been done in for instance [8], but this paper does not describe how to compute the tetrahedralization and has a very different focus from this work.

B. *Spatiotemporal databases*

The model for uncertainty in spatiotemporal databases presented in this paper will build on the time slice model for spatial data presented in [6].

If one assumes that one starts with a series of snapshots of the continuously moving object, a time slice is the time period between two snapshots. In each such period the object is assumed to change based on fairly simple functions. An example of this is shown in Figure 1a.

A point moves along a straight line. The end points of a line segment also move in a straight line, and the line segment itself cannot rotate. This restriction is there to ensure that the squares that are formed in space-time are planar rather than curved. Planar squares are much easier to deal with than curved ones. A line segment that rotate is transformed into two line segments each of which shrink to a point in the other snapshot like shown in Figure 1b.

In [16], a method for generating time slices like those from [6] from snapshots is presented. This works by building a 3D representation of the snapshot that is able to produce intermediate results for times between the snapshots. Because it is very unlikely that a line segment does not rotate at all, this method assumes that all line segments in one snapshot should be matched to points in the other like shown in Figure 1b.

C. *Spatiotemporal uncertainty*

[5] contains a model of uncertain points and shows some computations on this model. [11] contains a general model of spatiotemporal objects and [12] shows one example of how to use it for moving points. [10] describes a conceptual model of the different kinds of uncertainty that exist in a spatiotemporal database. [1] describes an algebra of uncertain moving points on a road network.

The basic method for representing uncertainty in spatial databases that is used in this paper was presented in [17]. In this representation an uncertain spatial object is represented as a support region, even if the object is a point or line. The support region is the region in which the object might possibly be. Region objects also have a core, which is the place where the object is guaranteed to be. Uncertain points and lines instead have a point or line of maximum probability which is the point or line in which the object is most likely to be.

In some applications (like the one from Example 1), one might want to compute the probability or fuzzy set value of the object in a location in the support. The method from [17] assumes that the probability falls from the core to the outer edge of the support according to some predefined function. One way to do this that ensures consistent answers is to create a triangulation of the support and use the triangulation to compute the probability value or fuzzy set value. Consistent answers in this case means that the answer to the question “Find the border of the area with at least 50% chance of being inside the object” is consistent with the answer to the question “find the probability of point X being inside the object”.

The probability function is assumed to take an input value that is between 0 and 1. This value indicates the relative distance to the edge of the support where 0 is at the edge of the support and 1 is at the edge of the core. To find the distance value of a point, compute the 3D plane defined by the triangle, with the value as the third dimension, and then find the distance as the height (value) of the plane at that point.

[19] presents a method for computing such a triangulation for a single snapshot using constrained Delaunay triangulation ([4]) as a basis. This paper will extend that method to an entire time slice.

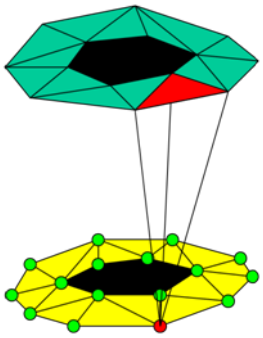


Figure 2: Matching a triangle to a point

III. Triangulating Spatiotemporal Objects

A database storing temporal geographic objects can be compared to a database storing three-dimensional spatial objects. This means that to extend the triangulation-based method of computing probabilities to the spatiotemporal case one would need to create a three-dimensional triangulation – a tetrahedralization. As mentioned in section II.A, many algorithms have already been developed for this purpose. A new algorithm was developed both in order to try to make one that is faster than existing solutions for this special case and to see if the tetrahedralization algorithm could also be used for shape interpolation. Another goal is to test whether such an algorithm always generates a result for the special case studied here. In the general case one cannot always create a constrained Delaunay tetrahedralization without inserting extra points [13].

This paper will not consider temporal uncertainty in the snapshots. [18] describes some ways of eliminating that uncertainty from the data model by converting it into spatial uncertainty, and this paper assumes that such a method has been used prior to the tetrahedralization.

The basic idea of the algorithm is to use the Delaunay triangulations of the two snapshots as a basis for creating the 3D tetrahedralization. Thus the first step is to construct a legal constrained Delaunay triangulation of each snapshot using the algorithm from [19]. After this the algorithm has two steps:

A. *Creating Tetrahedrons from Triangles*

After the triangulation is created, each triangle in one snapshot is matched to a point in the second, forming the initial tetrahedralization. This matching process is done in three passes. In the first pass each triangle is matched to the closest vertex in the other snapshot. Then the algorithm checks whether it overlaps any of the previously generated tetrahedrons. If it does, the tetrahedron is discarded and the algorithm moves on to the next triangle and tries to match that. This process is illustrated for a single triangle in Figure 2. In Figure 2, the core of the region is black and the support is gray-shaded.

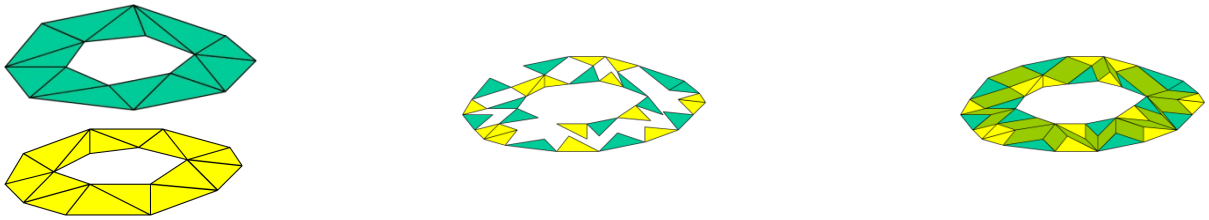
In the second pass, the algorithm tries to construct legal (non-overlapping) tetrahedrons for the triangles that were discarded in step 1 due to the tetrahedron overlapping another tetrahedron. It constructs the tetrahedrons using other points from the other snapshot, starting with the next closest. The current implementation of the application tries the ten nearest neighbors, testing for overlap until it finds one that does not overlap or it has used up the ten neighbors. The number ten was chosen to test enough points without testing all the points. A version testing all the points did not work noticeably better.

In the third pass, the algorithm creates a tetrahedron for each triangle remaining after the second pass by matching it to the nearest neighbor (the same tetrahedron that was tried in the first pass). Then a list of overlapping tetrahedra is created. The algorithm then tries to create new tetrahedral from the triangles that were used to create the tetrahedra in the overlap list. It created these new tetrahedra using the same method as in the second pass.

This method almost always manages to find a correct match for all the triangles. In the few remaining cases, moving a series of matches manually has always produced a good result. A Schönhardt polyhedron or other untetrahedralizable shape has not occurred in any of the tests of the program. Additionally, the algorithm has a random element (because the triangulation algorithm used is randomized and creates different triangulations each time it is run. Thus the triangles arrive in different orders to the tetrahedralizer). This means that even if the algorithm failed once it may succeed next time for the same time slice of the same object.

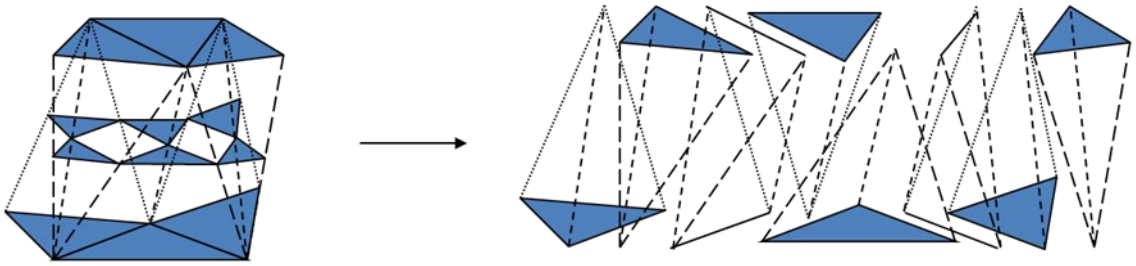
B. *Filling in the Gaps*

The tetrahedralization generated in Section III.A will be good at the time of the snapshots but will have large holes in between. In Figure 3a and b, this initial tetrahedralization of the two snapshots from Figure 2 is used to generate a snapshot halfway between the two initial snapshots. Figure 4a shows the tetrahedral as well as the in-between state. The holes in the tetrahedralization are readily apparent. However, these holes are themselves tetrahedra or a set of tetrahedra. None of the tests of the algorithm have turned up a case where this was not true, but the author has no mathematical proof that this is always the case. The basic idea for filling the gaps is to



a) Snapshots b) In-between state, only initial tetrahedra c) After gap filling

Figure 3: Example of the gap-filling algorithm



a) In-between value without filling triangles. b) The individual tetrahedrons in the final representation

Figure 4: Filling the gaps

check the edges of the tetrahedra. Each tetrahedron is bounded by four triangular surfaces. Those surfaces that go between snapshots should have a tetrahedron on each side. For each surface that goes between snapshots yet has a tetrahedron on only one side, a bordering tetrahedron should be inserted.

To find this bordering tetrahedron, the triangle needs to be matched to a fourth point. To find an appropriate fourth point, use the following algorithm:

- Two neighboring triangles are either matched to the same point or to different points. If they are matched to the same point no gap develops and nothing more needs to be done. If they are matched to different points a gap develops. To fill this gap, a tetrahedron consisting of the line that the gap opens in and the two points that the triangles are matched to is constructed.
- If this tetrahedron does not overlap any existing tetrahedra, it is inserted into the structure
- If it overlaps existing tetrahedra, do for each overlapping triangle (in the border of the overlapping tetrahedra) that borders only one tetrahedron:
  - Generate a tetrahedron from this triangle. The fourth point is the point in the original line that opened into the gap that the triangle does not already contain.

Figure 3c shows the in-between state of Figure 3b after gap filling. Figure 4b shows the tetrahedra that are generated based on Figure 4a.

This tetrahedralization can be used to compute alpha-cuts in the same way as a triangulation can in the non-temporal case. Figure 5 shows an example tetrahedralization with alpha-cuts 0.1, 0.25, 0.5 and 0.75 displayed at times 0, 0.5 and 1. Time 0 is the time of the first snapshot and time 1 is the time of the second snapshot. The core is displayed in a darker color than the support. In Figure 5b, the tetrahedrons filling the gaps are shown in a slightly lighter color than those originating as triangles in either snapshot. An example of using the interpolation algorithm on a real example is on the web on the following URL, but is not included in the paper due to its large size:

http://www.ux.uis.no/~tossebro/papers/Extra_figures_temporal_triangulation_paper.pdf

This example shows both two versions of the iso-lines and a raster. The two versions are without and with compensation for the jagged interpolation problem described in section VI. The raster was generated by asking the tetrahedralization for the fuzzy set value at each cell in the raster, where dark blue is 1.0 and white is 0.0.

C. Further Considerations

In most cases this algorithm produces a usable tetrahedralization. However, there are some cases in which the tetrahedralization becomes problematic. In some cases the closest point for a triangle in the support of one snapshot is in fact inside the core or outside the object in the other snapshot. This case produces strange interpolation artifacts and filler tetrahedrons with a constant function value. This case is eliminated by requiring that triangles are matched to points in the same region of the object. A triangle in the support (Where the average of the function values of the three corners is between 0 and 1) must be matched to a point on the outer border, inner border, or inside the support.

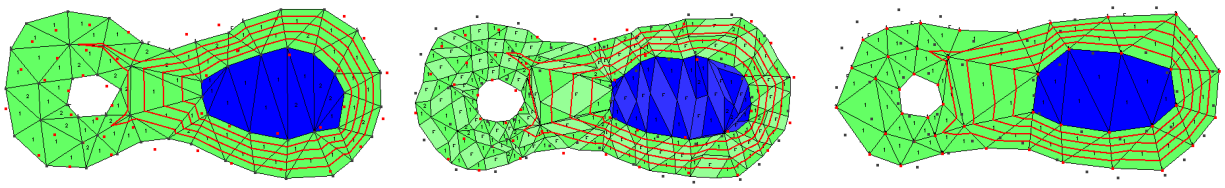
D. Running Time of the Algorithm

In this section the letter *n* is used for the number of points to be tetrahedralized and *t* is the number of triangles generated by the Delaunay triangulations that are used as input for the algorithm.

Construct Delaunay triangulations: O(n*log(n)) [19], [3]

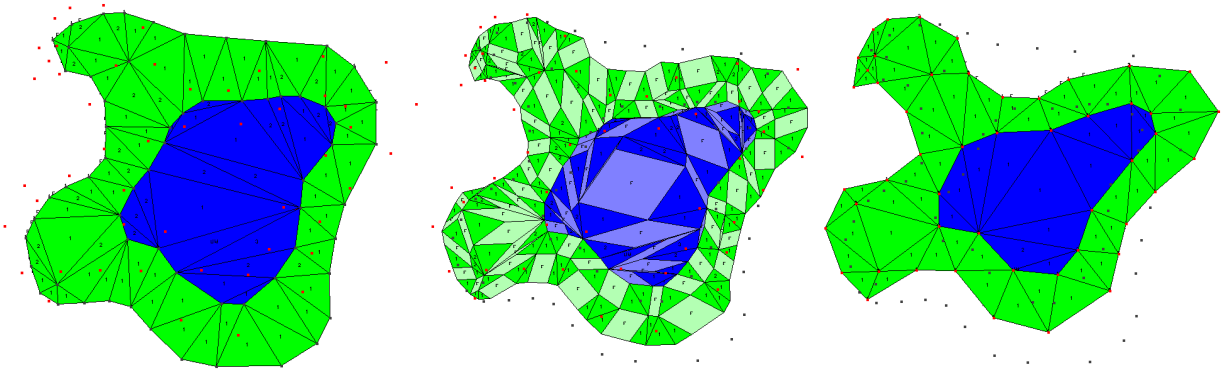
First pass:

- For each triangle (t, increases linearly with n in two dimensions according to Euler's formula for planar graphs):



a) Time 0 b) Time 0.5 c) Time 1.0

Figure 5: Example of tetrahedralization used for interpolation



a) Time 0 b) Time 0.5 c) Time 1.0

Figure 6: Tetrahedralization algorithm used for shape blending

- Find closest point in second triangulation: O(log(n)) using a spatial index
- Check resulting tetrahedron for overlap: O(log(t)) using a spatial index
- Insert new tetrahedron in spatial index: O(log(n))

- Total running time for all tetrahedrons: O(n*log(n))

Second pass:

- For each remaining triangle (this number probably increases linearly with n. Although a shape with more triangles is more complex, the probability of overlap should stay the same as the number of triangles in the neighborhood of the triangles, with which a tetrahedron potentially overlaps, stays the same.)
  - Find 10 closest points in other triangulation O(10 + log(n)). K-nearest neighbor has a running time of O(k + log(n)) using a spatial index according to [7]
  - Check tetrahedrons for overlap O(10*log(n))
  - Insert new tetrahedron in spatial index O(log(n))
- Total running time for all tetrahedrons: O(10*n*log(n))

Third pass:

- For each remaining triangle (probably increases linearly with n for the same reason as for the second pass)
  - Find closest point in second triangulation: O(log(n)) using a spatial index
  - Check resulting tetrahedron for overlap: O(log(t)) using a spatial index
  - For each overlapping tetrahedron: remove it and try to match the triangle as per 2nd pass. The number of overlapping tetrahedrons can theoretically be proportional to n but in most practical cases is a small constant. Thus the running time of this step is O(o*10*log(n)) where o is the number of overlapping tetrahedrons.
- Total running time for all tetrahedrons: O(n*log(n))

Filling in the gaps:

- Checking the triangles: O(n) as each triangle must be checked with 3 neighbors
- Create initial filling tetrahedron: O(1)
- Checking for overlap: O(log(n))
- Generating new tetrahedra: O(o), where o is the number of overlaps. o can potentially be large, but tests have shown that each triangle is overlapped by at most one such initial tetrahedron and most are overlapped by none, so the total o for all tetrahedra is less than n.

Total O(n*log(n)) as each initial filler needs to be checked for overlap. Checking for overlap is the step with the highest asymptotic running time.

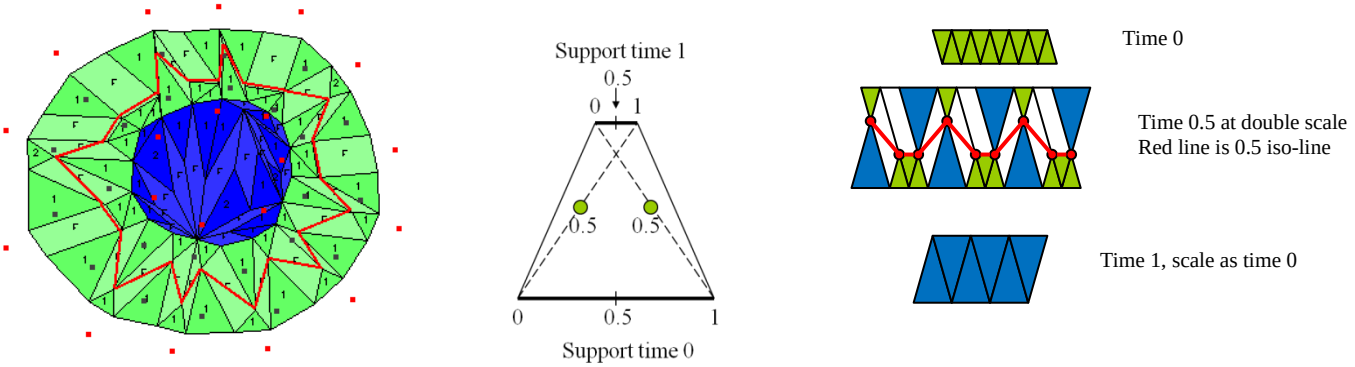
IV. Handling Changing Topology in Complex Regions

The algorithm described here also works on complex uncertain regions. A complex uncertain region may have several holes and may also have several disjoint core regions. While the method of tetrahedralization does not assume that the number of holes or core regions remains constant, there are still problems with such changes in topology. For instance, if a new core region appears between two snapshots, the individual triangles in the new core region may be matched to different points and therefore create a strange interpolation. This is solved by inserting an extra point in the triangulation in the other snapshot, giving that extra point a function value of 1.0, and matching all the triangles of the new core region to that point. The same can be done for holes. The algorithm is able to handle two core regions, support regions or holes joining into one with only minor interpolation artifacts.

V. Other Uses for the Algorithm

The tetrahedralization algorithm is also capable of interpolating the shape of the object between snapshots. For most of the cases that have been tested the interpolations that it creates are just as good as the interpolations that are created from the algorithm in [16]. Both algorithms create some interpolation artifacts but as long as the shapes consist of an adequate number of points (no very sharp angles) the interpolations are fairly equal in quality. One example of this is shown in Figure 6.

This algorithm can also be used to create a tetrahedralization for a spatiotemporal field that is updated at regular intervals but where the measurement points may move, as might be the case with a geo-sensor network with moving nodes such as a network measuring water



a) Example: iso-line 0.5 at time 0.5 b) Cause c) Cause

Figure 7: Jagged interpolation problem

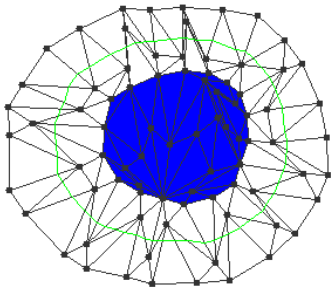


Figure 8: Triangulation from tetrahedralization for the example in Figure 7a

temperature at sea with nodes drifting along the ocean currents. The difference is that the field would cover the entire area of interest, and that there is no support and core boundaries. The last point only means that there are less limitations for the algorithm and the first point only means that it might need more time as the number of points to be triangulated may be much larger. Note that the algorithm allows the users to use different sample points for the different times of the spatiotemporal field but requires that the entire field is updated at the same times.

VI. Jagged Interpolation Problem

As long as the support of the object is roughly the same width in the two snapshots, interpolation based on tetrahedralizations works well. However, if it is used on an object where the support grows or shrinks significantly, a problem occurs as shown in Figure 7a. The support may grow or shrink because the measurements at different times may have different uncertainty. The effect shown in the figure is caused by the fact that the values are interpolated linearly along all the lines of the tetrahedrons. Figure 7b and c shows a schematic overview of the problem. The value is linearly interpolated along all the lines including the dashed ones. The marked points show where the value is 0.5 along each of them. While the 0.5 value should have been at the same place along both (the position in which they cross) it is in fact much further down.

A. *Possible Solution: Triangulation From Tetrahedralization*

Several strategies might be used to solve this problem. All of them have the problem that the iso-surfaces of probability in the 3D representation no longer become planar facets. The author has not been able to discover an algorithm that both creates planar facets for the iso-surfaces and does not suffer from the jagged interpolation problem.

One solution for the jagged interpolation problem might be to try to interpolate non-linearly along the dashed lines but then the question arises as to which function one should use. Additionally the normal formulas for interpolating in a tetrahedron assume that the interpolation is linear in all three dimensions. This means that a non-linear interpolation in one dimension might yield curved line segments even in single snapshots.

For this reason another approach was developed: Compute a triangulation from the tetrahedralization and use that to interpolate in two dimensions for the time instant requested. This algorithm, run on the shape from Figure 7a, is shown in Figure 8. The algorithm is as follows:

At any time instant t1 in the time slice in question do the following:

- For each tetrahedron create its representation at t1. This will be either a triangle or a parallelogram.
- For each triangle created in this fashion insert it into the new triangulation
- For each parallelogram created in this fashion split it into two triangles along the shortest diagonal.
- For each point in the new triangulation that is not already on the core or support boundary compute its value based on the ratio of the distance to the core and the distance to the outer edge of the support, as in [19].
- Use the resulting triangulation to create iso-lines of probability values in points.

This avoids the problem as the probability values of the vertices in the interior of the support are recalculated at each time instant. However, it is computationally somewhat more expensive and means that one cannot create and store the iso-surfaces for the entire time slice but must compute it individually for each time instant.

Running time:

- Finding the representation of each tetrahedron at t1: O(t) where t is the number of tetrahedrons
- Splitting the parallelograms: O(t) as roughly half the tetrahedrons become parallelograms.
- Computing new probability values: This requires finding the closest point along core and support for all the points in the interior of the support. O(t*log(t)) using a spatial index as the number of interior points is proportional to the number of tetrahedrons
- Creating iso-lines: O(t) as each tetrahedron results in either 1 (triangle) or 2 (parallelogram) triangles and each triangle needs to be examined once.

Total running time O(t*log(t)), which is worse than the O(t) running time for finding an iso-surface using the original algorithm.

VII. SUMMARY

In this paper, a way to create a tetrahedralization of an uncertain spatiotemporal object has been presented. This method is then used to interpolate fuzzy set or probability values. A problem with a naïve use of any tetrahedralization method for interpolation has been identified and a possible solution has been explored. The solution solves the problem but has the drawback that one cannot create iso-lines of probability once for the entire spatiotemporal object but must create them separately for each time instant.

REFERENCES

[1] V. T. de Almeida and R. H. Güting: Supporting uncertainty in moving objects in network databases. In the proceedings of the 13th ACM workshop on Geographical Information Systems (ACM-GIS), pages 31-40.

[2] Bajaj CL; Coyle EJ; Lin KN: Tetrahedral meshes from planar cross-sections. In Computer Methods In Applied Mechanics And Engineering, 1999, Volume: 179 Issue: 1-2 Pages: 31-52

[3] M. de Berg, M. van Krevald, M. Overmars and O. Schwarzkopf: Computational Geometry: Algorithms and Applications, 2nd edition. Springer-Verlag.

[4] L. P. Chew: Constrained Delaunay Triangulations. In Proc. 3rd Int. Symp. on Computational Geometry, 1987, pages 215-222.

[5] R. Cheng, D. V. Kalashnikov, S. Prabhakar: Querying Imprecise Data in Moving Object Environments. In IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 9, 2004, pages 1112-1127.

[6] L. Forlizzi, R. H. Güting, E. Nardelli and M. Schneider: A Data Model and Data Structures for Moving Objects Databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data (Dallas, Texas), pages 319-330, 2000.

[7] M. Kolahdouzan and C. Shahabi: Voronoi-Based K Nearest Neighbour Search for Spatial Network Databases. In Proceedings of the Thirtieth int. conf. on Very Large Databases, (VLDB04), pages 840-854, 2004.

[8] M.-J. Kraak, E. Verbree: Tetrahedrons and animated maps in 2D and 3D space. In Proc. 5th Int. Symposium on Spatial Data Handling (SDH), pages 63-71, 1992.

[9] D. Peuquet: Making Space for Time: Issues in Space-Time Data Representation. In GeoInformatica 5:1, pages 11-32, 2001.

[10] B. Plewe: The Nature of Uncertainty in Historical Geographic Information. In Transaction in GIS, vol. 6, issue 4, pages 431-456, 2002.

[11] D. Pfoser and N. Tryfona: Capturing Fuzziness and Uncertainty of Spatiotemporal Objects. In Proc. Advances in Databases and Information Systems, LNCS vol. 2151, 2001, pages 112-126, Springer Verlag.

[12] D. Pfoser, N. Tryfona, C. S. Jensen: Indeterminacy and Spatiotemporal Data: Basic Definitions and Case Study. In Geoinformatica, Vol. 9, No. 3, pages 211-236, 2005.

[13] J. R. Shewchuk: Sweep Algorithms for Constructing Higher-Dimensional Constrained Delaunay Triangulations. In Proc. of the 16th Annual Symp. on Computational Geometry (Hong Kong), pages 350-359, Association for Computing Machinery, June 2000

[14] J. R. Shewchuk: Updating and constructing Constrained Delaunay and Constrained Regular triangulations by Flips.

[15] Hang Si: Constrained Delaunay tetrahedral mesh generation and refinement. In Finite Elements in Analysis and Design no. 46(2010), pages 33-46

[16] E. Tøssebro and R. H. Güting: Creating Representations for Continuously Moving Regions from Observations. In Proc. 7th Int. Symp. on Spatial and Temporal Databases, pages 321- 344, July 2001.

[17] E. Tøssebro and M. Nygård: An Advanced Discrete Model for Uncertain Spatial Data. In Proc. 3rd Int. Conf. on Web-Age Information Management (WAIM), pages 37-51, August 2002.

[18] E. Tøssebro and M. Nygård: Extending Discrete Models for Uncertain Spatial Data to Spatiotemporal Data. Published in Proc. 2nd IASTED International Conference on Information and Knowledge Sharing, pages 57-68.

[19] E. Tøssebro and M. Nygård: Computing the Probabilities of Operations in Vector Models for Uncertain Spatial Data. Published in Proc. IEEE int. conference on Signal-Image Technologies and Internet-Based Systems (SITIS), pages 78- 85, 2008.

[20] L. A. Zadeh: Fuzzy sets. In Information and Control, Vol. 8, 1965, pp. 338-353.