INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI
800-521-0600
STA: SPATIO-TEMPORAL AGGREGATION OF PHYSICAL FIELDS WITH APPLICATIONS TO ANALYSIS OF DIFFUSION-REACTION PHENOMENA

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Iván Ordóñez, B.Sc., M.Sc.

The Ohio State University

1999

Dissertation Committee:

Feng Zhao, Adviser
Balakrishnan Chandrasekaran
Richard Parent

Approved by

[Signature]
Adviser
Department of Computer and Information Sciences
© Copyright by

Iván Ordoñez

1999
ABSTRACT

Spatio-temporal data sets arise when time-varying physical fields are discretized for simulation or analysis. Examples of time-varying fields are isothermal regions in the sea, or pattern formations in natural systems, such as convection rolls or diffusion-reaction systems. The analysis of these data sets is essential to generate qualitative interpretations for human understanding. However, phenomena such as diffusion-reaction can exhibit extremely complex behaviors that are time-dependent, spatially interacting, and sensitive to system parameter variations. It is often very difficult to predict such behaviors through analytical means alone. Recent advances in computational methods and hardware have led to increasing interest in automated means for generating and classifying behaviors of such systems.

This thesis develops Spatio-temporal Aggregation (STA), a temporal extension to Spatial Aggregation, for recognizing and tracking structures in spatio-temporal data sets. Specifically, novel data structures and operations to explicitly represent temporal changes are introduced. The STA algorithms record and maintain temporal events and compile event sequences into concise history descriptions to infer possible causal relations in the data. Such a compilation is carried out at several levels of description, from the bottom up: first, low level events are identified and tracked, and then, a subset of those events, relevant at the next description level, is identified.
The process is iterated until a high level narrative of the system's temporal evolution is obtained.

STA has been demonstrated on a class of diffusion-reaction systems in two dimensions and has successfully generated high-level symbolic descriptions of systems similar to those produced by scientists through careful hand-tuned computational experiments. The iterative behavior generation process first represents the data objects as particles at the lowest level, forms coherent objects described by uniform regions, and then extracts prototypical object evolution and interaction to generate a qualitative description of the system's behavior. The tracking of changes in object shape and topology gives rise to an efficient algorithm for identifying temporal behaviors of objects in the field. Finally, by comparing qualitative descriptions for different parameterizations, STA classifies systems into equivalence classes, each exhibiting qualitatively similar behaviors.
To Nancy
ACKNOWLEDGMENTS

Carrying out the research in this thesis would have been impossible without the constant support of my wife Nancy, who always held me straight, specially in my most stressful moments. She was always there to help me focus and keep my spirits high.

My advisor, Feng, was always a source of inspiration. His rigid honesty, his insightful comments and his unbridled enthusiasm always pushed me forward, specially at times when I was stuck.

My parents always let me know that I had their love, no matter what, and were a bright light that allowed me to go on in my moments of loneliness. My brothers, Gonzalo and Raul, were also always there to give me advice, support and to make me smile when I needed it most.

My good friend Chris Bailey-Kellogg was a key figure in the development of my research: not only did he develop the Spatial Aggregation Language, which this thesis extends, but he also provided many substantive comments that were crucial to this project.

My other committee members, Rick and Chandra, helped make a challenging yet pleasant experience out of my thesis defense. Also, their comments helped improve my work.
Xingang Huang, my lone meeting partner for over a year, was also a constant source of intelligent comments. His ability to see as obvious what was sometimes beyond my immediate reach helped me make many needed leaps in imagination.

I am also indebted to other previous members of the INSIGHT group: Jeff May, Shion Loh, and the late Sing Pai Goh (may his tragic, untimely demise be a constant reminder of the fragility of life to all of us). Our lively weekly meetings kept the intellect shining bright, and they will always be a fond memory for me.

This work was conducted as part of the research in the Intelligent Simulation Group at Ohio State. Support for the Group is provided in part by ONR Young Investigator award N00014-97-1-0599, NSF Young Investigator award CCR-9157802, a Sloan Research Fellowship, NSF grant CCR-9308639, and a Xerox Foundation grant.
VITA

March 29, 1965 .......................... Born - Guayaquil, Ecuador

                                      Escuela Politecnica Nacional, Quito, Ecuador

1990-1991 ............................... Graduate Research Assistant.
                                      Center for Artificial Intelligence,
                                      Instituto Tecnologico y de Estudios Superiores de Monterrey

                                      Instituto Tecnologico y de Estudios Superiores de Monterrey

1992 ..................................... Research Associate.
                                      Center for Artificial Intelligence,
                                      Instituto Tecnologico y de Estudios Superiores de Monterrey

1996-1998 ............................... Graduate Research Associate.
                                      Intelligent Simulation Group,
                                      The Ohio State University

PUBLICATIONS

Research Publications


FIELDS OF STUDY

Major Field: Computer and Information Science
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapters:</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.  Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1  Motivation and Challenges</td>
<td>4</td>
</tr>
<tr>
<td>1.1.1 Analytical Treatment and its Limitations</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 A Case Study: Diffusion-Reaction Systems</td>
<td>1</td>
</tr>
<tr>
<td>1.1.3 Challenges</td>
<td>10</td>
</tr>
<tr>
<td>1.1.4 A Sample Session: Classifying Patterns according to Behavior</td>
<td>12</td>
</tr>
<tr>
<td>1.2  Contributions</td>
<td>14</td>
</tr>
<tr>
<td>1.2.1 Spatio-temporal Aggregation (STA)</td>
<td>14</td>
</tr>
<tr>
<td>1.2.2 Managing Temporal Changes through KDS</td>
<td>16</td>
</tr>
<tr>
<td>1.2.3 Application: Classification of Behaviors in Diffusion-Reaction</td>
<td>17</td>
</tr>
<tr>
<td>1.3  Outline</td>
<td>18</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xiv</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xii</td>
</tr>
<tr>
<td>Vita</td>
<td>vii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Dedication</td>
<td>iv</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
</tbody>
</table>
2. Background and Related Work ........................................ 19
   2.1 Spatial Aggregation ........................................... 19
   2.2 Spatio-Temporal Reasoning ................................... 20
   2.3 Computer Vision and Object Recognition .................... 21
   2.4 Tracking Coherent Objects in Massive Data Sets ........... 22
   2.5 Diffusion-Reaction Fields .................................... 23
   2.6 Kinetic Data Structures ....................................... 23
   2.7 Pattern Recognition and Shape Classification .............. 24

3. STA: Spatio-Temporal Aggregation ................................ 26
   3.1 Spatial Aggregation ........................................... 26
       3.1.1 A Uniform Vocabulary ................................... 28
   3.2 Temporal Changes ............................................ 28
       3.2.1 Time and Identity ....................................... 30
       3.2.2 Aggregation and Persistence ............................ 34
       3.2.3 Change as Detection of Structural Failure ............ 38
       3.2.4 Abstracting Change in a Sampling Particle System .... 44
       3.2.5 Tracking Change in Time ................................ 54
       3.2.6 Temporal Aggregation: Reasoning About Change ....... 56
   3.3 STA Library Extensions ....................................... 58
       3.3.1 Spatial Change ........................................... 59
       3.3.2 Neighborhood Structure Change ........................ 59
       3.3.3 Change Operators for Spatial Partitions .............. 60
       3.3.4 Change Operators for Object Abstraction ............. 61
   3.4 An Example: Tracking Wave Packets in One Dimension ...... 63
       3.4.1 An Aggregation Layer ................................... 65
       3.4.2 The Iteration Step ...................................... 67

4. Application to Diffusion-Reaction Systems ....................... 71
   4.1 The Problem, Goals and Methods ............................ 71
       4.1.1 Overview of the Algorithm ............................. 75
   4.2 Behavior Description ......................................... 75
       4.2.1 Tracking High-Level Structures ....................... 75
       4.2.2 Shape detection and classification ..................... 106
       4.2.3 Putting it All Together: Extracting Behavioral Descriptions 123
       4.2.4 Results ................................................ 129
   4.3 Behavior Classification ....................................... 138
       4.3.1 Generalized Histories .................................. 138
5. Conclusions and Future Work ........................................ 149
   5.1 Thesis Summary .................................................. 149
   5.2 Applicability ...................................................... 150
   5.3 Future Work ...................................................... 152

Bibliography .............................................................. 156
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 A sample history</td>
<td>12</td>
</tr>
<tr>
<td>3.1 A procedure to determine the extent of influence of a point insertion in a Delaunay triangulation</td>
<td>51</td>
</tr>
<tr>
<td>3.2 Change Operators in Spatial Objects</td>
<td>60</td>
</tr>
<tr>
<td>3.3 Change Operators in Neighborhood Graphs</td>
<td>60</td>
</tr>
<tr>
<td>3.4 Change Operators in Classifiers</td>
<td>62</td>
</tr>
<tr>
<td>3.5 Change Operators in Abstractors</td>
<td>63</td>
</tr>
<tr>
<td>3.6 Aggregation layer for the longitudinal wave example</td>
<td>66</td>
</tr>
<tr>
<td>3.7 Iteration step for the longitudinal wave example</td>
<td>70</td>
</tr>
<tr>
<td>3.8 Collision detection algorithm</td>
<td>71</td>
</tr>
<tr>
<td>3.9 Separation detection algorithm</td>
<td>72</td>
</tr>
<tr>
<td>1.1 An algorithm for sampling a field through particles</td>
<td>84</td>
</tr>
<tr>
<td>1.2 A procedure to re-describe a set of adjacent triangles as a polygonal body</td>
<td>99</td>
</tr>
<tr>
<td>1.3 Events in a History</td>
<td>104</td>
</tr>
<tr>
<td>1.4 Multiple Curvature Segmentation Algorithm</td>
<td>111</td>
</tr>
</tbody>
</table>
1.5 An automatically generated history for a Lotka-Volterra system (part 1132)

1.6 An automatically generated history for a Lotka-Volterra system (part 1133)

1.7 Output of history aggregation program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 145
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>6</td>
</tr>
<tr>
<td>1.2</td>
<td>8</td>
</tr>
<tr>
<td>1.3</td>
<td>9</td>
</tr>
<tr>
<td>1.4</td>
<td>12</td>
</tr>
<tr>
<td>1.5</td>
<td>13</td>
</tr>
<tr>
<td>1.6</td>
<td>14</td>
</tr>
<tr>
<td>2.1</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>31</td>
</tr>
<tr>
<td>2.3</td>
<td>36</td>
</tr>
<tr>
<td>2.4</td>
<td>39</td>
</tr>
<tr>
<td>2.5</td>
<td>43</td>
</tr>
<tr>
<td>2.6</td>
<td>46</td>
</tr>
<tr>
<td>2.7</td>
<td>47</td>
</tr>
<tr>
<td>2.8</td>
<td>49</td>
</tr>
<tr>
<td>2.9</td>
<td>51</td>
</tr>
</tbody>
</table>
1.18 Field values above threshold for Lotka-Volterra ........................................ 98
1.19 Field values above threshold for a Brusselsator ............................................ 98
1.20 A lission transformation ...................................................................................... 103
1.21 Curve smoothing through curvature tuning ........................................................ 109
1.22 Curvature stress for different target curvature values ......................................... 110
1.23 Multi-curvature description ................................................................................ 112
1.24 Differing multi-curvature segmentations due to noise ....................................... 116
1.25 A set of polygonal objects to be classified ......................................................... 121
1.26 Shape clusters ..................................................................................................... 122
1.27 Shape cluster representatives ............................................................................... 123
1.28 A three-dimensional representation of a still diffusion-reaction field ............... 127
1.29 Time to update constructs on a field versus noise level .................................... 128
1.30 Fraction of particle movements causing structural change versus noise .......... 130
1.31 A short run for a Lotka-Volterra System ............................................................ 131
1.32 Representative shapes for nine clusters .............................................................. 131
1.33 Cluster 1 ............................................................................................................. 134
1.34 Cluster 2 ............................................................................................................. 134
1.35 Cluster 3 ............................................................................................................. 134
1.36 Cluster 4 ............................................................................................................. 135
1.37 Cluster 5 ............................................................................................................. 135
1.1 Motivation and Challenges

Spatio-temporal data sets typically arise when time-varying physical fields are discretized for the purpose of simulation or analysis. Some examples of the time-varying fields are turbulent fluids, isothermal regions in the sea, or pattern formations in natural systems, such as convection rolls or diffusion-reaction systems. The analysis of these data sets is essential in scientific visualization, modeling, or generating qualitative interpretations. However, many time-varying physical fields such as the diffusion-reaction phenomena can exhibit extremely complex behaviors that are time-dependent, spatially interacting, and sensitive to system parameter variations. It is often difficult, if not impossible, to predict such behaviors through analytical means alone. Because of recent advances in computational methods and hardware, there has been increasing interest in automated means for generating and classifying behaviors of such systems. In particular, the Spatial Aggregation (SA) approach [41] provides a framework for the identification of structures in spatially-distributed fields.

Regions of uniformity arise in a physical field because of continuities of properties such as intensity, temperature or pressure. A human observer would have little
trouble describing events such as the formation of convective rolls in boiling water in straightforward qualitative terms. Furthermore, such an observer would also easily recognize other phenomena also exhibiting convective rolls as belonging to the same class, even if they differ in details such as the size or the number of rolls.

A qualitative description of a physical field recognizes several events: the existence of coherent objects (that is, objects that are internally connected, of uniform features, and with a well-defined border), their persistence through time, and their abrupt change. The study of such high-level events arises frequently in many disciplines of scientific inquiry that deal with complex systems; for example, in interventional sciences such as medicine, it may be the high-level descriptions that provide the key to a problem: the cells of a heart in a state of atrial fibrillation often do not behave differently from the cells of a normal heart at the individual level; it is their aggregated behavior that has gone awry [1]. Inside the cell, activities such as the formation of organelles and the coordination of its various parts for global tasks such as mitosis, in the absence of a central-regulating agent, can only be understood in terms of aggregated behavior. Similarly, multiple cells in a developing organism self-organize to form highly specialized organs at particular locations, through a process known as **morphogenesis**. In a biological system that has failed, the more complexity we are aware of, the more numerous are the sites where intervention can take place [6].

Any attempt to study complex phenomena of this kind would benefit greatly from the automatic generation of high-level descriptions from raw data, as well as from the classification of such events based on topological and geometric characteristics of the involved objects and the nature of the transformations they undergo.
This thesis develops Spatio-Temporal Aggregation (STA), consisting of a temporal extension to Spatial Aggregation. This extension addresses systems that vary over time by recognizing and tracking structures in spatio-temporal data sets. STA is applied to a class of diffusion-reaction systems in two dimensions and it successfully generates high-level symbolic descriptions about the systems. In addition, by comparing multiple narratives, STA classifies systems with different parameterizations into equivalence classes each of which exhibits qualitatively similar behaviors.

The importance of the study of high-level phenomena in diffusion-reaction systems is well established. For instance, the Belousov-Zhabotinskii (BZ) reactor has been found to behave in a similar way to the heart muscle [20]. This reactor has also been shown to have computing abilities, as it was harnessed to solve the problem of finding the quickest route through a maze. High-level structures of the kind whose behavior we will describe in this thesis were shown to ripple through paths of minimal length [33].

The target audience of this thesis is constituted by scientists and researchers who are interested in the qualitative description and classification of complex, spatially distributed systems that vary in time. This research is aimed at relieving the need to use human mediation and subjective criteria. By introduced standard, well defined tools of description and classification, these scientists benefit not only from the elimination of tedious work, but also from an increase in reliability: human standards vary widely when dealing with something as complex as classifying objects and events. When the possibility of human error is removed, the scientific community stands to gain much from the existence of a standardized, computer-aided methodology of classification.
1.1.1 Analytical Treatment and its Limitations

Many nonlinear dynamical systems do not lend themselves to analytical treatment because of the lack of closed form solutions. Worse, chaotic behaviors in some systems make long-term prediction impossible. In many cases, even getting a crude idea of how the system behaves is impossible without the use of a numerical simulation. In particular, parabolic partial differential equations such as the diffusion equation or the time-dependent Schrödinger equations have solutions that are extremely sensitive to such variables as the initial conditions and the geometry of the domain. Certain standard tools from bifurcation theory such as amplitude equations have been successful in providing approximate analytic solutions in certain special cases—namely spatially uniform states that are near the transition from linear stability to linear instability [15]. The general case remains elusive, however.

1.1.2 A Case Study: Diffusion-Reaction Systems

A typical instance of time-varying fields is the set of phenomena known as Diffusion-Reaction. These phenomena are of great scientific importance, because they are associated with the problem of morphogenesis, first addressed by Turing [35]. Particularly interesting instances, where noticeable patterns emerge and vary in seemingly unpredictable ways, will be examined.

Diffusion-reaction systems model the dispersive behavior of concentrations of chemicals. Let \( c(x, t) \) be the density function (that is, concentration per unit of volume) of a substance in a certain medium, where \( x \) is a vector in the domain space and \( t \) is time; let \( Q(x, t) \) be the rate of creation of the substance, i.e. the rate at which its concentration increases, and let \( D \) be a constant known as the diffusion coefficient.
Through application of the divergence theorem and Fick's law, we obtain the general Diffusion-Reaction equation (see [12] for a full derivation):

$$\frac{\partial c}{\partial t} = D \nabla^2 c(x, t) + Q(x, t). \quad (1.1)$$

We are interested in cases where there are two reactants which diffuse independently but whose reaction components are coupled together. In such cases, the general model is given by

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + f_u(u, v) \quad (1.2a)$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + f_v(u, v) \quad (1.2b)$$

where $u$ and $v$ are the concentrations of the two reactants and $D_u$ and $D_v$ are their diffusion rates.

**The Gray-Scott System: A Model of Glycolysis**

The phenomenon of glycolysis is found in virtually all living organisms. It probably evolved before the existence of an oxygen-rich atmosphere, because it allows the extraction of energy from sugar in the absence of free oxygen. It is shown below that the Gray-Scott model of glycolysis is a diffusion-reaction system.

Glycolysis is an autocatalytic process, in which two ATP molecules are consumed to produce four ATP molecules. A model of glycolysis introduced by Gray and Scott was first studied from a computational perspective by Pearson [21]. It is a simplified model, which abstracts away the details of this complex phenomenon, and considers two reactants, $U$ and $V$, which interact in an irreversible autocatalytic process: $U + 2V \rightarrow 3V$. Because the reaction speeds are proportional to the molarity of the reactants, it follows that $U$ is lost at a rate proportional to $uv^2$, while $V$ is gained...
proportionally to the same rate. To compensate for the loss, \( U \) is fed continuously into the system at a rate \( F \). On the other hand, \( V \) is assumed to be consumed by some other process at a constant rate \( k \). Finally, both \( U \) and \( V \) are removed by the feed process, so we are left with the following equations:

\[
\begin{align*}
\frac{\partial u}{\partial t} &= D_u \nabla^2 u - ur^2 + F(1 - u) \\
\frac{\partial r}{\partial t} &= D_r \nabla^2 r + ur^2 - (F - kr). 
\end{align*}
\tag{1.3}
\]

This model is of interest not only as a model of glycolysis, but also because it exhibits a variety of behaviors unlike anything observed before in theoretical or numerical studies.

In Figure 1.1 three successive snapshots for a Gray-Scott system are shown. Colors code the concentration of the substances: red indicates high concentration of substance \( U \) and low concentration of substance \( V \), whereas blue indicates the opposite. It should be pointed out that different combinations of the four parameters \( D_u, D_r, F \) and \( k \) produce very different behaviors in the formation of patterns for
this systems. These behaviors range from the formation of quasi-stable hexagonal patterns to fast-changing bodies in perpetual chaotic motion.

Pearson has argued that since glycolysis occurs inside the cell, it is possible that patterns such as the above could form within it. Furthermore, he observes that the process of mitosis, by which cells divide, requires the formation of a bipolar structure known as the mitotic spindle, which is likely governed by simple physical processes such as chemical reactions and diffusion, rather than by complex genetic mechanisms [23].

Hence it seems plausible that, as Turing suggested, there is a strong link between the qualitative behavior of Diffusion-Reaction systems and the formation of structures in living organisms.

The Lotka-Volterra Equations: Predator/Prey Systems

Another diffusion-reaction system we will consider is the distributed Lotka-Volterra model. This model of predator-prey systems was the first to be well understood from a mathematical point of view. It consists of a prey, or producer, and a predator, or consumer. The prey grows at a rate proportional to its population, and is consumed by the predator at a rate proportional to the populations of both species. The predator is destroyed proportionally to its population (by natural causes, one could presume) and grows proportionally to the population of both species (as it feeds). Therefore the stationary point differential equations for the Lotka-Volterra system are:

$$\frac{du}{dt} = Au - Bur$$
$$\frac{dv}{dt} = -Cv + Dur.$$  \hspace{1cm} (1.1)
Figure 1.2: Three successive snapshots of a pattern in the Lotka model of Equations 1.1 for values $D_v = 2 \cdot 10^{-7}$, $D_p = 1 \cdot 10^{-7}$, $A = 0.001$, $B = 0.0021$, $C' = 0.001$, $D = 6 \cdot 10^{-4}$.

where $u$ is the prey population and $v$ is the predator population.

Typical functions described by the Lotka-Volterra model are periodic, out of phase population variations. When the prey population is low, the predators start dying off; as a result, predation decreases and the prey can multiply. Eventually a point is reached when there are enough prey to feed the few remaining predators, and their population grows until they begin depleting the prey population; and so on.

This simple model becomes more interesting when the concept of spatial location is introduced. It is assumed that the species propagate in space, following the diffusion equations. Then, the Lotka-Volterra equations become the reaction component of a diffusion-reaction system, as seen in the following equations. The result constitutes a very simple model of spatially distributed inter-species population dynamics.

\[
\frac{\partial u}{\partial t} = D_u \nabla^2 u + Au - Buv \\
\frac{\partial v}{\partial t} = D_v \nabla^2 v - C' v + Du v.
\]  

(1.5)
Figure 1.2 shows three snapshots of a Lotka-Volterra distributed system. The oscillatory behavior for punctual systems is also seen in space as slow moving waves.

The Brusselator: Traveling Chemical Waves

The Brusselator is another model of a chemical reaction, introduced by Progogine and Lefever [26]. It involves six chemical compounds that interact non-linearly, and can be modeled in a simplified manner by Equations 1.6, where $a$, $b$, $\gamma$ and $\omega$ are system parameters.

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + u^2 v + u - (b + 1) u - \gamma \sin(\omega t)$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v - u^2 v + bu. \quad (1.6)$$

where $u$ and $v$ represent concentrations of two salient compounds (other quantities are not used in this simplified model).
The interest of the Brusselator is that it was one of the first chemical systems to show self-organization. The sinusoidal component of the model forces it to stay away from equilibrium, and causes waves of uniform concentration to spread in space.

Figure 1.3 shows three snapshots of a Brusselator. Large waves propagate slowly through the field.

1.1.3 Challenges

As seen above, some dynamical systems with very succinct mathematical models often exhibit complex behavior. The simulation or observation of such systems result in massive data sets. The problem of extracting high-level descriptions of such data sets for the purpose of classification and control has been tackled in a number of ways. In particular, Spatial Aggregation does so by introducing a simple hierarchy of conceptual operations. However, existing applications of SA, which abstract over such domains as phase spaces and configuration spaces, or physical fields in a fixed steady state, do not require to deal with the concept of temporal change. This is rather limited, in the sense that many interesting problems in fields such as computer graphics, scientific visualization and other areas involve dynamic transformation in scalar or vector fields.

We are interested in extending SA to include reasoning about time at multiple levels of abstraction. For this we need to address the dynamic update of objects while introducing minimal overhead in maintaining the spatial data structures. At first sight, it would seem that this problem is far from tractable: in general, one could think that there is nothing we could say about the relationship between data
sets across time, and that therefore the only possible general approach is the brute-force, or compute-everything-at-each-time-step approach. However, having certain minimal knowledge about the nature of the systems allows to reduce update costs substantially, in the measure that it is possible to re-use pre-computed structures with only relevant modifications.

There already exists a body of research focused on tracking features of distributed data sets through time [36, 29, 28, 40]. All these existing techniques could be used to track high-level events in diffusion-reaction systems. In particular, an algorithm developed by Yip [40] uses Spatial Aggregation as a framework to track coherent objects in data sets obtained from numerical simulations of turbulence. The research in this thesis builds on Yip's work to develop a general mechanism for reasoning about temporal phenomena in spatial domains. This thesis is different from the previously mentioned work in that it emphasizes more the formal aspects of change as a set of abstract operations, and in that it produces quantitative metrics of difference between system behaviors.

Furthermore, the case study itself, pattern-forming diffusion-reaction systems, presents interesting challenges on its own. Although bifurcation theory can, in some cases, shed some light on why small parameter variations can cause drastic changes in behavior, mathematical theory by itself has not yet been able to anticipate the surprising variety of irregular spatiotemporal patterns that arise in some systems. Without a method that automatically differentiates qualitatively different behaviors, telling these irregular spatio-temporal patterns apart requires nothing less than the eye of a human observer. The challenge is to make the observer unnecessary.
At step 88 body 3 was born
At step 88 body 2 was born
At step 88 body 1 was born
At step 88 body 0 was born
At step 229 bodies 0 (born 88), 3 (born 88) fused into body 1
At step 237 body 2 (born 88) fused into body 1

Table 1.1: An automatically generated history for the diffusion-reaction system depicted in Figure 1.1.

1.1.4 A Sample Session: Classifying Patterns according to Behavior

To acquaint the reader with the tasks that the program developed in this thesis can carry out, we document a short run.

The program narrates the events that take place in an evolving diffusion-reaction field. For instance, when a system such as that shown in Figure 1.1 evolves, the program can generate a history file such as that of Table 1.1.

The program is also capable of finding when objects change their shape; for the sake of simplicity that feature is turned off for this example.
Figure 1.5: Snapshots for system evolutions on a diffusion-reaction system. Histories were later classified into groups of similarity. Each snapshot shows a single frame chosen roughly from the middle of each run. Histories were recorded from generations 4000 through 5000, approximately.

The program can also compare several histories and group them into classes of similar behavior. For example, consider the systems pictured in Figure 1.5.

The following five groups were discovered:

- Cluster 1: History (a1).
- Cluster 2: Histories (e1) and (g).
- Cluster 3: History (b1).
- Cluster 4: Histories (c1), (d) and (f).
- Cluster 5: History (a1).
I'm not sure what patterns you're referring to. Some patterns discovered by Pearson. The algorithm developed in this thesis classifies diffusion-reaction systems into qualitatively distinct classes, similar to those found by Pearson. Reproduced from [21].

Notice how the above groups compare with similar classes discovered by Pearson [21], shown in Figure 1.6: cluster 1 corresponds to pattern (b); cluster 2 to (c); and cluster 5 to (a).

In the subsequent chapters all the details pertaining to this program will be explained.

1.2 Contributions

This thesis makes contributions in three areas: Introduction of spatio-temporal aggregation mechanism for distributed time-varying systems, and its implementation as extensions to the Spatial Aggregation Library; a novel use of Kinetic Data Structures as a tool to determine relevance of change; and an application of this mechanism to the classification of behaviors in a set of diffusion-reaction systems.

1.2.1 Spatio-temporal Aggregation (STA)

Spatial Aggregation is extended, both conceptually, and in an actual implementation, as C++ extensions to the Spatial Aggregation Library. Conceptually, this
implies the introduction of various abstract operations to represent the notions of persistence and change. Common qualitative events such as birth, death, collision, separation, acquisition or loss of components or properties are identified for objects in spatio-temporal domains. The relations between objects at different levels of descriptions such as points and regions are established via aggregation.

**Change Operator Definitions**

New mechanisms to manage change in a SA aggregation hierarchy are developed. Object change operators start with simple existential changes (birth, death) of elementary objects (such as particles), changes in their positions, or changes in their feature values. Neighborhood graph change operators allow the specification of structural changes in neighborhood graphs aggregated from collections of objects, brought about by the changes in the objects. Classification change operators allow the detection of how changes in the objects and their neighborhood relations affect an existing partition of the space into classes: classes may collide, cease to exist, come into being, or alter their internal composition. Redescription change operators examine changes in classes and determine how those changes affect an existing redescription of classes as higher-level objects, carrying out the relevant changes in redescription.

**Implementations**

The new operations are implemented as C++ extensions to the Spatial Aggregation Library. These extensions consist both in new abstract methods in the interfaces (with care to maintain consistency), and in implementations of these methods for various data structures. We also present, mostly through example, a programming methodology for the use of the new operations.
1.2.2 Managing Temporal Changes through KDS

SA relies heavily on geometric data structures, because it depends on neighborhood graphs and simplicial complexes. Therefore, it is natural that any study of change in the context of SA needs to address what happens with such geometric data structures when their constituting elements have changed.

This question has been extensively addressed by the kinetic data structures approach (KDS) [3]. The KDS method is concerned with finding out under which conditions a geometric data structure (such as a minimum spanning tree or a convex hull) suffers a structural transformation given that its constituent objects are in motion. Because in SA a structural change in a neighborhood graph is likely to cause changes in the objects that are built on top of this graph, KDS provides an efficient implementation of change operators in STA.

It should be noticed, however, that KDS is not concerned with aggregation. By layering a hierarchy of constructs on top of a set of elementary objects, we generalize the KDS question as follows: under which conditions do changes at a particular aggregation level result in changes at higher levels of description? what is the nature of the relationship between changes at various levels? While answering these questions in the most general sense goes well beyond the scope of this thesis, they are considered and answered for particular cases arising in the case study of diffusion-reaction systems.
1.2.3 Application: Classification of Behaviors in Diffusion-Reaction Systems

STA has been applied to the classification of behavior for diffusion-reaction systems that exhibit coherent spatio-temporal patterns. The algorithm uses spatio-temporal abstraction to extract behaviors as qualitatively distinct temporal structures. This case demonstrates all major elements of typical STA applications: change in low level objects, structural change, abstraction of change, tracking of objects through various kinds of transformations.

Tracking Objects

Coherent objects in a diffusion-reaction field are tracked through time by sampling them with a particle system. The particles adjust themselves to sample with high detail those regions of the field where the gradient is high. As the sampling particle system changes with the field, the aggregated structures built on top of the particles change too.

Detecting Geometric Properties

Detecting catastrophic events such as collisions or separations is relatively straightforward. Detecting the subtler, and arguably more subjective notion of qualitative change in the shape of an object is considerably harder.

No qualitative description of change in objects could be complete without introducing the notions of shape and change in shape. Those notions are application-dependent. In the case of this thesis, we are interested in crude geometric features (corresponding to what could be called “overall shape” in two dimensions) rather than fine detail. Therefore, an algorithm to compare high-level shapes in two dimensions
that is insensitive to low-level detail, previously introduced in [9], is implemented and then further adapted for the purpose of shape classification.

**History Classification**

The domain of samples for the highest level of classification pursued in this thesis consists of all the instances of steady-state solutions for various parameter values. By observing the behaviors of such solutions and comparing them through each other, we divide the parameter space into regions of qualitatively similar behavior. More specifically, for each parameter set, a history is generated; after that process is completed, histories are compared with each other, and then they are classified into groups. Each group contains histories whose behaviors are considered to be qualitatively equivalent.

### 1.3 Outline

The remainder of this thesis is divided as follows: Chapter 2 provides context for this research by pointing out related work to Spatio-Temporal Aggregation and to Diffusion-Reaction Fields that form spatio-temporal patterns. Chapter 3 presents the detail of STA, including specifications of all the operators it adds to SA. Chapter 4 applies STA to the problem of describing and classifying behavior of diffusion-reaction systems, explaining in detail how the problem is approached and solved. Finally, Chapter 5 concludes and discusses future work.
CHAPTER 2

BACKGROUND AND RELATED WORK

This thesis benefits from work done in many fields, and its contributions lie not only in substantially extending and generalizing previous research results, but also in introducing several new concepts and abstractions that allow for tackling the challenging problem of qualitatively describing a class of complex, time-varying, spatially distributed nonlinear systems, namely pattern-forming diffusion-reaction systems. This chapter makes a brief review of that related work and explains how it relates to the contents of this thesis.

2.1 Spatial Aggregation

Spatial Aggregation was introduced by Yip and Zhao [11] as a unifying model for imaginistic reasoning following the implicit mechanisms of preceding problem solvers such as HIPMR [16], that analyzes the kinematics of fixed-axis mechanisms; MAPS [12], that performs a geometric analysis on the state equations of a dynamical system to design control laws from them; and KAM [39], that interprets the behavior of Hamiltonian systems from their phase-space diagrams. SA was further developed into a language by Bailey-Kellogg, Zhao and Yip [2], and later turned into a full-fledged programming environment by Bailey-Kellogg [1]. SA formalizes operations
to form conceptual aggregates from physical fields. This thesis develops STA, which stems from SA to introduce new operations that allow to reason about time-varying fields.

2.2 Spatio-Temporal Reasoning

Spatio-temporal Aggregation as developed in this thesis focuses on the representation of discrete events in physical processes. There has been much work in the field of qualitative reasoning focused on what has come to be known as the process ontology [11], that formalizes the qualitative notion of physical process, and introduces ideas such as using ordinal relationships to provide qualitative representation for numerical values (that is, a discrete order of events is used to represent a continuous numerical process). Other concepts such as functions and causal relationships are also given qualitative definitions in process ontologies. The sub-field of qualitative reasoning that has process ontologies as subjects of study is known as qualitative process theory.

Some of the ideas of qualitative process theory are further developed through qualitative physical fields [19], which seeks to model the spatio-temporal behavior of distributed parameter systems based on physical fields. The purpose of such models is to support the qualitative methods used by scientists in disciplines such as ecology (where there are few processes with known numerical models) or weather forecasting, which goes from accurate numerical simulations to subjective qualitative discourse that makes reference to such fuzzy abstractions as rain regions or cold fronts. STA shares with this discipline the goal of modeling high-level, qualitative discourse that
is centered around a process (or event) ontology, rather than an object (or "being") ontology.

Although STA has much in common with qualitative process theory, there are some differences. QPT models have two main components: the static model and the dynamic model. The former is a qualitative model of "interesting" regions of uniform value within a field; the latter represents discrete temporal events as "regions of influence," or (possibly overlapping) zones in the field where qualitative processes take place. These two sub-models are roughly equivalent to the static aggregation and iterative change propagation steps of STA, the main difference being that STA approaches description bottom-up, by abstracting layers of low-level change into higher level events. QPT has qualitative simulation as a goal, and hence is plagued by ambiguity problems inherent to predictive qualitative reasoning. STA, on the other hand, focuses on description as a means to establish a qualitative difference of behavior among quantitatively similar systems.

2.3 Computer Vision and Object Recognition

The field of computer vision is concerned with the design of algorithms that can interpret the contents of a sequence of images. This can be used, for example, to enable a robot to understand its environment using camera sensors. Such complex task requires, among many other things, to implement the ability to track moving bi-dimensional representations of objects. This goal bears some resemblance to what STA seeks to accomplish: to track over time the behavior of changing objects. However, computer vision is less concerned with inherent change and more with the maintenance of coherent representation of moving objects.
Computer vision algorithms have to solve problems such as that of color constancy [21], which is the perceptual mechanism which provides humans with color vision in a fashion relatively independent of the spectral content of the illumination of a scene. Various algorithms attempt to correct for various variables such as illumination and reflectance, and then track moving objects based on the (hopefully) uniform color information left after the processing. We are less concerned with the fact that such a difficult problem needs to be addressed, than with the fact that computer vision algorithms use notions such as color or shading for tracking. STA, as well as its predecessor, SA, regard uniformity of features for a field region as a criterion for the identification of coherent structures. STA extends this idea further to the domain of efficient tracking by means of the selective processing of change.

2.4 Tracking Coherent Objects in Massive Data Sets

The problem of tracking aggregated objects in a varying field has been studied by a number of researchers. The notion of object coherence itself has been studied in the field of cognitive science [32], where several features of coherency (from a human perspective) have been identified, namely connectedness, cohesiveness and continuity. Villasenor and Vincent [36] developed an algorithm to track vorticity tubes in turbulence, and Silver and Wang [29] carry that approach further and draw some generalizations about the problem of tracking turbulent features in three dimensions. Their approach uses similar ideas to the ones STA uses to classify catastrophic transformations. There has also been research aimed at formalizing temporal tracking of features for a class of domains [28]. Yip also addresses the problem of tracking aggregated objects in turbulent fluids [40]. His approach is based on Spatial Aggregation.
In this thesis we take many of these ideas and extend them into a general model to reason about time-varying fields.

2.5 Diffusion-Reaction Fields

This thesis applies STA to the problem of extracting qualitative descriptions of the behavior of diffusion reaction systems. The fact that these systems may, under certain circumstances, exhibit interesting spatial patterns, was discovered by Turing [35]. His insight into this discovery was bright, particularly given that in his time there were no computers, and no computer simulations. Turing did go through his equations with the help of a desk calculator, drawing by hand what he found.

However, Turing's discovery was preceded by Belousov's observation of actual chemical patterns in 1951, and Bray's observation of an oscillating chemical reaction in 1921. These phenomena only began to receive wide attention in the 1960s, when Zhabotinski reproduced Belousov's reaction. The Belousov-Zhabotinski (BZ) reactor was later simplified into what now is known as the "Oregonator," and its intriguing properties are still studied by groups such as that of Swinney [18, 13, 23].

Another rich diffusion-reaction model, the Gray-Scott system, was studied by Pearson [21], who discovered that it exhibits a very surprising variety of behaviors. In this thesis we look at the behaviors of some of these systems and design a computer program that can differentiate between qualitatively different behaviors.

2.6 Kinetic Data Structures

The problem of identifying the conditions under which certain geometric data structures suffer structural change has been given ample attention by the field of
kinetic data structures [3]. This field is concerned with systems of objects that bear relationships to each other: when these objects move, some of these relationships may be altered. KDS seek to certify when the data structure that represents all these relationships is valid, and under which conditions it ceases to be valid; further, it seeks to find efficient methods to repair such data structures. SA has the neighborhood graph as an essential component. When temporal variation is introduced into a field, this graph may suffer changes. SA needs to address the problem of how to identify changes and repair damages, so it has a set of problems in common with those KDS. In this thesis some of these commonalities are explored.

2.7 Pattern Recognition and Shape Classification

There are many approaches to pattern recognition, from the very general, such as neural networks, to the very specific (and highly effective), such as special purpose application for recognition of fingerprints [27]. We are interested in approaches that can be computed efficiently and repeatedly, without the need of training stages or other computational overhead. Because we approach the problem of classification from an Spatial Aggregation perspective, and because SA defines similarity based on spatial proximity, we need a method to quantify similarity, or difference, between coherent objects (see Section 1.1).

The problem of measuring shape differences in two dimensions has been considered by Dudek and Tsotsos [9]. This thesis borrows their technique and extends it for the purpose of classification of shapes that cannot be otherwise differentiated (because they have no qualitative differences, such as the presence of holes in them).
There are a number of techniques for comparing shapes. For instance, in [5], a stick figure or “backbone” is used to match protein molecules in three dimensions. That approach is useful for objects that have a clear medial axis. In contrast, the approach taken in this thesis matches the *contour* of objects rather than their skeletons.
CHAPTER 3

STA: SPATIO-TEMPORAL AGGREGATION

This chapter presents the key elements of spatio-temporal aggregation. Section 3.1 briefly reviews Spatial Aggregation (SA) and Section 3.2 explains in detail how SA is conceptually extended to address issues arising from temporal changes in data structures.

3.1 Spatial Aggregation

Spatial Aggregation (SA) provides a uniform mechanism to represent and reason about spatial fields using a multi-layer approach. Several layers of interpretation are built above the spatial field, which constitutes the base layer. Each layer above the base makes explicit representation of objects that are formed as aggregates of objects from each immediately preceding level. Objects are aggregated according to their adjacency relations to other objects and to their feature values; the specifics about how this is done are domain-dependent. Feature values are simply properties or sets of properties associated with objects; examples of features are color, temperature, pressure, etc.
SA facilitates a generic set of operators that provide a uniform vocabulary. This vocabulary is used to talk about the task of carrying out a conceptual abstraction of the features of a field. The fundamental components of SA are:

- **Spaces and Objects**: The basic domains for reasoning are spaces that contain objects. SA makes spaces to be containers of spatial objects, possibly providing a metric to determine spatial relationships among objects, such as orientation, position, distance, etc. In a typical application, the basic objects in a space are very low-level constructs, characterized by a small set of properties.

- **The Neighborhood Graph**: SA uses the Neighborhood Graph as a common interface to represent adjacency. A neighborhood graph is computed on the underlying field, as the first abstracting step to make explicit representation of primitive objects and their adjacencies, and at every subsequent level of abstraction. Notice that the concept of neighborhood graph is generic enough to include any data structure that represents adjacency, because the notion of nearness is domain-dependent.

- **A Space Partitioning**: Once an adjacency relation is explicitly established by means of a neighborhood graph, it is necessary to extract clusters of adjacent objects that are meant to represent meaningfully distinct groups. All nodes in a group are considered to belong to a class, and the group itself has an explicit representation.

- **Object Redescription**: Having classified the objects into distinct groups, an abstraction step allows to treat a class of objects as a single object in a subsequent layer of abstraction. The geometric and topological properties of an abstracted
object are derived from its components in the previous layer in a domain-specific manner.

The above set of operations defines a layer of aggregation. Once a new layer is established, the spatial relations among these objects can be found, and they can be aggregated, classified or redescribed into new layers, as many times as required by the application.

### 3.1.1 A Uniform Vocabulary

As seen above, there are three steps in SA that allow forming several layers of abstraction with enormous flexibility of implementation. We associate explicit operators with each step:

- **aggregate**(*Field || Object Space*): Form a Neighborhood Graph either on the underlying field or on object spaces at higher layers of abstraction.

- **classify**(*Neighborhood Graph*): Form equivalence classes

- **redescribe**(*Set of Classes*): **localize**(*Set of Objects*): Inverse operations that, respectively, abstract a class as an object in the next layer, or retrieve a class in the previous layer corresponding to an object.

Figure 3.1 shows a schematic representation of SA.

### 3.2 Temporal Changes

Existing applications of SA abstract over domains such as phase spaces and configuration spaces, in which time is only implicitly represented. Others deal with physical spaces in a fixed, steady state. In all these cases the field, as an ontology, and all the
Spatial Aggregation represents various layers of conceptual abstraction in a uniform manner, dividing each abstraction step into three operations: aggregate to form Neighborhood Graphs, classify to form Equivalence Classes, and redescribe to abstract each equivalence class as an object in the next layer; its inverse operation is localize.
conceptual layers built on top of it, are static. The process of abstraction does not require the concept of dynamic transformation through time.

However, there is a set of interesting problems that involve dynamic transformations in scalar and vectorial fields. Scientific visualization, a subfield of computer graphics, deals with some of these problems, and there is research in the area dealing with issues such as tracking vorticity tubes or other 3D features in turbulence [36, 29].

There has even been research aimed at formalizing temporal tracking of features for a class of domains [28]. This section explores some of the issues involved in these temporal explorations, including previous formalizations carried out in related research, and proposes an extension of the unified vocabulary of SA, by itself an inherently spatially oriented framework, adding a temporal dimension to it.

Notice that problems that use time per se are not necessarily outside the domain of Spatial Aggregation. For example, KAM [39] is used to study Hamiltonian systems that describe frictionless motion. These systems are studied in phase space, where temporal variation is implicitly represented. More in general, SA could be used to study time-varying systems as simple static systems where time has been represented as an extra spatial dimension. What STA offers beyond such approaches is the ability to reason about time-varying systems without having to compute and store the entire space-time volume beforehand. STA allows for the observation and representation of events as they happen, a fact that might be useful for real-time systems.

3.2.1 Time and Identity

We make a distinction between the classical notion of identity, which simply states that objects are equal to themselves, and a more practical idea, which adds the notion
of persistence in time. The practical notion of identity is necessarily heuristic: we establish a “near-identical” relationship for objects across time based on the similarity of their properties.

A far less philosophical, and far more practical instance of the problem of establishing a heuristic identity arises when dealing with abstracted constructs in fields that change in time. In computer software, the identity of an object may be associated with its location in memory, for instance. Fields that are to be visualized are usually discretized as grids of pixels in 2D or voxels in 3D; identifying each pixel or voxel is a simple matter, because a unique number can be easily associated to it. On the other
hand, high-level features of a raw field are not explicitly represented anywhere (see Figure 3.2). SA creates explicit representations by aggregating regions of a field and assigning identifiers to those regions, so these regions can be later treated as explicit objects. When temporal change is added, the problem is made more complex by the need to maintain explicit representations of changing objects.

The problem of tracking the identity of an object in time is usually known as the correspondence problem. In most cases, the input is a sequence of snapshots of the data corresponding to consecutive time stamps for a given discretization of time, uniform or not. On output, objects in successive snapshots are made to correspond to each other using a particular heuristic. This heuristic may be domain-dependent, if some of the properties of the objects are known to have some invariance characteristic useful to establish correspondence. Barring that, generic methods can be used, such as extracting regions of maximum overlap, or identifying the object with the nearest centroid in the successive snapshot. Such methods assume that the temporal sampling is good: objects do not change too much from frame to frame, both in terms of position and feature values.

To be more formal, we first introduce a few definitions and criteria to establish correspondence:

**Definition 3.2.1** An elementary object is an object that is not composed of simpler objects. Instances of elementary objects are pixels in a grid or particles in a sampling particle system.

**Definition 3.2.2** A complex object is an object that is not an elementary object.
Definition 3.2.3 A feature or property of an object is a member from some space that is associated to the object. For instance, color would be a feature or property of a pixel.

Definition 3.2.4 A uniting feature is a minimal set of properties of elementary objects that is sufficient as a criterion to determine whether they belong to the same class.

Definition 3.2.5 A coherent complex object is a set of adjacent simple objects that share a uniting feature at any particular time $t$.

Definition 3.2.6 An elementary object is said to be unchanged with respect to a complex object from time $t$ to time $t+1$ if at both times it has the same uniting feature that defines the complex object. An unchanged elementary object is said to belong to the complex object at both times, $t$ and $t+1$.

Definition 3.2.7 Let $O_t$ and $O_{t+1}$ be coherent objects that exist at times $t$ and $t+1$ respectively, both sharing the same uniting feature. Let $|O_t|$ and $|O_{t+1}|$ be the number of elementary objects contained by each complex object. Let $O = O_t \cap O_{t+1}$, and let $\alpha = |O_t|$. The two objects are said to $\alpha$-correspond if and only if $\gamma > \alpha|O_{t+1}|$ and $\gamma > \alpha|O_t|$. Each of the two conditions are called $\alpha$-forward-correspondence and $\alpha$-backward-correspondence respectively.

Definition 3.2.8 If, for a sequence of coherent objects in a time period, all two successive objects are $\alpha$-correspondent, then the sequence is said to be $\alpha$-correspondent.

Definition 3.2.9 An object is said to preserve its identity during a time period if it is associated with a unique $\alpha$-correspondent sequence in that time period.
Although the value of $\alpha$ is domain-dependent, in general it makes little sense to use values smaller than 0.5, because doing so would allow correspondence to be determined by a minority of the object. On the other hand, values much greater than 0.5 would make object correspondence depend on minute changes in objects. For these reasons, values of 0.5 or slightly greater seem appropriate in most cases.

The correspondence problem is complicated by the fact that objects may change substantially in time, either through modifications of their features, be they geometric, topological or other, domain-dependent features, or through changes in their existential status: objects may appear or die out, or they may split into many or many may fuse into one. Notice that from the above concepts it is relatively straightforward to define such events. Collisions and splits occur when an object $o$-corresponds to the union of a set of objects in a previous or subsequent time step, respectively. Death and birth are established when an object does not $o$-correspond to anything else in the immediate future or past. These themes are revisited in Section 4.2.

3.2.2 Aggregation and Persistence

This section introduces the main temporal extensions to SA. Each step within an aggregation layer is examined, and the concept of update is explained for each case.

Usual approaches to temporal tracking deal with a single level of aggregation: objects such as vorticity tubes are usually first-level aggregates of primitive elements, usually pixels or voxels in a grid, or other unstructured data sets [30]. As seen before, sophisticated techniques have been developed to deal with this problem. For example, Wang and Silver track coherent turbulent vortex structures in 3D using a maximum superposition criteria as a heuristic, as shown in Figure 3.3. It would seem natural
to try to find whether there is a natural generalization of these tracking approaches, which would let them deal with not just one, but multiple abstraction layers, in the SA style.

The first addition made to the SA standard vocabulary is the *update* operator, which takes a field or an object space and applies a set of transformations corresponding to the passage of a time step. This operator should not be confused with the one used for SA-based control systems, which generate control actions that propagate downwards through the aggregation layers to eventually manipulate the field. Instead, the temporal update operator *expresses* a change brought about by the nature of the phenomenon under study, rather than *produces* it.

The update operation for SA is by no means a simple action. It allows for changes in an object's features, position and existence, and it affects all levels of conceptual entities: objects, neighborhood graphs, equivalence classes and inter-layer mappings. The very notion of update implies the premise that these conceptual entities are *persistent*, at least in a limited sense. For example, a neighborhood graph on a particular abstraction layer at time \( t + 1 \) should be conceived as a *revision* of the graph on that layer at time \( t \), rather than as a new construct built from scratch. Similarly, space partitions and redescribed objects should also be considered to be revisions of previous instances. This idea is, however, independent of the actual implementation of update: even if all entities are re-built from scratch at each time step, they are conceptually persistent if adequate correspondence mappings are drawn to and from the previous step.

In some cases, the update operation needs to be divided into elementary sub-operations. Next, a case-by-case analysis of the update operation is presented.
Figure 3.3: Four snapshots of a three-dimensional representation of an evolving set of coherent turbulent vortex structures. Temporal correspondence is established by means of a maximum overlap heuristic. Reproduced with permission from [37].
Updates on the Space

Taking a rather Kantian-Newtonian approach, space is considered in STA, along with time, to be fixed and immutable. Hence only objects contained by the space can change. They can change either their spatial properties (position, orientation, etc.) or their intrinsic properties (color, temperature, pressure, etc.). Furthermore, they can also change their existential status: they may come into existence or cease to exist, or they may split or collide.

Because the notion of adjacency depends exclusively on spatial adjacency, only spatial or existential changes can affect neighborhood graphs of objects. On the other hand, the intrinsic properties of objects, along with other information such as adjacency, determine whether they belong to particular classes, so changes in these properties may influence the way objects are classified. In turn, changes in classes may have an effect on the re-description of higher-level objects.

Updates on Neighborhood Graphs

When the objects in space come into existence, cease to exist or change positions, their adjacency relations may be modified. The changes in the neighborhood graph due to a change in a single object may remain localized in space, or may propagate everywhere, depending on the nature of the neighborhood graphs. For instance, in a k-nearest neighbors graph, changes that do not lead to large displacements in object positions will remain localized, while in a minimum spanning tree even those small changes may propagate globally, because the minimality of such trees is itself a global property (for instance, if an edge that belongs to the MST becomes elongated, it may be necessary to drop it from the MST in favor of a far away, shorter edge). In either
case, it is necessary to identify which adjacencies are preserved, which are created, and which are destroyed as a result of the update. Thus the change is represented as a structural difference.

**Updates on Object Classes**

Adjacency is a fundamental criterion to establish object equivalence. Therefore, changes in adjacencies may cause objects to cease to belong to a certain class or to start belonging to a new class. On the other hand, even if the adjacencies are not altered, changes in the intrinsic properties of the objects may also affect the way they are classified. Changes in classification are annotated as sets of objects added or removed from each class, as well as classes that are newly formed or newly deceased.

**Updates on Redescribed Objects**

Changes in classes of objects may affect the way higher level objects are redescribed, depending on what features are kept in the redescription process and which are abstracted away. For example, if clusters of objects in space constitute classes and they are redescribed as convex hulls, internal changes in the clusters do not affect the higher level objects as long as they do not involve the hull (see Figure 3.4). Therefore it is necessary to have mechanisms that detect when lower-level change affects the structure of higher-level redescribed objects.

### 3.2.3 Change as Detection of Structural Failure

This section explains the relationships between updates at various aggregation levels, and how the use of KDS may help determine such relationships.
Figure 3.1: A space has been partitioned into point clusters, and each cluster (or class) has been redescribed as the convex hull of the cluster. From instant (a) to instant (b) point P has moved, but the movement is not large enough to affect the convex hull of its cluster. Hence this change is not propagated upwards in the abstraction chain.
At each step of the temporal aggregation chain there is a description of change up to that step. Several mechanisms are needed to determine how much of this change is relevant for the subsequent step, and how the structures at that step are affected.

Whatever the abstraction step, change can always be conceptualized as structural failure. If the lower level change is not sufficient to disrupt any of the higher level structures, (such as in the convex hull example above) then that change is not propagated upwards in the chain.

At the beginning of aggregation level, change occurs in objects and may be existential, positional or intrinsic, as seen previously. Deciding whether intrinsic change should propagate upwards is a simple matter, because a predicate that determines whether particular objects should belong to particular classes is already defined as part of the classification step. On the other hand, existential and positional change require the use of additional knowledge.

Existential change in objects will almost always cause structural change in the neighborhood graphs, because the appearance or disappearance of an objects will cause the creation or destruction of adjacencies between the object and other objects in the space. An exception to this rule are neighborhood graphs that allow certain objects not to have any adjacency, such as one that defines as adjacent to an object all other objects within a certain radius: objects that are farther from any other object than this radius would not have any adjacencies. Still, even in such extreme cases existential change may propagate through the classifier.

On the other hand, positional change may or may not cause structural change in the neighborhood graph. If an objects moves a distance small enough that none of the constraints that define the existent adjacencies are violated, the structure of the
graph will be preserved. Also, structural change may be present but remain confined to a small region of the graph. Conversely, the change may be of such nature that the entire graph is affected.

**Kinetic Data Structures: Reasoning about Change Detection**

There is a large body of research focused on geometric data structures that are subject to change due to displacement of their elements. The generic name for these is *kinetic data structures* (KDS) [3]. The problem consists of determining under which conditions the structure of certain geometric constructs is altered given that the elements are subject to particular motion laws.

KDS are usually about maintaining knowledge about the relationship between several moving objects. For example, one such piece of knowledge could be the separation between the closest pair of objects, a datum that could be used for collision detection. Another could be a minimum spanning tree connecting the objects, which could be used as a way of maintaining a minimal-cost network of message-relaying among the objects.

KDS work under the assumption that the motion of an object is continuous and follows simple laws (that is, it can be described by linear or quadratic equations). This allows the avoidance of fixed-step motion simulations, which may either over or undersample the system, which would either waste computational resources or miss critical events, respectively. The continuous approach is focused on finding exact solutions to equations derived from the motion laws, that would determine when a structural failure (requiring an update of the discrete knowledge we have of the system) is going to occur. The system can then be “fast-forwarded” until that event. For example, if we are keeping track of the closest pair of objects for a set of objects
that move linearly, we can calculate the exact moment in the future when the current closest pair ceases to be the closest and is replaced by another pair.

Structural failure in a KDS is detected by maintaining a set of validity certificates, predicates that determine the conditions under which the current conditions of the system are valid. When a certificate is violated, an event is said to occur. The event is then processed and the certificates are updated to reflect the new conditions. There are various properties that are desirable for these certificates, such as locality (each object is involved in few certificates), compactness (there are few certificates) and responsiveness (processing a certificate is cheap).

The existing corpus of research on finding and maintaining good certificates for various data structures is rich and varied. This thesis is much less concerned with the particulars of each data structure and its maintaining algorithm than with the fact that such algorithms exist, and that they all fit within a simple model of change as a violation of a certificate. Because this thesis deals with various levels of description, it is necessary to add conceptual mechanisms to determine the relevance of each certificate for the structure at the next abstraction level.

Notice also that while the event-drivenness of KDS is extremely useful in finding the minimal amount of computation required for a particular simulation, it depends on having the objects behaving in predictable, simple manners. Such a constraint cannot be forced into a general conceptual framework. For this reason, event-drivenness is not a conceptual element of STA; however, special implementations for particular data structures in the STA library elements of SAL can be restricted in such way that they are event-driven. STA itself neither requires nor forbids event-drivenness for its temporal concepts.
Figure 3.5: Three snapshots of a small particle system whose objects suffer positional change. The neighborhood graph is defined by nearness within a small radius. As the particles move, a moment is reached when the graph suffers a structural failure: an adjacency is broken in (c). This forces the need to record a high-level change event: a single object (represented by the curved line around the graph) has split into two smaller objects.

Update Mechanisms

We enhance the static SA to include certificate-violation based update mechanisms borrowed from KDS. This is done first at the neighborhood graph level, by associating the graph (namely, its vertices and its adjacencies) with a set of certificates that establish how much deformation the graph can take without suffering a structural change. The classifier operator now does not only map objects to classes via the neighborhood graph, but it also maps graph changes due to certificate violations to class changes.

The certificate-violation mechanism is extended to include detection of non-geometrical change, namely, change in intrinsic object properties and existential change. The former kind of certificates exist at the classifier level, but not at the neighborhood graph level: typically it will consist of simple inequalities that test whether certain object features are within certain ranges. The latter exists at all levels.
Also, detection at the redescription level requires being able to determine what low-level objects are relevant to the structure of higher-level objects (see Figure 3.5). Because we know which objects are involved in which certificates, all certificates that contain relevant lower-level objects are certificates needed for redescription. Such a filtering scheme is general, and allows for an unifying method to reason about abstraction of change.

The next section explains the concepts of change abstraction in a particle system in two dimensions using a Delaunay triangulation as a neighborhood graph. However, the various concepts are domain independent, and they extend to many other kinds of structures, such as minimum spanning trees, $k$-closest neighbors, etc. What varies from structure to structure are the actual algorithmic implementations of methods for change detection and structural repairs. Those methods come from the existing KDS research corpus. The abstract programming interface, introduced in Section 3.3, is generic and domain-independent; this will be illustrated by the example in Section 3.4, which uses quite different data structures from those explained in the next section.

### 3.2.4 Abstracting Change in a Sampling Particle System

The use of particle systems as sampling constructs is quite common. Sometimes it is necessary to aggregate several particles together to represent high level objects whose existence is implied by the object being sampled. For example, a particle system sampling a temperature field might be aggregated into regions that correspond to high temperature and low temperature.

Assume that the sampled field is variable, and that the sampling particles adjust themselves to changes in the field. These changes in the sampling system require
to update the particle aggregates as the particles move. A neighborhood graph that can be fixed locally and that is easy to maintain needs to be used. If the problem is restricted to the plane, one such graph is the Delaunay triangulation, which is local because it is sufficient to verify the local satisfaction of the circumcircle test: for each edge, check that neither of its two adjacent triangles’ circumcircles contain the opposing vertex [8].

Because a particle system in the plane is meant to sample a continuous area, a way to define a region for a cluster of points needs to be established. Simple ways to define such regions are to use a bounding box or a convex hull. We will use the slightly more accurate method of aggregating polygonal structures as sets of adjacent triangles, where each triangle is considered to belong to the region if and only if its three vertices can be determined to belong in the region (all three are “low temperature,” for example). In Figure 3.6 an example of one such structure with two holes is shown.

The three types of changes stated before will be considered: existential, intrinsic and positional. For simplicity, it will be assumed that only one of these changes can occur at each particular time for each particle.

**Intrinsic Changes**

Let $Q_t$ be the set of all triangles that belong to a polygonal structure at a given discrete time $t$, and let $A_t$ be the set of all triangles that are adjacent to any border of the polygonal structure, but that do not belong to it; that is, $A_t$ is the set of all triangles $q_t$ such that there is an edge $e$ of $q_t$ that is also an edge of some triangle $q_t' \in Q_t$, but $q_t \notin Q_t$. Then, let $P(q_t)$ be an intrinsic property (such as being within a certain range of temperature, color, etc.) that is true if and only if at time $t$ triangle
Figure 3.6: A polygonal aggregate in a Delaunay triangulation, represented by the gray region. The aggregate is constructed out of particles that share a uniting feature. The resulting aggregated object is a polygon with two holes.

$q_t$ qualifies to belong to a polygonal structure. Then border invariance is established as follows: a structure is invariant from time $t$ to time $t + 1$ if two conditions are met:

\begin{align}
\tau q_t \in Q_t & : P(q_{t+1}) \\
\tau q_t \in A_t & : -P(q_{t+1}).
\end{align}

(3.1) (3.2)

Notice that the empty structure, for which $A_t$ is the universe, has to be explicitly tracked; when the second condition is broken that structure ceases to be empty, and a new empty structure needs to be created. Therefore, certifying the non-appearance of new polygonal structures requires an exhaustive search in the space of all triangles that do not already belong to a polygon. Such a search has a linear worst-case complexity, and hence it is unlikely to have a significant impact in the efficiency of the whole algorithm, because the necessary task of tracking the change of all vertices in the triangulation also has linear complexity. On the other hand, verifying absence
Figure 3.7: When a triangle violates conditions 3.2, switching the "border status" of its edges with respect to the polygonal aggregate is necessary and sufficient to fix the border of the aggregate. See main text for a proof.

of structural change in existing structures only requires looking at the components of the structure and its immediate vicinity.

What to do when one of the above conditions is violated? The offending triangle \( q \) needs to be detected (there may be more than one, in which case we repeat for each) and proceed as follows: invert the "border status" of all edges of \( q \); those already on the border should cease to be, and those not on the border should be added to it. These operations may cause two or more borders to collapse into one, a new border to be created, or a border to disappear. Notice that these operations are applied without concern as to which particular condition was violated.

**Theorem 3.2.1** Changing the border status of a triangle that violates conditions 3.2 correctly fixes the border of a polygonal aggregate.

**Proof:** We consider the four possible cases, depending on whether the offending triangle shares zero, one, two or three borders with the polygonal aggregate:

- **Case 1:** The offending triangle does not share any borders with the polygonal aggregate. In such case, none of the triangle sides will be in the border of
the aggregate; switching their status makes all of them to be in the border. If the triangle was outside the polygonal aggregate, this process simply creates a new aggregate consisting of exactly one triangle. If the triangle was inside an aggregate, the process adds a triangular hole to the aggregate.

- Case 2: The offending triangle shares exactly one edge with the polygonal aggregate. Refer to Figure 3.7. If the triangle is outside of the aggregate (Triangle $T_1$ in the Figure), then a change would add it to the aggregate. This requires edge PR to be removed from the border and edges PQ and QR to be added to it; a border status inversion achieves exactly that. Similarly, when the triangle is inside the aggregate (Triangle $T_2$ in the Figure), removing it from the border requires removing side UV and adding sides UW and VW.

- Case 3: The offending triangle shares exactly two edges with the polygonal aggregate. The proof of correctness here is exactly the reverse of Case 2: removing Triangle $T_1$ in the Figure (assuming it belongs to the aggregate) requires switching back all the borders; adding Triangle $T_2$ requires the same.

- Case 4: The offending triangle shares all of its borders with the polygonal aggregate. This is only possible if the triangle occupies the space of a triangular hole inside of the aggregate, or if the aggregate is the offending triangle. Switching the status of all three edges makes this internal border, or the lone-triangle aggregate, to disappear, effectively plugging the hole with the new triangle, or destroying the aggregate.

Q.E.D.
Existential Changes

Because particles are considered to be elementary objects, we only consider two existential changes: insertion of a new particle and deletion of existing particles. These operations have been implemented using variations of De Floriani’s [7] and Heller’s [14] algorithms.

- Deletion: First look at the conditions under which a vertex may be deleted without compromising the border. Because deletion will not cause any triangle not adjacent to the deleted vertex to cease being valid, only the polygon immediately surrounding the deleted point will be affected (see Figure 3.8). Unless the border of the structure has edges inside this polygon, it will not suffer a structural change as a direct result of the deletion. Because all vertices inside the polygon are incident to the point to be removed, it immediately follows that point deletion of vertex \( P \) will directly cause a structural change if and only if \( P \) lies on the border of the structure.

Figure 3.8: When vertex \( P \) is removed, only the gray area, comprised by triangles adjacent to \( P \), will be affected.
What to do if the condition is violated? After the affected area is re-triangulated, predicate $P$ must be applied to the new triangles to determine which must belong to the structure, and then the border must be fixed accordingly.

Vertex deletion can still cause a change in the border even if the point does not lie directly in the border. A deletion causes a region to be re-triangulated, and predicate $P(q)$ may evaluate differently for some of the new triangles than it did for those that existed before the deletion. For this reason, a second invariance condition for point deletion is needed. Let us say that deletion occurs at time $t + 1$. Let $A_t$ be the area comprised by the triangles that had as vertex the point to be deleted, and let $A_{t+1}$ be that same area after the deletion has occurred (same geometric space). Because the point does not lie in the border, all triangles in area $A_t$ (either of them) must evaluate to the same value; hence invariance occurs if the following condition is maintained:

$$\exists p : 7q \in A_t : P(q) = p \land 7q \in A_{t+1} : P(q) = p \quad (3.3)$$

- Insertion: Dealing with point insertion is slightly more complicated, because the extent of the affected area is not immediately obvious. In fact, this area is comprised by all triangles whose circumcircles contain the new point $[22]$ (see Figure 3.9). Notice that this area can be found before actually inserting the point, by recursively checking all adjacent triangles until no triangles are in the affected area, as shown the algorithm in Table 3.1.

Once the affected area is defined, we need to determine whether the border passes through it; this would be the case if some of the triangles of this area (those existing before the insertion) are inside the aggregated polygon, and
Figure 3.9: Point P is about to be inserted. In addition to the triangle that contains it, three other triangles have circumcircles that contain P. The affected area is the union of all these triangles.

<table>
<thead>
<tr>
<th>INPUT: Triangle $T$ containing point $P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure AffectedArea($T$)</td>
</tr>
<tr>
<td>$A = { T }$</td>
</tr>
<tr>
<td>for each edge $e$ of $T$</td>
</tr>
<tr>
<td>if some triangle $T'$ adjacent to $e$ not in $A$</td>
</tr>
<tr>
<td>if circumcircle of $T'$ contains $P$</td>
</tr>
<tr>
<td>$A = A \cup$ AffectedArea($T'$)</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>return $A$</td>
</tr>
<tr>
<td>end AffectedArea</td>
</tr>
</tbody>
</table>
some others are not. If such is the case, the border must be fixed using the new triangles, which come into being after the insertion.

Positional Changes

A vertex in a Delaunay triangulation has freedom to move without causing structural change as long as it does not enter the circumcircle of any triangle, and as long as it does not cause any opposing vertex to enter a triangle it is adjacent to, by virtue of the deformation of this triangle. This defines a continuous permissible area around the border, so predicting changes is a simple matter for any motion law as long as that law has an analytic or numeric solution for time given a set of space coordinates.

When a certificate is violated, there are two kinds of possible structural changes: those that occur when the moving point enters the circumcircle of a triangle outside its immediately enclosing area (Figure 3.10(b)), or when one of the deformed triangles adjacent to the moving point begins to contain another vertex in its circumcircle (Figure 3.10(c)).

Notice that for the purpose of determining structural changes due to point movement, it is not necessary to distinguish explicitly between cases (b) and (c); they are presented here for illustration purposes only. Change can be detected simply by determining whether an edge has ceased to exist as a result of the vertex movement.

Once the affected area is determined, we must check for changes in intrinsic properties, like what we did in the deletion and insertion cases.

**Theorem 3.2.2** The procedure described in this section is correct and complete.

**Proof sketch:** The procedure is complete because it exhaustively considers all possible instances of change for a triangulation: insertion, deletion and movement of vertices.
Figure 3.10: Effects of moving vertex $P$ to position $P''$ in a Delaunay Triangulation. Gray edges correspond to newly created edges after the movement operation. Thick edges enclose the region of the triangulation affected by the operation. (a) Moving point $P$ to $P''$ causes no circumcircle to be entered by any vertex, so the triangulation suffers no structural change. (b) When moving to position $P'''$ the circumcircle of triangle $q$ is entered, so a structural change is introduced. (c) Moving to $P^{'''}$ causes a change similar to the one in (b); additionally, the deformation of triangle $PFR$ to $P^{'''}FR$ causes its circumcircle to contain vertex $S$, thus requiring a second structural change.
as well as the loss or gain of triangles due to the change in intrinsic properties. It is correct because it identifies the extent of influence for each of these changes, and when the border of the polygonal aggregate is within this extent, it is fixed following a correct procedure (Theorem 3.2.1).

3.2.5 Tracking Change in Time

We have seen that we can develop a unifying reasoning scheme to deal with the propagation of change through an aggregation change. We now must focus on how to provide support for mechanisms that seek to interpret this change.

Relevance

Keeping track of change in a system may be useful in many applications. For example, by looking at the history of a weather system meteorologists are able to find tendencies and make predictions based on those tendencies. When studying transitional phases in self-organizing systems (such as the formation of convection rolls in boiling water) one might want to determine which kinds of events precede such transitions; for this a well kept record of events described at various levels of abstraction would be very useful.

We need, thus, to have a generic methodology to represent a history. More specifically, we require the ability to narrate a sequence of events that take place at various aggregation levels, namely changes in spatial objects, neighborhood graphs and object classes.
Exploiting Intelligent Updates

A history should register relevant change. The ease with which this can be done depends on how well the update mechanisms at various levels work. On one extreme, there is no attempt at any updates, and all structures are reconstructed from scratch at fixed intervals. In such conditions, finding relevant change is very difficult, since there is no knowledge to start with to draw correspondences. On the other extreme, a good update mechanism that operates on structures with a high degree of locality and that explicitly generates all change events requires virtually no extra work from a history-tracking mechanism, that only needs to log these changes with their respective time stamps.

Inherent Limitations of the Approach

The STA approach performs aggregation in space, and then tracks changes to the aggregation structure in time. It should be noticed that there is a large class of domains where this approach would simply fail, namely those where object aggregation for particular static frames is impossible, and coherence can only be established from temporal change itself.

The generic term for phenomena of this kind stems from gestalt psychology, and it is Objects by Common Fate. A typical experiment performed to illustrate this phenomenon is carried out by showing people apparently random collections of dots in the plane. People are unable to identify any coherent structure from these dots. However, when the dots are allowed to move, a group of them do so in a coherent fashion. People report seeing a large object move. The "objectness" of the large
structure comes from the shared kinetic behaviors of the constituting dots, rather than from any spatial structure.

For STA to be able to deal with objects by common fate, it would have to do aggregation on the temporal histories of objects, rather than on particular snapshots of the system. This would amount to doing SA on a space-time volume, something that STA is meant to avoid in the first place. However, real-time execution might still be possible if aggregation were done not on the entire space-time volume, but on small time intervals. After all, people recognize the instantaneous motion, and do not need to wait for the entire set of events to end before realizing that a complex object was moving. Such an approach would treat these small intervals in the same way that regular STA treats instantaneous snapshots. This kind of extension constitutes an interesting topic for future research on STA.

### 3.2.6 Temporal Aggregation: Reasoning About Change

In the same way that Spatial Aggregation represents sets of objects as higher level objects, an analogous temporal aggregation scheme should aggregate sets of events as higher-level events. One important difference is that whereas there is no inherent order in space (except for the one-dimensional case), time imposes a natural sequence of events; therefore any temporal-reasoning model should consider the concepts associated with temporal ordering, such as before/after. Of course, one can always impose an order along any particular dimension; however, such an order would be arbitrary. Temporal ordering is more natural, as it corresponds to our intuition and experience.

One obvious way to classify events is to look at their distance in space-time: if a set of events occurs one immediately after another, and if those events involve
Figure 3.11: Three independent events, (a), (b) and (c), represented as sequences of snapshots. In all events we see a cluster of elementary objects aggregated as a higher level object (represented by the curved outline). In each case, because the low-level objects shift their position, the high-level object undergoes a transformation. In (a) and (b), the object first opens up and then it splits into two separate objects. In (c), the object simply shrinks. We could compare all these events and conclude that sequences (a) and (b) are similar to each other and dissimilar from (c). A temporal aggregation scheme based on this criterion will cluster (a) and (b) together.
objects that are close together in space, then those events are considered to be "neighbors." Neighboring events can be clustered into a single high-level event. In fact, there is little more that can be done in terms of aggregating events that involve elementary objects. However, when looking at complex events (that is, those comprised of sequences of more elementary events), there are more criteria to choose from for the purpose of defining nearness. One such criterion is that of similarity of structure (see Figure 3.11). Whereas to infer a causal relationship given a single sequence of simple events amounts to committing the "post-hoc ergo propter-hoc" (after this, hence because of this) fallacy, we are in much safer grounds to infer such a causal connection when the same kind of event sequences are observed multiple times. However, although these ideas seem general enough, it should be up to each application that makes use of this generic methodology to decide which particular temporal aggregation mechanisms to use at each level.

In the next section the operation explained thus far will be introduced as formal operators.

### 3.3 STA Library Extensions

All change operators are application-driven, meaning that it is the responsibility of each STA application to call specific methods that cause change, and to ensure that the change is propagated through the aggregation chain. However, the framework maintains an application-independent mapping of update dependencies (that is, what is affected by each update operator) that does not have to be maintained by the application.
3.3.1 Spatial Change

Changes in space are introduced by changing the objects contained in the space; the space itself is considered to be immutable. Objects of the lowest level can only be changed explicitly by the application. Typically, these elementary objects will have little or no geometric structure (most of the time they will simply be points), but they can have several intrinsic properties, or features. In general, higher level objects should not be manipulated directly by the application; instead, their change arises as a result of the propagation of change from the lower levels. This change is brought about when the application calls the update operators; the application-independent dependency mapping ensures that these updates are applied where appropriate.

There is no generic interface to introduce change in elementary objects, because their structure is application dependent. However, since points are expected to be common as elementary objects, and they usually reside in metric spaces, functions to change their positions are provided. Additionally, a generic update function is defined for all objects, but it is meant to be effective only in the non-elementary cases: to gather change from the lower levels and propagate it to the current objects. See Table 3.2.

3.3.2 Neighborhood Structure Change

Neighborhood graphs are only concerned with the geometric attributes of objects in a space, and not with their intrinsic properties. Since these graphs have as their purpose to define adjacency relationships, and because these relationships reflect the notion of nearness in a metric space, it is necessary to have the ability to instruct the graph of changes in the position of objects (which would affect the measurement of
**SObject**: A spatial object

- **Geom**: A spatial object with geometric properties in a Metric Space, also known as a *geometric object*
  - **Point.Geom**: A geometric point.
    - `set.coord(i:INTEGER, v:FLOAT)`: set the coordinate `i` of the point to value `v`
    - `update()`: carry change from the lower aggregation layers

Table 3.2: Change Operators in Spatial Objects.

**Ngraph**

- **remove(p:Geom)**: remove object `p` and update adjacencies
- **insert(p:Geom)**: insert object `p` and update adjacencies
- **move(p:Geom, c:Coords)**: migrate object `p` to new coordinates `c` and update adjacencies

Table 3.3: Change Operators in Neighborhood Graphs.

their distances, and hence possibly the adjacencies). Similarly, the graph needs to be informed of existential changes. Refer to Table 3.3 for detail on these operations.

### 3.3.3 Change Operators for Spatial Partitions

Classifiers partition a space into disjoint sets of adjacent objects, where adjacency is defined by a neighborhood graph. Changes in adjacency relations should therefore be taken into consideration when evaluating whether the existing partitions, or classes, need to be updated.
Existential change in the objects of the space is the simplest to deal with as far as classes are concerned. Objects that cease to exist simply need to be removed from the classes that contained them, if any. Objects that come into existence, on the other hand, need to be evaluated to determine where they should be placed. This determination is application-dependent, and therefore mechanisms to add an object to a class explicitly are provided.

Other types of change are harder to assess in general with regard to their effect in a classification. Therefore several mechanisms to manipulate classes are provided. See Table 3.1 for details. Notice that adding objects to a class may cause this class to become adjacent to another class; if the two classes share uniting features, this may require them to fuse together. Four methods to make this determination are also provided. Similarly, removing objects from classes may break adjacencies in such a way that the class needs to be split into two or more. Both of these effects can also occur due to positional change.

### 3.3.4 Change Operators for Object Abstraction

Classes may suffer existential change, as seen above; this requires changes in high-level object abstraction. For example, if a new class is formed, a new high-level object corresponding to this class is created. More subtly, changes in the internal structure of a class may or may not cause the higher level object to change, depending on how the structure of the object relates to the structure of the class.

A method to create a new high-level object given a new class was already in place in the SA library. New methods to destroy said objects explicitly, and to gather
Classifier

- **remove(p:SObject)**: remove object p from the class that contains it.

- **add.to.class(p:SObject, c:Space)**: SET of Space: add object p to class defined by subspace c. Return classes to which this one becomes adjacent by effect of the insertion.

- **grow.class(c:Space)**: SET of Space: examine class defined by subspace c and its surroundings to determine whether it requires any modifications due to structural change in adjacencies: carry out necessary changes. Return classes to which this one becomes adjacent by effect of the update.

- **re.classify(c:Space)**: SET of Space: examine class defined by subspace c to determine whether it needs to be split into two or more classes: carry out necessary changes. Return new classes that result from the split.

- **remove.class(c:Space)**: destroy class defined by subspace c. This is usually called when a class becomes empty.

- **transfer(from,to:Space)**: move all elements of class “from” to class “to”. This method should be called when the two classes have become adjacent and it is determined that they should be fused into one.

Table 3.4: Change Operators in Classifiers. Notice that the type of classes, or space partitions, is Space. This is because spaces are simply containers of objects, and partitions, or subspaces, are also simple object containers.
changes in the class to reflect them into the objects’ structure are provided. See Table 3.5.

3.4 An Example: Tracking Wave Packets in One Dimension

In this section the use of the operators previously introduced will be illustrated by means of a very simple example of spatio-temporal aggregation.

It is common both in ordinary discourse and in the sciences to talk of mechanical waves as objects that move; their speed can be measured, their trajectory can be traced. However, the objects that constitute the medium through which the wave propagates do not propagate; instead, they oscillate within a bounded region of space. Waves are, therefore, not physical objects themselves, but conceptual aggregates of the behavior of many objects that compose the medium.

Consider Figure 3.12. A particle system samples an elastic medium in one dimension. Observers can think of the particles as representing marks painted at uniform intervals through an extended spring. As the systems evolves, each particle oscillates.
Figure 3.12: A particle system in one dimension samples the propagation of longitudinal waves. Frames 1 through 13 illustrate successive states of the propagation. Even though the position of each particle remains constrained to a small fixed region, observers can see two waves, one starting at each side, moving towards the center, colliding, and then bouncing back towards the extremes.
giving the impression that waves move through the medium. We are interested in developing a mechanism to not only represent explicitly the aggregated objects that observers perceive, but also to track their movement through time.

A simple way to conceive a longitudinal wave is to identify it as a cluster of sampling points. This makes intuitive sense, because in physical media longitudinal waves are defined as alternating regions of high and low pressure. Because the low pressure regions always occur in between high pressure regions, they can be neglected for the purpose of observation of movements.

3.4.1 An Aggregation Layer

The first thing that needs to be done is to frame the problem in terms of Spatial Aggregation. The particles shall be regarded as elementary objects in a one-dimensional space.

The natural adjacency relation is that of simple spatial proximity: each point is adjacent to the two points around it except for the extremes, of course, that are adjacent to only one point.

Having the adjacencies defined, space can be partitioned into point clusters. These partitions are then defined such that if two points $p_1$ and $p_2$ are adjacent and their distance is less than a fixed threshold $\xi$, then the points belong to the same partition. The threshold can be naturally associated with the distance between points in a rest state, assuming that the medium in such a state is uniformly sampled:

$$\xi = \frac{\lambda}{n},$$  \hspace{1cm} (3.4)

where $l$ is the length of the medium, $n$ is the number of points, and $\lambda \leq 1$ is a constant that translates as a tolerance to small peaks. When $\lambda = 1$, all points
// Create instance of adjacency test operator
Neigh_Test* nt = new Neigh_Test(&field);

// Create adjacency graph and aggregate field
ng = new Ngraph_Boolean(nt);
ng->aggregate(&field);

// Create instance of space partitioning test operator
Near* ne = new Near(&field, 0.85);

// Create classifier and classify neighborhood graph
cl = new Classifier_Transitive_Update(ne, ng);
cl->classify(ng);

// Create instance of redescription operator
Line_ redes* lr = new Line_ redes;

// Redescribe space partitions
ab = new Abstractor_Redescribe(lr);
ab->redescribe(cl->classes());

Table 3.6: An aggregation layer for the longitudinal wave example. This is a standard SA aggregation step, which needs to be done once as initialization.

that are closer than the average distance will be considered to be in a partition. By making λ smaller small perturbations from equilibrium are ignored. For experimental purposes a value of λ = 0.85 was used.

The last step in this single aggregation layer is the abstraction step: each space partition is redescribed as a single line segment that goes from one extreme of the partition to the other.

Table 3.6 shows code to carry out a static layer of spatial aggregation. This layer generates a set of lines, each associated with a cluster of particles. This code should be called as an initialization step, before the simulation is allowed to begin. After
that, all the objects generated by the initialization are iteratively updated, keeping track of what happens to them.

3.4.2 The Iteration Step

At each iteration step a numerical integrator updates the positions of the particles. This results in changes that propagate through the aggregation layer. This propagation has to be carried out at each step, and is analogous to the initial aggregation step.

Updates on the Neighborhood Graph

In general, the first thing to do in the iteration step is to detect whether the neighborhood graph needs to be updated. Changes in the elementary objects may cause some adjacencies to break and new adjacencies to form. In this example, the nature of the phenomenon under study is such that the adjacencies never change, because all points maintain their ordinal position along the elastic medium.

Updates on the Space Partition

In general, a space partition may need to be changed for two kinds of reasons: changes in features of the components (be they existential, intrinsic or positional features), or changes in the neighborhood graph. In this case, only the former kind of reasons applies.

Several events can take place at the partition stage, and all of them need to be detected. The possible events are:

- Partition formation: elementary objects that do not belong to any class may move close enough to each other for their distance to fall below the threshold.
giving rise to a new partition. This can only occur under aggregation schemes that are not exhaustive, that is, that allow some objects not to belong to any partition. Otherwise, such objects would belong to singleton classes, and the event described would be a collision rather than a formation.

- Partition collision: two or more partitions are fused into a single one. For this to happen, their extreme points must be adjacent and their distance must be below the threshold. When such a condition is detected, the larger partition is made to absorb the smaller ones.

- Partition separation: a partition splits into two or more smaller partitions. This happens when the forming condition is violated at some point inside the partition. The largest piece keeps the identity of the original partition, and the other pieces are assigned new identities.

Updates on the Redescription Map

The redescribed objects must suffer changes that are analogous to the changes in the space partition. Newly formed partitions are redescribed as new objects; partitions that cease to exist cause their redescribed objects to cease to exist, and changed partitions (via fusion or separation) require the corresponding redescribed object to be updated.

Carrying out updates in redescribed objects may benefit from the information collected through the process of updating the partitions, namely which elementary objects have been gained, and which have been lost. In general, there may be cases when not all such information is relevant for the purposes of redescription, and an intelligent redescription update operator could be able to discern the relevant from
the irrelevant. In this very simple example, no partition can be changed without requiring a corresponding change in the redescribed object. Update is, therefore, straightforward: the endpoints are moved to the extremes of the newly gained points, or to previously internal points when the current endpoints are lost.

Table 3.7 shows the code used for the iteration step. Various auxiliary functions are called, defined in Tables 3.8 and 3.9. Two additional auxiliary procedures, `merge` and `remove_dead_objs`, are not shown.

In Figure 3.13 the result of the described algorithm is shown. Where there were only oscillating particles, explicit moving objects with persistent identities are now represented by the system.

Notice that this very simple example illustrates the notion of change propagation, but does not carry out temporal aggregation per se. Although temporal events are detected, they are not aggregated, classified or even registered. All these more complex tasks will be presented fully in the next chapter, where an application of STA to the analysis of diffusion-reaction fields is developed.
// Advance the simulation one step
field.step();

// Declare containers for changed partitions
ClassSet all.emptied;
ClassSet all.split;
ClassSet all.born;

// Ngraph update would normally go here.
// Not needed in this example.

// Detect and add newly formed partitions
c1->re.classify(&all.born);

// Detect collisions
find.collisions(&all.emptied);

// Detect separations
find.splits(&all.split)

// Add new classes to classifier
c1->add.classes(&all.split);

// Remove dead classes from classifier
c1->remove.classes(&all.emptied);

// Propagate dead objects to abstractor
remove.dead.redes.objs(&all.emptied);

// Propagate changed partitions to abstractor
ab->update();

Table 3.7: Iteration step for the longitudinal wave example.
void find_collisions(Classes* all_emptyed) {

    // Explore all existing partitions
    TIterator<Space*>* cl_iter = cl->classes()->new_iterator();

    for (; !cl_iter->at_end(); cl_iter->advance()) {
        Space* cla = cl_iter->get();

        // Consider only non-empty partitions
        if (cla->size() > 0) {

            // Detect collisions
            ClassSet_FSL adjacent;
            cl->grow.class(cla, &adjacent);

            // If collision detected, merge
            if (adjacent.size() > 0) {

                ClassSet_L_E emptied;
                Space* ncla = merge(cla, &adjacent, &emptied);

                // Remember absorbed classes
                copy(&emptied, all_emptyed);
            }
        }
    }
}

Table 3.8: An algorithm to detect collisions and take action when they are detected.
Table 3.9: An algorithm to detect separations and take action when they are detected.
Figure 3.13: Thick line segments represent particle clusters that are redescribed as high-level objects. Various events can be identified through the evolution of the system. In frames 1 and 2 two initial objects begin moving from the extremes towards the center. In frame 3 a small spurious object separates from the rightmost large object, only to fuse to it again in frame 4. The objects collide and then separate again (frames 5 through 7), and in their way back towards the extremes they form various small spurious objects (frames 8 through 12). In the final frame (13) the objects reach the extremes and begin to bounce back, completing a cycle.
CHAPTER 4

APPLICATION TO DIFFUSION-REACTION SYSTEMS

In this chapter a detailed case study of an application is presented. The previously introduced Spatio-Temporal Aggregation will be applied to the observation, description, and classification of the behavior of a class of Diffusion-Reaction systems in two dimensions.

4.1 The Problem, Goals and Methods

Regions of uniformity arise in physical fields due to continuities of properties such as intensity, temperature or pressure [11]. As previously explained, a qualitative description of a physical field recognizes several events: the existence of coherent objects, their persistence through time, and their abrupt change.

A class of diffusion-reaction phenomena will be studied under this perspective. As seen in Section 1.1, these phenomena do indeed exhibit time-varying regions of uniformity that appear clearly coherent to the human eye. By categorizing these regions by their geometric and structural properties and by finding common patterns in the way they change, behavioral classes will be extracted.
4.1.1 Overview of the Algorithm

A structure-identifying algorithm that does not suffer from the drawbacks of naive
region growing and other standard methods will be introduced. It is based on the
central idea that qualitative structures of a spatial field can be constructed from an
adaptive spatial subdivision rather than directly from a regularly discretized field.
This adaptive subdivision changes as the field changes, but the identity of its struc-
tural components is persistent. The persistence of these components simplifies the
correspondence between successive temporal snapshots.

The algorithm comprises several elements, as shown in Figure 4.1. The system
simulator takes as input a field model, its parameters, and a set of initial condi-
tions, and numerically integrates the model to generate a continuously varying field.
After initialization, a neighborhood graph of the particles (also called "floaters") is
constructed. The particles are classified to form polygons describing regions of uniform-
ity. Tracking these polygonal objects over time, the algorithm generates a high-level
description of the history of the objects.

4.2 Behavior Description

In this section the mechanism to extract behavioral features from steady-state
diffusion-reaction systems will be explained.

4.2.1 Tracking High-Level Structures

The existence of coherent structures in a field implies that there are regions of
approximately uniform characteristics. For example, in the Gray-Scott model (in-
troduced in Section 1.1.2), each region clearly belongs to one of two classes, low or
Figure 1.1: A flowchart for the main components of the identification and tracking algorithm. Boxes represent operations, while ovals indicate objects.
The fewer the number of classes observed, and the larger the regions of uniform attributes are, the higher the organization perceived by an observer. As the number of classes increases, and as uniform regions become smaller, in the limit all coherence is lost. Notice that these two global attributes of a field, namely the size of regions of uniformity and the homogeneity of the object types may vary with relative independence of each other, and each contributes significantly to the perception of coherence. For these reasons, local uniformity is one of the main features to look for when studying patterns. Once regions of uniformity are identified, characteristics such as topology and temporal behavior can be studied.

Figure 1.2 shows four snapshots of an evolving Diffusion Reaction system that exhibits coherent objects.

Notice that in multi-stable Diffusion-Reaction systems the stable regions typically constitute solutions to the differential equations that describe the systems. Boundaries between regions tend to be very sharp (the field gradient is high, so choosing a
threshold value to separate the regions is not particularly difficult. Also, small variations in this threshold value do not significantly affect the behavior of classification algorithms.

Tracking coherent regions in Diffusion-Reaction systems can be achieved using many strategies. The most common is naive region-growing followed by brute force search for correspondence. In the worst case, solving this correspondence problem has exponential complexity. Even when that problem can be solved easily, the computational cost of representing and maintaining regions of pixels depends on the resolution of the discretized field. That cost is usually too high. It seems more appropriate to look for an alternate method that does not keep unnecessary detail (such as pixel values inside uniform regions) and whose cost is not dependent on the resolution. Next we introduce one such method, based on sampling by means of particle systems.

**Sampling Through a Particle System**

Diffusion-reaction fields will be sampled using a particle systems. Particles have the advantage of being persistent: they have discrete identities and hence whatever happens to them can be tracked in time with ease. Furthermore, any structures constructed by aggregating particles can also be tracked, because the identities of such constructs can be established recursively through a simple heuristic from the identities of its components. For example, one such simple heuristic is the following: if constructs $A$ and $B$, existing at different time instants, share a majority of their components, they can be said to be identical.

The particles must behave in such a way that they sample the field accurately. Therefore they must exist in large densities wherever the field gradient is large, and in
low densities where it is small. Because these particles can be pictured as free-floating in a waving field, we call them floaters.

Two different algorithms will be considered to allow the particle system to adapt itself to changes in the field, always maintaining an adequate sampling. The first algorithm is a modification of a method introduced by Witkin and Heckbert [38] to sample implicit surfaces. The second is a simple algorithm to maintain Terrain Irregular Networks through vertex repopulation and decimation.

**The Modified Witkin and Heckbert Algorithm**

The main idea of this algorithm is to allow floaters to move more or less freely in the field, interacting with one another. They repel each other, thereby tending to occupy space uniformly. Additionally, they modify their distribution and density to compensate for under or oversampling. Witkin and Heckbert’s algorithm is meant to sample implicit surfaces, namely those described by an implicit mathematical equation. However, it will be adapted here to be used on explicit fields. This is achieved by treating the field as an $n$-dimensional hypersurface in $n + m$ dimensions, where the field is a mapping from $\mathbb{R}^n \rightarrow \mathbb{R}^n$. In this particular case, a two-dimensional Diffusion-Reaction field, the field value of one of the reactants will be used as a third dimension. In many systems, such as Gray-Scott and Lotka-Volterra, the other field value is inversely correlated to the first, so it can be ignored; if this were not the case, a standard scalar metric of the field value could be used. The floaters interact with each other by computing their distances along this two-dimensional surface in a three-dimensional space.

To make floaters repel each other, following Witkin and Heckbert, a Gaussian energy function will be used. The value of function increases when floaters get close
together, and decreases when they separate. For any two floaters $i$ and $j$, their mutual energy is

$$E_{ij} = \alpha \frac{|r_{ij}|^2}{r_{ij}}.$$

where $\alpha$ is a global constant and $r_{ij}$ is the distance between floaters $i$ and $j$. The energy for each floater is given by

$$E_i = \sum_{j=1}^{n} E_{ij}.$$

Floaters are assigned a velocity that is negatively proportional to the gradient of energy, such that their local energy (given by Equation 4.2) is minimized. Notice that the energy of a floater is a measure of how close it is to other floaters. By minimizing this energy, floaters seek to be as far away from each other as possible, and they can only do this by placing themselves in a position approximately equidistant to all their neighbors. In all, this algorithm amounts to using local search to minimize the total energy of the floaters. In a flat surface, an optimal solution would be a uniform hexagonal grid.

When localized regions of the field change substantially, the above scheme will leave too many floaters in contracting regions, and too few in expanding ones. To solve this problem two measures are adopted: first, instead of using a constant, global value for $\sigma$, each floater is assigned an adaptive $\sigma_i$ that grows when the floater is in sparsely populated regions and shrinks when it is in crowded areas; second, floaters whose $\sigma_i$ grows too much are allowed to generate offspring, and where it has shrunk too much, they are allowed to die. This ensures an even distribution of floaters throughout the surface. The value of $\sigma_i$ is adaptively changed so that the floater's
energy is always near a fixed "optimal" energy. The change is carried out using simple binary search.

The positions of the floaters are used to construct a spatial subdivision. To do this, the positions of the floaters are projected to the domain. This has the effect of producing a spatial distribution where floater density is approximately proportional to field gradient, as illustrated in Figure 1.3. The subdivision is computed by dividing the space into simplices whose vertices are the floaters. The simplices need to be small and non-sharp, so a Delaunay triangulation is used. Such a triangulation offers the added advantage that it can be computed efficiently in two dimensions [17]. Also, this triangulation is a superset of the closest neighbor graph, and therefore it captures the notion of spatial locality: local variations in a floater's position cause changes in the triangulation that do not propagate beyond its immediate vicinity.

An improvement to Witkin and Heckbert's algorithm will now be introduced. First, notice that there is an explicit representation of a neighborhood relationship
between floaters, defined by the edges of the triangulation. Next, notice that the energy contribution of surrounding floaters decreases exponentially with the distance. Hence the calculation of the floater energy can be approximated considering only each floater's immediate neighbors. On average, this reduces the complexity of the energy computation algorithm from $O(n^2)$ to $O(n)$. This optimization extends to higher dimensions as long as there is an efficient means to construct a neighborhood relationship. By introducing an order of magnitude in reduction of the complexity of the algorithm, this optimization makes the operation of repeatedly updating the position of the floaters manageable.

In Figure 4.4 a snapshot of a Diffusion-Reaction field is shown next to a spatial subdivision (a Delaunay triangulation) built on a set of floaters that sample the field.
The triangulation follows the structure of the field. Note the higher density of floaters in regions of high gradient.

As the field varies in time, so does the position of the floaters. This, in turn, causes the spatial subdivision to change: some edges cease to exist and some new ones arise at every time step. However, given the assumption that the underlying field changes slowly, the vast majority of edges and triangles are preserved through successive time steps, even though their shape is slightly changed. Herein lies the main advantage of the approach being described: since most structures are preserved across time, the need to use statistical methods to perform correlations decreases substantially. Moreover, it is not necessary to reconstruct the spatial subdivision at every time step. It is sufficient to update the parts that have varied. Because of the local nature of the Delaunay triangulation, these updates do not propagate far.

A Simple Adaptive Terrain Irregular Network (TIN)

An alternate way to do adaptive sampling relies on a technique widely used to sample irregular terrains: for any particular region of space, add particles until the difference between the field values at any two adjacent particles falls below a certain threshold. If the field varies with time, this technique can be repeated iteratively, and extended to avoid an excessive number of particles: wherever deleting a particle would not cause the previous condition to be violated, do so.

One clear difference between this approach and the previous one is that here particles do not move: the accuracy of sampling is achieved not by re-arrangement, but only by insertion and deletion until desired densities of sampling are reached.

This method requires the ability to perform a single query on the field: given two points of it, do they belong to the same class? This is the same query that
Adaptive TIN algorithm

- Initialization step: place particles in a uniform grid and construct a Delaunay triangulation.
- Decimation step: On each iteration, look at all particles $p$. If all neighbors of $p$ belong to the same class as $p$, delete $p$.
- Population step: On each iteration, look at all triangles $t$. If none of the sides of $t$ is smaller than a threshold, and the three vertices and the center of gravity of $t$ do not belong to the same class, insert a new particle at the center of $t$.

Table 1.1: A simple algorithm to maintain a sampling particle system in a changing field

the classifier has to make when deciding whether two particles should belong to the same partition. The algorithm then proceeds in two steps: a decimation step and a population step. Details are on Table 1.1.

In Figure 1.5 the snapshot of Figure 1.4 is repeated for comparison. It is shown next to a Delaunay triangulation built on a particle system that samples the field.

The system was constructed following the simple algorithm in Table 1.1

Convergence Rates

We now examine experimentally how long it takes for the two methods described above to reach a stable particle configuration for a fixed field state.

The experiment is set as follows: for a fixed field configuration, 625 floaters are arranged in an hexagonal grid. For each of the two algorithms above described, to which we will refer as MWH and ATIN (short for Modified Witkin-Heckbert and Adaptive TIN) from now on, the floaters are allowed to reach a stable state, that is, a state where no floaters move, are born or die. We measure, per iteration, the
number of floaters that move causing structural change, the number of newly inserted floaters and the number of deceased floaters. All these measurements are made for a fixed set of floater parameters, making these parameters have the same value for both algorithms whenever they are relevant to both.

Figures 4.6 and 4.7 show the behavior of Algorithm MWH, the modification of Witkin and Heckbert's. Convergence towards fixed floater positions is slow, and in many cases it is never reached. Some floaters oscillate in place.

Figures 4.8 and 4.9 show the behavior of Algorithm ATIN. Ironically, this much simpler algorithm converges considerably faster than Algorithm MWH, and invariably reaches true equilibrium.

While the above tests indicate how the algorithms behave when sampling a field that is on a fixed state, they do not give any indication of how they behave when
Figure 1.6: Starting from a state far from equilibrium, the population of floaters under Algorithm MWH grows slowly until it converges on a fixed value.

Figure 1.7: The fraction of floaters that move, are born or die is plotted for Algorithm MWH. When far from equilibrium, many floaters are quickly moved and inserted in an attempt to reach stability. Deletions are always few, because the algorithm seldom overshoots when inserting floaters. After the transient, insertions and movement to not converge to zero, but instead remain bounded below a small value.
Figure 4.8: Starting from a state far from equilibrium, the population of floaters under Algorithm ATIN converges very quickly to a fixed value.

Figure 4.9: The fraction of floaters that move, are born or die is plotted for Algorithm ATIN. When far from equilibrium, many floaters are quickly inserted; equilibrium is reached as soon as all the field is sufficiently sampled. After a short transient, the system reaches true equilibrium.
Figure 4.10: Starting at near equilibrium, the population of floaters under Algorithm MWH adjusts itself to changes in the field.

sampling a changing field. The experiments above are repeated by allowing the field to evolve in parallel with the floaters after they have reached equilibrium. Clearly the behavior of the algorithms will depend on how fast the field evolves with respect to the floaters. A configuration is chosen where the field varies continuously without ever reaching a fixed configuration.

Figures 4.10 and 4.11 show the behavior of Algorithm MWH, on a changing field, a still of which is shown in Figure 4.12. In the initial state of the field, the regions of high gradient are small. In the approximately 200 iterations plotted, these regions grow, and then shrink again. Algorithm MWH adjusts the population of floaters quickly, but it is not eager to let floaters die off when they are no longer needed. Instead, these floaters arrange themselves into a quasi-uniform grid to minimize their local energies. The fraction of floaters that move grows substantially around the middle of the run, when many abrupt events take place. Insertions and deletions follow the same pattern, but to a much lesser degrees.
Figure 1.11: The fraction of floater events increases substantially during times in the field evolution when abrupt changes take place. Around the middle of this run, many field clusters grow quickly and then die off.

Figure 1.12: Still of field used to test floater algorithms
Figure 4.13: The population of floaters adjusts itself to the sampling needs of the field as it evolves for Algorithm ATIN. Unneeded floaters do not linger, but are destroyed quickly.

The behavior of Algorithm MWH is sensitive to its various parameters. The higher the local energy that is set as optimal, the higher the floater density, and also the lower the stability of the system. The density and stability of the particle system also depend on the thresholds that determine when to split and when to kill each floater. If these thresholds are too tight, few floaters can remain for any non-trivial amount of time within these thresholds, so stability becomes low. If they are too loose, floaters do not adapt their density fast enough as the field changes. The parameters were tuned by trial and error, to reach an approximate average density of 2000 floaters per square unit. The values used were 0.2 for the optimal energy, and 0.01 and 0.04 for the kill and split thresholds, respectively.

Figure 4.13 and 4.14 show the behavior of Algorithm ATIN on the same field shown in Figure 4.12. Unlike Algorithm MWH, the sampling of the field is always
Floater behavior approaching stability

![Graph showing fraction of movement, insertions, and deletions over iterations.]

Figure 4.14: Insertions and deletions exhibit a strong direct correlation for Algorithm ATIN during a run, and both match closely the field changes.

adjusted quickly. New floaters are added when needed, but they are destroyed just as fast when no longer needed.

The behavior of Algorithm ATIN is regulated only by one parameter: the desired maximum floater density, which is specified explicitly as the minimum distance that is allowed for any two floaters. Furthermore, this parameter has very little impact on system stability; when set too small, it only causes the convergence time to increase slightly, simply because more time is needed to insert all the necessary floaters.

The two algorithms presented above each have advantages and disadvantages. The first algorithm (MWH) is field-driven: floaters expand, like molecules in a gas, trying to occupy all available space without overcrowding any region. The second algorithm (ATIN) is mesh-driven: particles are introduced or removed depending on the appropriateness of the existing set of triangles. The first algorithm is noisier and slower to converge than the second, but it has one distinct advantage: because the second is mesh-driven, it can only fix an existing mesh of floaters by inserting or
deleting floaters from *inside* the mesh. The convex hull of the floaters may decrease in extension, but it may never increase. For this reason, to ensure that all regions of the field are always sampled, the second algorithm requires an ad-hoc addition: permanent floaters need to be fixed at the extreme points, so the convex hull of the mesh is ensured to coincide with the full extension of the field. This approach may be inconvenient for field geometries without borders, such as spheres.

The field used in the experiments explained above changes slow enough so that Algorithm ATIN can adapt very quickly to the changes. For a rapidly changing field, it is hard to tell whether Algorithm ATIN is inherently more stable than Algorithm MWH. For this reason and the above, there is no straightforward answer to the question of which algorithm is better in general.

**Description Through Iso-Surfaces**

Cluster boundaries are associated with field regions of high gradient. Those regions can be identified using iso-surfaces, continuous zones of uniform or almost uniform field value. The ratio of field value change to the distance between iso-surfaces gives an estimation of the gradient. Therefore, a field that is characterized by near-uniform regions that vary smoothly is well described by iso-surfaces that sample evenly-spaced field values. All the problems described before—fluid dynamics analysis, weather maps, Diffusion-Reaction fields— are well suited to description through iso-surfaces. Temporal variations in fields will be studied through the examination of geometric and topological change in iso-surfaces.

The field will be divided into regions whose size corresponds to the degree of uniformity of the field: wherever the gradient is high, regions should be small, and wherever the gradient is small, regions should be large. This results in a sampling
resolution that varies through space, high only where it is needed. Regions will later be aggregated to form bodies whose boundaries approximate iso-surfaces for high-gradient regions. This subdivision will be re-usable, because with minimal re-computing efforts, it can be updated to reflect changes in the field. For this reason, the components of the subdivision should be persistent, as they will allow to identify time-varying spatial structures in the field.

The particle placement algorithms previously described have the desired properties. These algorithms require the ability to do two things: to determine class equivalence between particles, and to evaluate distances between particles that take the field into consideration. Since this case study is of two-dimensional diffusion-reaction systems, the distances between floaters can be computed by using the field values as additional dimensions, with an appropriate scaling constant. Class equivalence for adjacent particles is computed by thresholding the distance between the particles in feature space, that is, by considering only the values of their properties.

The extraction of structures from the spatial subdivision is analogous to a pixel-based region growth algorithm, with the difference that the element of aggregation is not the pixel, but the triangle. More precisely, instead of using pixel values as the criterion to aggregate elements, the average field value inside each triangle is used. In Figure 4.15 the result of carrying out this process for the snapshot in Figure 4.1 is shown. Because only two classes of structures are present, only one kind is outlined.

Notice that the polygonal representations of the structures are somewhat lacking in smoothness. This is due to the fact that the overall density of floaters in the example is rather coarse. However, considering the relatively low pixel resolution in
the underlying field, this polygonal representation is adequate, as it approximates the irregular shape of an iso-line surrounding the blue bodies.

Thresholding

As seen above, polygon borders approximate field iso-lines. This opens the question of how to choose an appropriate threshold. Varying the threshold value, it is possible to extract several iso-linear representations from a single spatial subdivision, so one of these representations has to be chosen above the others.

How to define a good representation based on iso-lines? This problem is somewhat alleviated if different regions in the field are relatively homogeneous, and noticeably distinct from regions of a different class. This seems to be the case for the Gray-Scott model (defined in Section 1.1), because it typically exhibits regions that are sharply different from each other. Given that, it is to be expected that no significant variation
Figure 4.16: As a threshold value is varied in stills from a Gray-Scott system, the fraction of the field whose value is above the threshold varies. It decreases very quickly for a near-initial state of the field, and more smoothly for steady state.

To test the above hypothesis, two stills of an evolving field are taken, corresponding to a near-initial state (consisting of a square blue perturbation in the middle of an otherwise entirely red field) and a steady state. The threshold value is varied from 0 to 0.1 (the range of the values for the second, "blue" compound), and the fraction of pixels whose field value is greater than each threshold is measured. Figure 4.16 shows the result of this experiment. For the initial state, almost any non-zero threshold value in the range will produce almost the same iso-lines. However, the steady-state field shows a smoother change. Notice that the curve drops very sharply at first, about half of its total range when the threshold varies from 0 to 0.05, barely 12.5% of its range. The next 25% is very slow, requiring the threshold to change from 0.05 to 0.20, which constitutes 37.5% of the entire threshold range. Hence, it is expected
that threshold changes within this range will result in relatively small changes in
the iso-linear representation. This assumption is verified by taking a fixed floater
configuration in the steady-state field still, and by drawing polygonal isolines for
various threshold values within this range. The results are shown in Figure 1.17.

As expected, within the near flat range, the sensitivity to threshold variation is
small. Henceforth a value in the middle of the range, 0.15, will be used for the Gray-
Scott diffusion-reaction system. The discovery of this threshold could be automated,
but it would require the system to perform a "trial" run, when the field is allowed to
change without carrying out any sampling, and the field values are observed to draw
a graph such as the one shown in Figure 1.17.

The previously described experiment is repeated for a Lotka-Volterra system and
a Brusselorator, both defined in Section 1.1. Results are shown in Figures 1.18 and 1.19,
respectively. The sharp decrease exhibited by the Gray-Scott system is also exhibited
by these two other systems, so the same relative robustness to threshold values should
be expected from them. The threshold values used for these systems, respectively,
are 0.3 and 0.3.

Redescription Algorithm

The re-description of a connected group of triangles as a single polygonal body is
of critical importance to achieve topologically correct super-structures. The algorithm
to achieve this is straightforward: start with a set of triangles and proceed as shown
in Table 1.2.

Notice that the above algorithm is applicable to higher dimension spaces. How-
ever, two dimensional objects have one-dimensional borders, and one-dimensional
objects are inherently well-ordered. The existence of such an intrinsic order makes
Figure 1.17: Four iso-linear contours for different threshold values. (a) corresponds to the beginning of a near-flat region in threshold-variation; the value is still small enough for nearby objects not to be differentiated. However, substantial changes in the threshold above this value (b through d) cause little further change in the contours.
Figure 4.18: As a threshold value is varied in a Lotka-Volterra system still, the fraction of the field whose value is above the threshold decreases sharply, and then has a long, nearly flat plain.

Figure 4.19: As a threshold value is varied in a Brusselator still, the fraction of the field whose value is above the threshold decreases sharply, and then has a long, nearly flat plain.
INPUT: List of adjacent triangles
   initialize polygon P as empty
   for every triangle t in the list
       remove from P those edges that are also in t
       insert into P the remaining edges of t
   end for
   identify all connected regions in P, labeling them as separate borders
   output P

Table 4.2: A procedure to re-describe a set of adjacent triangles as a polygonal body

the implementation of this algorithm very efficient for the two-dimensional case, as it can be handled by means of cut-and-paste string operations. Also, the above algorithm identifies all borders in objects. An object with more than one border has hollow regions in it. Although this representation exhibits no significant topological advantage over, say, a stick diagram, it also approximates geometric information such as volumes and surfaces.

Dealing with Time

As time elapses and the field changes, floaters adjust their positions, and sometimes they split or die, to maintain an accurate reflection of the field structure. Change is slow, since most particles either stay in place or change their position slightly (in the first of the two algorithms considered). Correspondence through time is almost always straightforward: immediate for floaters, since their identities are persistent, and recursively constructible for any elements that are recomputed at every generation; this is true for both floater algorithms. Therefore any brute-force approach that simply creates a Delaunay triangulation from scratch at every time
step could draw correspondences across generations for edges, triangles and triangle
groups. For instance, for line segments, we would proceed as follows. For time steps
\( t \) and \( t + 1 \) and floaters \( p \) and \( q \), the following cases are considered:

- if at \( t \) a line segment \( s \) joined floaters \( p \) and \( q \) and at \( t + 1 \) a segment \( s' \) joins \( p \)
  and \( q \), we identify \( s \) and \( s' \);

- if at \( t \) a line segment \( s \) joined \( p \) and \( q \) and at \( t + 1 \) it happens that \( p \) and \( q \) are
  not adjacent, we say that segment \( s \) died at time \( t + 1 \):

- if at \( t \) it is the case that \( p \) and \( q \) are not adjacent and at \( t + 1 \) a segment \( s' \) joins
  \( p \) and \( q \), we say that segment \( s' \) was born at time \( t + 1 \).

Triangles can be tracked in a similar fashion. However, the concept of identity for
polygons must be relaxed, to allow for minor alterations. Two polygons that exist at
successive time steps are identified if the identity of most of their components (the
triangles or the floaters inside of the polygon) has been established.

Notice that if instead of recomputing the triangulation at every time step it is
simply repaired wherever floater changes force structural alterations, then edges and
triangles that do not change keep their identity, and there is no need to draw corre-
spondences explicitly.

The same would be true of polygons, except that they are not encoded in the
triangulation, so they need to be named explicitly. Therefore, it is necessary to
carry out computation specifically aimed at tracking events that involve aggregated
polygons. Whether the triangulation is recomputed at every time step or it is updated
intelligently, two alternatives are left as far as polygonal aggregates are concerned: (1)
to recompute them from scratch using the triangulation and the field, to later infer
Recursive Tracking of Non-Persistent Polygons

Let us first consider the case where polygonal aggregates are computed from scratch at every time step. To generate a history of relevant events, an efficient method to record change in the elements (edges and triangles) is needed. This is achieved as follows:

An overall history of triangulation elements is initialized by registering all elements during the first time step. From that point on, at every update step, an update list is generated. Then, this list is run through to annotate changes: elements that are destroyed are annotated as dead in the overall list, while those that are created are inserted as new. In each case, a death or birth time stamp is recorded. This results in a comprehensive compendium of transformation events, which can be indexed either spatially (by element location) or temporally (by birth or death date).

The above idea works well for edges and triangles, but it must be extended for aggregated polygons. It is necessary to preserve identities even when the polygons suffer structural change, namely the loss or gain of a few triangles or border edges. This only requires to elaborate on the comparison algorithm used to identify entries in the global history. For edges, the identity of the two endpoints was used, and for triangles, the identity of the three vertices was required. In contrast, two polygons will be said to be identified if it is possible to establish identification among the majority of their elements.

However, there are more events that need to be recorded in a history of aggregated objects:
• Non-catastrophic events of type I: changes of internal structure. Objects may gain or lose holes. These events are considerably simpler to notice in two-dimensions than shape changes, because they involve clear, discrete changes in structure. In higher dimensions, however, there are considerably more cases in need of examination.

• Non-catastrophic events of type II: changes of geometric properties. Objects may suffer elongations or other distortions in shape, without necessarily suffering any significant structural change. Events of this kind may be noted first by simple heuristic detection (such as noticeable changes in area, enclosing rectangle, etc.), or by examining alterations in geometric relationships among border elements. This requires the use of a mechanism that allows the computation of a “distance” between objects that depends on their geometric properties. By forming clusters of geometrically similar objects in distance space, it will be established whether there have been qualitative shifts in shape when an object moves between clusters. This is a complex procedure, which will be examined in detail in the next section.

• Catastrophic changes in structure: objects may collide with other objects, or split into several smaller objects. Such events will be annotated in the history as death events followed by birth events. However, a subtler record can also be established by examining what happens to the components of a dying object: if most of them are included in a new object then collision is inferred, and if most of them are separated into disjoint sets, fission is inferred.
Figure 4.20: A fission transformation. Stills are approximately 100 generations apart.

See Table 4.3 for a comprehensive list of events that should be recorded in a history.

Figure 4.20 exhibits a sequence that represents a fission transformation. To formalize the above ideas, we adapt the definitions from section 3.2, where \( \alpha \) is a real such that \( 0 < \alpha < 1 \).

**Definition 4.2.1** Given finite sets \( S \) and \( U \) such that \( U \subseteq S \), we say that \( U \) is an \( \alpha \)-majority of \( S \) iff \( |U| > \alpha |S| \).

**Definition 4.2.2** Given two finite sets \( S_1 \) and \( S_2 \), we say that \( S_1 \) contains an \( \alpha \)-majority of \( S_2 \) iff there is a subset \( U \) of \( S_1 \) such that \( U \) is an \( \alpha \)-majority of \( S_2 \).

**Definition 4.2.3** Two finite sets \( S_1 \) and \( S_2 \) are said to be \( \alpha \)-identifiable iff \( S_1 \) contains an \( \alpha \)-majority of \( S_2 \) and \( S_2 \) contains an \( \alpha \)-majority of \( S_1 \).

**Definition 4.2.4** A set of sets \( \{S_1, S_2, \ldots, S_n\} \) is said to be an \( \alpha \)-decomposition of a finite set \( S \) iff \( S \) and \( \bigcup_{i=1}^{n} S_i \) are \( \alpha \)-identifiable.
<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>birth</td>
<td>a new object comes into existence ex nihilo</td>
</tr>
<tr>
<td>death</td>
<td>an object ceases to exist</td>
</tr>
<tr>
<td>fusion</td>
<td>one or more objects coalesce together to form a new object</td>
</tr>
<tr>
<td>fission</td>
<td>an object splits into several parts</td>
</tr>
<tr>
<td>assimilation</td>
<td>one or more objects fuse with and begin to form part of a larger object</td>
</tr>
<tr>
<td>separation</td>
<td>one or more small parts of a large object separate from it, giving rise to new objects</td>
</tr>
<tr>
<td>topological change</td>
<td>an object gains or loses holes</td>
</tr>
<tr>
<td>geometric change</td>
<td>an object suffers a substantial transformation in shape</td>
</tr>
</tbody>
</table>

Table 1.3: A comprehensive list of events that need to be recorded in a history whenever they occur.

These definitions allow us to establish various temporal correspondences among polygons given the sets of their embedded triangles or floaters: a parameter $\alpha$ is fixed, and then it is said that two polygons $P_1$ and $P_2$ consecutive in time are identifiable if their triangle or floater sets are $\alpha$-identifiable. Similarly, if a polygon $P$ is followed by several polygons whose triangles or floaters constitute an $\alpha$-decomposition of $P$, it is said that a fission has occurred, that is, unless one of the new polygons is $\alpha$-identifiable with $P$, in which case it is said that the other polygons have separated from $P$. If the above process is reversed in time, it is called a fusion or an assimilation, respectively.

Notice that to compute temporal correspondences between polygons is efficient, in the sense that it is not necessary to check every pair of polygons across time steps. The persistence of floaters and the use of maps from floaters to the objects that contain them allow the determination of which polygons to compare in almost constant time (or rather, the time it takes to query each map).
Intelligent Updates on Polygonal Aggregates

We now consider ways to keep track of polygons without having to recompute them from scratch at every time step. It is considerably simpler to achieve polygon updates if the underlying triangulation is also being updated. This is in fact the approach taken in this implementation. However, this low-level update is not strictly required: because it is possible to do recursive identification of elements, polygons can be updated based on consecutive full recomputations of the triangulation.

At initialization, the space is partitioned into regions of adjacent triangles. Two triangles $t_1$ and $t_2$ are placed in the same partition if they are adjacent to each other and they enclose field regions of similar values. Each class is redescribed as a polygon, computed by the procedure in Table 1.2.

At the first iteration, all associative classes (space partitions) of triangles are computed from scratch. From that point on, whenever the triangulation changes, a list of deleted triangles is extracted. The classes to which these triangles belong to are identified, and they are repaired using the newly formed triangles. Afterwards, the polygons associated with the affected classes are fixed if their borders are compromised. To determine this, a list of affected edges is extracted from the triangulation changes, namely those that are either destroyed or belonging to a triangle that has been created or destroyed. If a polygon contains edges on the affected list, it is considered to be compromised.

Notice that the removal and insertion of triangles can cause a class to split into disconnected regions, or two or more previously disconnected classes to fuse into a new class. Therefore these events are logged whenever they happen. As a result, a
history of polygonal transformations is generated. This list is extended with changes in topological and geometric features.

The above scheme makes use of the abstraction of change mechanisms provided by STA that borrow ideas from KDS. The details were explained in section 3.2, so they will not be repeated here.

The methodology presented in this section provides a simple way to track topological change. The next step is to combine this kind of tracking with geometric change, which will give a complete picture of all that is qualitatively observable about a time-varying field that exhibits patterns.

### 4.2.2 Shape detection and classification

In the previous section it was shown that one important step when tracking objects is to notice changes in their geometry. Therefore, the problem of finding how similar, or how different, two arbitrary two-dimensional objects are will now be addressed. This will permit us not only to determine through which different shapes an object evolves, but also to perform higher level queries, such as how many distinct shapes show up in a particular diffusion-reaction system, and what are typical sequences of shapes.

**The problem**

One simple way to deal with the problem of detecting geometric change is the use of simple heuristics, as explained previously. A simple test, such as noticing large changes in area or extension, will always be sufficient to infer a geometric change (if the result of the test is positive). However, such tests will often fail to notice other sorts of geometric changes that are too subtle for such simple heuristics.
It is therefore necessary to introduce a way to quantify the difference between geometric shapes. Such quantification should be both rotationally and translationally invariant, and should be insensitive to noise and very low-scale variations.

There are a number of methods available to address this problem. However, typical computational approaches have focused on shape matching as a process of comparison to previously predefined templates selected from a library. Other methods compare angular deviations among the medial axes of objects. None of these approaches model well the human recognition process. People recognize objects based on their contour, rather than their skeleton, and they are also capable of recognizing similarity even among objects that have never been seen before. For instance, people can recognize similarities among the silhouettes of objects such as clouds or leaves, even if these shapes are unique. For this reason, we choose to focus on contour-based shape matching approaches.

One method that has been extensively developed by other researchers uses fixed curvature smoothing and tuning, curve segmentation and curve matching to measure the differences between the bi-dimensional shapes of leaves from various plant species [9]. We would like to examine closely how well their approaches perform, and determine how they have to be modified in order to accommodate to the purposes of this thesis. Next these methods are introduced, along with the modifications we have made on them.

The Curvature-Tuned Smoothing Method

The concept of shape is ill-defined. However, evidence from psychology [10] and computer vision [25] suggests that a fundamental task for the identification and modelling of shape is the estimation of curvature. We are interested in a method that
would allow the automatic classification of a large number of curves according to their shapes. If it were possible to measure the difference between the shapes of two arbitrary curves as a single number, independently of rotation or scale, then similar curves could be clustered into equivalence classes.

The curvature-tuned smoothing method [9] goes very far in this direction. Formally, it is formulated as follows: Given a parametric curve \( d(t) \) sampled as a sequence of points \( \{d_i\}_n \), a new set of points \( \{u_i\}_n \) is computed such that the expression

\[
E(u) = \sum_{i=1}^{n} (u_i - d_i)^2 + \lambda s(u_i) c^2
\]

is minimized, where \( \lambda \) and \( c \) are constants, and \( s(u_i) \) is the curvature of the curve approximated by the new points at point \( i \). By minimizing this expression, the new points are made to be as close as possible to the original ones, while at the same time making the curvature approach a fixed value \( c \). There is a tradeoff between both goals, and constant \( \lambda \) determines their relative importance. Figure 4:21 shows an example of curvature smoothing for two different values of constant \( c \).

**Curve Segmentation**

The minimization of the \( E(u) \) function produces a curve that is distinct from the original one. The curvature at each point of the new curve is usually nearer the target curvature than the curvature of its corresponding point in the input (unless the input already was smooth, of course). However, because there is a tradeoff between how much the output can deviate from the original curve and how near it gets to the target curvature, this target is seldom reached. However, whenever some regions of the input curve will be naturally described by curves of curvature very near to the
Figure 4.21: A closed curve and the result of minimizing equation 4.3. (a) the input curve; notice that it is not smooth; (b) output after minimizing with $c = 0.02$; the parts of the curve where the curvature is positive in the large scale are well approximated, while concave regions are not; (c) output after minimizing with $c = -0.1$; the opposite happens. $\lambda$ is 1000 in both cases.

target, others will not. Some points will be very poorly fitted to this target; these are called high-stress points. See Figure 4.22 for an example.

It is desirable to segment the curve into regions that are consistently well fitted to a target curvature. It is natural, then, to break it at the high stress points. The number of such points depends on how well the target curvature fits the overall curve. A simple thresholding method is used: for a constant $\delta > 1$, at each point $u_i$ an acceptable range is defined for the value of curvature $\kappa(u_i)$ given the target curvature $c$. This range is $\left[\frac{1}{\delta}, \delta c\right]$ if $c > 0$, and $\left[\delta c, \frac{1}{\delta}\right]$ if $c < 0$. Those points whose curvature falls outside of the acceptable range are considered to be of high stress.

**Extracting Multi-Curvature Descriptions**

Fixed-curvature segmentations for two different values of $c$ that are very close to each other should be similar, because both values will produce well-fit and ill-fit regions in approximately the same places. Therefore it is possible to compare those
Figure 4.22: Curvature stress (computed as \((\kappa(u_i) - c)^2\) for each fitted point of index \(i\)) varies considerably along the curve shown in Figure 4.21(a). When the curve is well adjusted to the global fitting curvature (a), the upper bound of the stress is small and the variations are regular and predictable from the contour of the curve. When the curve is ill-fitted to the global value (b), the stress varies widely and in a seemingly random manner.

Two segmentations by looking at corresponding segments and determining which one has the smallest average value for \((\kappa(u_i) - c)^2\). If it is possible to estimate the range of variation for the curvature in the input curve, several minimizations of \(E(u)\) can be performed for various values of \(c\) within that range. Those segments that best fit the curve at each region are chosen. The algorithm is shown in Table 4.1 in simplified pseudocode form.

The algorithm prefers long segments over short ones whenever the shorter segment would be fully covered by the longer one. On the other hand, segments that span a region not covered by any single other segment are left in place. However, some of those segments are removed in a post-processing step as follows: for every segment \(k\) such that \(k\) is fully overlapped by the combination of other segments, if every segment \(k'\) that has a region in common with \(k\) has less curvature stress that \(k\) within that common region, remove \(k\).
INPUT: Set $S = \{ s_j \}_{j=1}^n$ of segmentations, each corresponding to a curve, and each consisting of a sequence of segments.
OUTPUT: An ordered set of segments

Procedure ChooseSegments(Set S)

   Initialize ordered set $T$ of segments to the empty set:
   for each member $s_j$ of $S$
      for each segment $k_i$ of segmentation $s_j$
         repeat
            find segments $k_{i^\prime}$ and $k_{i^\prime\prime}$ in $T$ that are ordinarily immediately before and after $k_i$;
            if neither of these two segments is fully covered or neither of these two segments fully covers $k_i$, then add $k_i$ to $T$;
            else for each segment
               if segment is fully covered by $k_i$
                  replace $k_i$ for segment in $T$;
               else
                  $k_i$ is fully covered by segment; ignore $k_i$;
            until neither surrounding segment overlaps $k_i$;
         return $T$
   end ChooseSegments

Table 4.1: An algorithm to extract a multiple-curvature segmentation of a curve given a set of fixed-curvature segmentations
I'm interested in various fixed-curvature segmentations so that the deviation from their target curvature is minimized.

In Figure 4.23 the result of applying the multiple-curvature segmentation algorithm to the input of Figure 4.21(a) is shown. Notice that some of the output segments overlap in their spans.

Matching

We now examine how to quantify the difference between the multiple-curvature segmentations of two curves. For this, we first introduce the Smith-Waterman algorithm [31], and then we modify it to suit the purposes of this thesis. Notice that the originators of the curvature-tuned smoothing method use an un-modified version of the Smith-Waterman algorithm: they are not concerned with dealing with redundancy.

Let $S_1$ and $S_2$ be two strings of lengths $n$ and $m$ respectively. It is desired to align the strings so that a maximum number of elements are matched. For example, consider the two strings “CAGCCTGGATTCTCGG” and “CAGCGTGG”. An optimal alignment of these sequences is:

```
CAGCCTGGATTCTCGG
CAGC-----G-T-----GG
```
To find such alignments a dynamic programming method is used. A cost function is defined so that it evaluates at each pair as follows: if the pair is a match, there is no cost; if the pair is a mismatch, the cost is 1, and whenever an element on either string has been skipped (represented above by the "-" character) the cost is 2 (these particular values are arbitrary; what matters is that the skipping penalty be twice the mismatch penalty). Define a table $C$ indexed from 0 to $n$ horizontally and from 0 to $m$ vertically, and initialize the first row and first column as follows:

\[
C(i, 0) = 2, i = 0 \ldots n; \quad (4.1)
\]
\[
C(0, j) = 2, j = 0 \ldots m. \quad (4.5)
\]

The rest of the table is filled sweeping it by columns (or by rows) in order, as follows:

\[
C(i, j) = \min \left\{ \begin{array}{ll}
C(i - 1, j) + 2 & \\
C(i, j - 1) + 2 & \\
C(i - 1, j - 1) + \left\{ \begin{array}{ll}
0 & \text{if } S_1(i) = S_2(j) \\
1 & \text{otherwise}
\end{array} \right. &
\end{array} \right. \quad (4.6)
\]

The cost of the optimal matching is given by the value of $C(n, m)$, and the alignment can be extracted moving backwards from that cell depending on the decisions made by each evaluation of equation 4.6.

This algorithm can be understood as follows: each cell $(i, j)$ in the table represents a partial match of the first $i$ elements of the first string with the first $j$ elements of the second string. A path from cell $(0, 0)$ to cell $(i, j)$ establishes exactly how these prefixes are aligned. The value stored at cell $(i, j)$ represents the optimal cost of matching these two prefixes. Each iteration of the algorithm decides which move achieves the best cost for a new pair of prefixes, until both strings are consumed. At that point, the minimal cost for the entire match is stored at cell $(n, m)$. 

\[ \text{113} \]
When the strings to be matched are circular, it is necessary to be able to "wrap around" after the end of each string. Punishing a lack of global alignment should be avoided, because such an alignment only has significance when the strings have definite beginnings and ends. To achieve this, the algorithm is modified as follows:

First, a new string \( S'_2 \) is generated as two consecutive copies of \( S_2 \). \( S_1 \) will be matched with \( S'_2 \), and the cost table will have \( 2m \) rows. To avoid the cost associated with global alignment, all the elements of the first column are set to 0, and then the algorithm is allowed to proceed as usual. The optimal cost is not the last element of the cell, but the minimum chosen from the cells that range from \( C(n, m+1) \) to \( C(n, 2m) \).

The algorithm as presented above recognizes only two possible values for element alignments: match and mismatch. If the specification of the problem includes a definition of distance between different elements, that definition is used instead, taking care of adjusting the skipping cost so that it is always more expensive to skip an element than to align it with another. The first row and column should be filled with the progressive costs of skipping prefixes of each sequence.

To apply the above mechanism to multi-curvature segmentations, a cost function for any pair of segments is required, as well as the cost of skipping an arbitrary segment. The original curvature-tuned smoothing method defines the cost of matching segments \( S_1 \) and \( S_2 \) of lengths \( l_1 \) and \( l_2 \) and fitting curvatures \( c_1 \) and \( c_2 \), as

\[
(S_1, S_2) = |l_1 - l_2| + |sgn(c_1)\log|c_1| - sgn(c_2)\log|c_2||. \tag{4.7}
\]

However, this approach places a definite balance between the cost of matching lengths and the cost of matching curvatures. Their relative importance is determined by the base of the logarithm. This tradeoff can be decreased by using an alternate
Additionally, the cost of skipping a segment is defined as twice the value of its length.

**Considering Redundancy**

As seen previously, redundant segments (those whose covered regions of the curve are better represented by other segments) are removed. This, however, still leaves regions covered by more than one segment, as seen in Figure 4.23. How to lessen the cost of skipping a segment that has regions of redundancy? It would seem that an obvious way to do that is simply to consider only the non-redundant length of the segment for its skipping cost, but that would have the undesirable effect of making it very cheap to skip a long sequence of mutually redundant segments. Instead, we define the *uncovered length* of a segment as the length of its region that is not covered exclusively by its ordinal predecessor. The cost of skipping becomes twice the uncovered length. This way, the cost of skipping a sequence of segments is the same as the cost of skipping a single long segment covering the same region as the sequence.

A more interesting problem arises from the fact that redundant segments that are mostly covered by other segments may not have a corresponding segment at all in the segmentation of a slightly distorted version of the curve. This is illustrated in Figure 4.24, where each segmentation has a segment for which there is no corresponding segment in the other segmentation. If the Smith-Waterman algorithm as
Figure 4.24: Multiple-curvature segmentations of two very similar, but not identical, curves: the second curve was obtained by adding random perturbations to the sampling points of the first. Segment 3 of (a) has no corresponding segment in (b), and segment 7 of (b) has no corresponding segment in (a). However, both segments are highly redundant, and an alignment that skips them should not suffer a heavy penalization.

defined so far were applied to these two sequences, the alignment cost would be excessively high, because of the penalty paid for skipping the two segments. Applying the redundancy-reducing described above would help, but it would not be enough: a very large portion of segment 3 in Figure 4.24(a) is not covered by segment 2.

The solution to this redundancy problem is multifold. First of all, for each segment, its covered part is differentiated from its uncovered part. The covered part is the region that overlaps previous segments in the curve (where the “previous” relationship depends on the order given by the matching process). The uncovered part is what remains. With these concepts, the distance function is adjusted as follows: for segment $i$, let $l_c$ be the length of the covered part, and $l_u$, that of the uncovered
part. Then the matching cost for two segments $S_1$ and $S_2$ of target curvatures $c_1$ and $c_2$ is:

$$\langle S_1, S_2 \rangle = t |l c_1 - l c_2| + |l a_1 - l a_2||11 + |\text{sgn}(c_1) \log |c_1| - \text{sgn}(c_2) \log |c_2|\rangle. \quad (4.9)$$

On the other hand, the skipping cost for a segment is simply twice its uncovered length. This way, this cost only arises from skipping a part of a segment that corresponds to a section of the curve that is actually being skipped. Similarly, the distance functions will yield low costs if, in addition to having similar lengths, the segments have similar coverage of the curve.

The next idea to be introduced is that of reward, a value that is deducted from the cost. The purpose of rewards is to account for covered parts of a segment *a posteriori*. More explicitly, if a previous segment has an uncovered region that is covered by the current segment, then the current segment is rewarded for this. The reward is twice the length of the covered region. The operations $\text{reward}_1(i)$ and $\text{reward}_2(j)$ are thus defined as the reward cost for segments $i$ or $j$ for covering up any of the previous segments, either in curve 1 or curve 2. The cost function is now defined as

$$C(i, j) = \min \left\{ \begin{array}{l} C(i - 1, j) + \text{skip}(i) - \text{reward}_1(i) \\ C(i, j - 1) + \text{skip}(j) - \text{reward}_2(j) \\ C(i - 1, j - 1) + \text{match}(i, j) - \text{reward}_1(i) - \text{reward}_2(j) \end{array} \right\}. \quad (4.10)$$

The first row is defined as the increasing cost of skipping segments from the first curve. The first column is defined similarly unless the curves are closed, in which case it is initialized to zeroes; that way, there is no penalty for starting matching anywhere in the curve. However, if that is the case, a post-processing step is needed to ensure that the cell in the last column chosen to represent the final cost corresponds to a full match, that no part of the second curve is left unmatched. For that, the algorithm
determines whether any segments of the second curve need to be matched, and adds to the value of the cell the cost of skipping those segments.

**Achieving Scale and Sampling Invariance**

The cost of aligning multiple-curvature segmentations of curves is invariant to rotation, because the features used to match are local curvature and segment length, both features that do not depend on orientation. Other kinds of invariance are also desirable.

Because curves are represented using discrete sets of points (namely, the vertices of polygonal aggregates), the estimation of the curvature at each point depends not only on the global scale of the curve, but on the local density of the sampling points. Invariance to sampling density can be achieved by using the *uniform resampling* technique, which works as follows: for a sequence of points \( \{p_i\}_n \) that samples a curve, find a new set of points \( \{p'_i\}_m \) such that

- \( m \geq n \),
- there is a constant \( l \) such that for all \( i \) such that \( 1 \leq i \leq m \), the distance between points \( p'_{i-1} \) and \( p'_i \) is \( l \), and
- for each of the new points \( p'_i \) there are two successive points in the old set such that \( p'_i \) lies within the segment between these two points.

Although finding the uniform length \( l \) for the new sampling would be exceedingly difficult if done analytically, the problem is straightforward if done numerically: use simple binary search. Sampling invariance is therefore achieved by resampling all the curves to be compared using the same value of \( m \).
Scale invariance should only be pursued if one is strictly interested in comparing shape, and considers differences in size to be irrelevant. In such a case, the area covered by each curve is computed, and then each curve is re-scaled so they have unit area. Given fixed values for $\lambda$ and the curvatures $c$, to be used in the segmentation process, new values for these parameters are found that consider the density of the sampling. Because $l$ is inversely proportional to $m$, maintaining balance in Equation 4.3 requires adjusting the target curvature proportionally to $m$ and $\lambda$ in inverse proportion to $m^4$.

Notice that even if full scale invariance is not desired, the above method can still be used; an extra cost should later be added that depends exclusively on the scale difference between the curves that are being compared.

**Shape Clustering**

A method to compute the “distance” between two two-dimensional curves, either open or closed on themselves, has been developed. With the aid of regular Spatial Aggregation, this method permits the classification of a multitude of shapes into groups of similarity. To do this, the following steps are taken:

- A *Shape Space* is defined. This is a non-dimensional metric space whose objects are shapes, and where the distance between any two objects is computed using the algorithm above.

- A neighborhood graph is aggregated from Shape Space. It is necessary to use a Neighborhood Graph that can be computed solely using a distance function, and that captures well the notion of proximity. One such graph is the Minimum Spanning Tree, which is computed following Kruskal’s algorithm by sorting all
pairs of objects by their distance and adding edges to the graph in order starting with the shortest, avoiding those edges that would create cycles. Other graphs that could have been used are the $k$-nearest neighbors, or all neighbors within a fixed radius. The MST is advantageous over the others because it does not depend on any parameters ($k$ or the radius), it is simple and quick to compute, and it is a tree, which makes sure that every time an edge of the graph is cut, the graph is separated into two disjoint regions. This is useful when partitioning the space by discarding edges that are too long, below.

- A partition of the space is carried out by cutting edges from the MST that are too long. This method was originally proposed by Yip in a different context: bundling orbit groups in a phase space. An edge is said to be “too long” if it is longer than the average length of surrounding edges plus a number of standard deviations. The algorithm thus depends on two parameters: how many edges are used to compute the average length, and how many standard deviations are tolerated.

- Shape partitions are redescribed as abstract shapes by having each partition elect a representative shape. This is done by selecting an object whose average distance to all the other objects in the partition is minimal.

**Implementation and Experimental Results**

The shape clustering algorithm here developed is tested by selecting 25 shapes from a aggregated objects in a Gray-Scott diffusion-reaction system at random. The polygonal shapes are shown in Figure 4.25.
Figure 4.25: A set of 25 objects that will be classified using the modified multicurvature-description algorithm. These shapes have already been smoothed by an intermediate step of the algorithm.
Figure 4.26: The shapes from the previous Figure are classified into these clusters (shapes are not shown in proportional scales).

The algorithm is run with values of $\lambda = 100000$, target curvatures ranging from $0.0125$ to $1$, and from $-1$ to $-0.125$, and a scaling factor of $0.0001$. The bounding box of each object is approximately $0.1 \times 0.1$. As it can be seen in Figure 4.26, the algorithm does a good job at classifying the shapes.

After the classification step, each cluster is abstracted as a single shape by electing a representative, that is, a member of each cluster that best represents all members. These are shown in Figure 4.27.
The algorithm has been shown to work well at least in this limited case. In the next section the algorithm will be put to the test using varying diffusion-reaction systems.

4.2.3 Putting it All Together: Extracting Behavioral Descriptions

All the tools described so far allow to take a time-evolving diffusion-reaction system and to extract from it two things:

- a detailed history of significant events (from a qualitative perspective), including births, deaths, collisions and fusions of objects, and their changes in shape, specified as transitions from one shape cluster to another, and
• a summary of significant events that have taken place in the history, namely, which are the most common shapes, and which are the most common events. This is derived by counting how many times each particular transition takes place.

Part of the above information, namely simple catastrophic events, are generated in real time. The rest, which involves doing such things as computing shape classes and counting transitions, can only be done once the run is finished.

Implementation Summary

The implementation follows the same basic approach explained in the example of section 3.1, with some added details:

• At an pre-initialization step, floaters are allowed to sample a field correctly; they are allowed to run until they converge to a stable or quasi-stable state.

• A standard spatial aggregation layer is then computed. According to the region of the field they occupy, floaters are classified into different partitions, and each partition is redescribed as a high-level coherent object. A history is initialized.

• The field numerical integrator is started. At each time step, a layer of change detection is computed. Floaters are modified by either of the two algorithms described above, and the neighborhood graph (a Delaunay triangulation) is updated wherever necessary.

• Each update generates a list of changes, namely adjacencies and triangles that have been destroyed or created. Using this information, partitions are updated: destroyed objects are removed from the partitions that contained them, and
newly created objects are classified and added either to new or to existing partitions.

- The changes to each partition are examined to determine whether the redescriptions need to be updated or re-computed. Very often the partition will change, but the redescribed object will not. This is the case when the partition changes do not involve any edges of the polygon that represents the redescribed object.

- When redescribed objects do change, the change is examined to decide whether it merits to be logged in history as a qualitative change. All catastrophic changes (splits, fusions, etc.) qualify, and so do changes in internal structure, such as the acquisition or loss of holes. Geometric change is harder to detect, because it requires knowledge of the shape clusters, and this knowledge does not exist until the end of the run. Therefore, a conservative heuristic is adopted: when the area of an object changes beyond a very tight threshold, report a possible change. Of course, many of these reported possible changes will later be found to be no changes at all.

- The run is forced to terminate using simple criterion such as the passing of a fixed number of iterations or a certain period of simulated time. At this point, a log with all the raw events described events is generated as a file.

- A new application analyzes the log, extracting all recorded instances of objects as shapes, and classifying them into shape clusters. A new history is then generated, containing shape cluster transition events as well as catastrophic events, plus a summary of all shape clusters found and all the kinds of transitions observed.
The process described above generates a narrative that is human readable, and that can also be parsed by a simple visualization program that shows the representatives of each of the shape clusters found; this complements the textual narration with a visual indication of the transitions found during the run.

**Effect of KDS-like Techniques on Performance**

How does the use of selective update (KDS-like techniques) and the propagation of change impact the use of computational resources? To answer this question, several experiments are timed. In these experiments, a set of floaters is allowed to change. Two update methods are compared: a high-level brute force method, where the lower-level sampling construct (the Delaunay triangulation) is updated efficiently, but the high-level aggregates are recomputed at every time step; and the KDS-like mechanism described before. For simplicity, the underlying field (depicted in Figure 4.28, and corresponding to a typical configuration of a Diffusion-reaction field) is left fixed, and change in the floaters is forced by introducing random variation in point positions, and also by randomly inserting and destroying points. This randomness results in very poor sampling constructs, but it allows an empirical measurement of efficiency, without having to account for the additional computation overhead required to maintain a proper sampling by means of algorithms MWH or ATIN.

The positions of the particles are randomly changed using Gaussian noise. Also, with probabilities of 0.001, floaters are deleted or duplicated. The process starts with an hexagonal grid of 2500 particles, and the algorithm is run for 50 iterations. The time (in seconds) it takes to reach generation 50 for various values of the standard deviation of the noise is measured. The program was coded in C++, and it was run
on a Power Macintosh with a 604e processor running at 210 MHz. Figure 4.29 shows a plot of the results.

The curve labeled “No polygons” shows the time it takes the algorithm to update the triangulation only; no polygonal aggregates are computed. The curves labeled “polygon update” and “polygon rebuild” represent the condition-violation-based update and the brute force rebuild, respectively. The overhead curves represent the difference between these two last curves and the triangulation update curve.

For small values of the standard deviation of the noise, the polygonal update mechanism outperforms the brute-force method. However, for higher noise values, the situation is reversed. This is to be expected, because when the noise increases the number of things that need to be updated also increases. Beyond a certain point the number of required updates is higher than the number of operations required to simply reconstruct the polygonal aggregates.
Figure 4.29: A plot of the time it takes to compute 50 generations of 2500 particles whose position is randomly altered using Gaussian noise; the standard deviation of the noise is presented in the X axis.

In general, we should always expect to see a crossover point as the number of object that require update grows. When using brute force update, there is a clear upper bound on the number of operations per generation: the number of objects to be reconstructed. Notice that, in Figure 4.29, the rebuild overhead does not change substantially compared to the update overhead. On the other hand, the update cost has a much higher upper bound, which occurs when every particle in the system changes to the point they all cause updates. Because updates maintain consistency, multiple small updates on the same polygonal aggregate may cost more than the re-construction of that object from scratch.

Predicting the crossover point is difficult. It involves determining the average number of particles that, on average, cause structural changes in a Delaunay triangulation on a random distribution of points in the plane. If few points cause change.
then updates on the polygon should outperform the brute-force rebuild. A rough
estimate of how the number of changes relates to the standard deviation of the noise
can be obtained as follows: given that the field is on a unit square and that there
are 2500 particles, then the average length of all triangulation edges is roughly 0.02.
An hexagonal arrangement of particles with another particle at its center suffers a
transformation whenever the center particle moves more than half the size of the side
of the triangles, so it should be expected that about half of the particles will cause
changes for a standard deviation of the noise of 0.01. Hence, the crossover point
should be under 0.01. It can be seen from the graph that it is at 0.0033.

The actual relationship between the number of triangulation updates and the stan-
dard deviation can be determined experimentally. The result is shown in Figure 4.30.
The crossover point corresponds to a fraction of approximately 0.01; hence we can
say that the update algorithm is efficient when less that 10% of the vertices in the
triangulation move causing a structural change.

It is therefore advantageous to use the update mechanism presented here whenever
relatively small perturbations can be expected in the triangulation; this, in turn,
happens when the underlying field changes slowly enough. When the perturbations
are large, it may make more sense to simply recompute the higher level structures at
every time step.

4.2.4 Results

To exhibit the methods described in this chapter, a Lotka-Volterra system (see
Section 1.1) is allowed to run for several hundred generations. This is a rather short
run, but it is sufficient to show these methods at work. The captions in Figure 4.31
Effect of sigma on a randomly altered Delaunay triangulation

![Graph]

Figure 4.30: For 2500 particles whose position is randomly altered using Gaussian noise, with the standard deviation of the noise presented on the X axis, the Y axis shows the fraction of all particles that cause a structural change in the Delaunay triangulation due to their change in position.

show three moments of this short run. The triangulation is shown, as well as the borders of the aggregated bodies. The floater updates were done using Algorithm 2.

All raw events are recorded in a file, which is later post-processed to generate a history including shape transformations. This history is shown in Tables 4.5 and 4.6.

The textual history is incomplete, because it provides numerical identifiers for objects and clusters, without specifying what the objects and the abstract class shapes look like. For this reason, the same pre-processed file used to generate the textual history is passed through a visualization program, whose output is shown in Figures 4.32 through 4.41.

In the next section a further step is taken: several histories are aggregated together to form history classes.
Figure 4.31: A short run for a Lotka-Volterra System. It starts with single large body (a) that grows while a second body begins forming (b) until they collide, while a new body begins to form (c).

Figure 4.32: Representative shapes for the nine clusters referred to in Tables 4.5 and 4.6.
9 shape clusters detected

From step 295 body 1 was in shape cluster 8
At step 295 body 1 was born into cluster 8
From step 359 body 1 was in shape cluster 3
From step 373 body 1 was in shape cluster 8
From step 31 body 0 was in shape cluster 8
At step 31 body 0 was born into cluster 8
From step 192 body 0 was in shape cluster 7
From step 222 body 0 was in shape cluster 8
From step 247 body 0 was in shape cluster 6
From step 252 body 0 was in shape cluster 5
From step 309 body 0 was in shape cluster 4
At step 438 body 1 (born 295) fused into body 0
At step 438 body 2 arose from the fission of body 0 (born 31)
From step 439 body 0 was in shape cluster 2
At step 439 body 2 (born 438) fused into body 0
From step 454 body 0 was in shape cluster 1
From step 438 body 2 was in shape cluster 8
From step 458 body 3 was in shape cluster 8
At step 458 body 3 was born into cluster 8
From step 485 body 3 was in shape cluster 0
From step 491 body 3 was in shape cluster 8

Table 1.5: An automatically generated history for the Lotka-Volterra system depicted in Figure 1.31 (part 1). Events are grouped by body, and for each body, they are listed in chronological order.
Cluster 0 contains 1 objects.
Cluster 1 contains 3 objects.
Cluster 2 contains 3 objects.
Cluster 3 contains 1 objects.
Cluster 4 contains 4 objects.
Cluster 5 contains 1 objects.
Cluster 6 contains 1 objects.
Cluster 7 contains 1 objects.
Cluster 8 contains 29 objects.

11 sequences observed.

Sequence 0, 8 observed 1 time.
Sequence 2, 1 observed 1 time.
Sequence 3, 8 observed 1 time.
Sequence 4, 2 observed 1 time.
Sequence 5, 4 observed 1 time.
Sequence 6, 5 observed 1 time.
Sequence 7, 8 observed 1 time.
Sequence 8, 0 observed 1 time.
Sequence 8, 3 observed 1 time.
Sequence 8, 6 observed 1 time.
Sequence 8, 7 observed 1 time.

Table 4.6: An automatically generated history for the Lotka-Volterra system depicted in Figure 4.34 (part 2)
Figure 4.33: Cluster 1 has a single member. Its identifier is on its upper-left corner, and the iteration number at which it was discovered is at the upper-right corner.

Figure 4.34: Cluster 2 has three members.

Figure 4.35: Cluster 3 has three members.
Figure 4.36: Cluster 4 has a single member.

Figure 4.37: Cluster 5 has four members.
Figure 4.38: Cluster 6 has a single member.

Figure 4.39: Cluster 7 has a single member.
Figure 1.10: Cluster 8 has a single member.

Figure 1.11: Cluster 9 has 29 members. It is the most numerous group.
4.3 Behavior Classification

In the previous section a mechanism to extract textual narratives of qualitative events that happens to an evolving diffusion-reaction system. In this section we focus on a mechanism to compare several of these sequences in order to determine classes of qualitative behavior.

4.3.1 Generalized Histories

A generalized history is a set of transitional events, where the "before" and "after" objects of each transition are not concrete objects but abstract shapes. A generalized history is, therefore, a summary of characteristic events: which are the usual shapes observed, and how these shapes typically transform into one another. Notice that a generalized history is no longer a linear narrative, because the global sequentiality of time is removed from it.

Detection and classification of event sequences

Generalized histories are generated simply by detecting transitions in regular histories, and by abstracting away individual events. This is done after shape clusters have been detected, as follows: given that all object instances belong to a shape cluster, and all shape clusters are represented by an abstract shape: if object $X_i$ is followed by object $X_j$, and $X_i \in C_k$ and $X_j \in C_l$ where $C_k$ and $C_l$ are shape clusters, then an instance of the abstract transition $C_k \rightarrow C_l$ is registered. A generalized history is then a set of abstract shapes plus a set of abstract transitions.
4.3.2 Classification

A sensible classification scheme for generalized histories requires the ability to measure the similarity of any two such histories. With that ability, histories are treated as objects in a "history space," and they are aggregated into a history neighborhood graph which is later used to form partitions, each representing a history class, whose members are interpreted to have qualitatively similar behaviors.

Comparison Methods for Generalized Histories

The computation of the difference between two generalized histories has two components: comparison of the abstract shapes that arise in each history, and comparison of abstract transitions.

The first component involves aligning the generalized objects of the two histories and computing their distance in shape space; the average of all these distances is then computed. To align the shapes, each object from one history is compared to all the objects of the other; the one that produces minimal distance is selected.

The second component is computed by mapping the abstract transitions of one history into the other, using as a map the alignment of shapes used to compute the first component. This translations permits to count how many transitions are common to both histories. This quantity, when compared to the total number of transitions, provides a normalized metric.

Notice that the second component only makes sense when the alignment map is good, that is, when it is one-to-one or almost one-to-one and when the distances among aligned abstract shapes are small. If the distances are high, this map does not
provide a trustworthy translation; therefore it should only be used as a fine-tuning device, to differentiate otherwise similar histories.

Behavior aggregation based on similarity of histories

Once the distances between generalized histories can be computed, the histories are aggregated using a Minimum Spanning Tree as a neighborhood graph, in the same way that shapes were aggregated, and for the same reasons. Long edges are cut, and the remaining sub-trees are considered to be classes.

Experimental Results

Several histories were run for a Gray-Scott system with different parameter values. Captions for each of the systems are shown in Figures 4.12 through 4.15.

When the algorithm was run, the output in Table 4.7 was produced. It made no difference whether Algorithms 1 or 2 were used for the purpose of floater change.

The parameters of the aggregation algorithm were not tuned; instead, common values were used: MST edges longer than the average of surrounding edges plus 1.1 standard deviations were cut, and “surrounding” was defined as reachable within a depth of 3. It is apparent that the algorithm is somewhat overzealous in its emphasis on difference; there are more clusters than a human would probably find. On the other hand, those clusters that have more than one element do correspond to what people would intuitively point at as “similar.” In the next section we compare these results with those obtained by a human scientist observing the same phenomenon.

Evaluation

The results shown in this section fare well when compared with human observation. Pearson [24] classified patterns for the Gray-Scott system purely on the basis of his
Figure 4.42: Stills for system evolutions on a Gray-Scott system, part 1. Each was recorded as a generalized history. Histories were later classified into groups of similarity (see text).
Figure 4.43: Stills for system evolutions on a Gray-Scott system, part 2. Each was recorded as a generalized history. Histories were later classified into groups of similarity (see text).
Figure 4.44: Stills for system evolutions on a Gray-Scott system, part 3. Each was recorded as a generalized history. Histories were later classified into groups of similarity (see text).
Figure 1.45: Stills for system evolutions on a Gray-Scott system, part 1. Each was recorded as a generalized history. Histories were later classified into groups of similarity (see text).
30 history clusters found.
Cluster 1: (6 6.4)
Cluster 2: (6 6.2)
Cluster 3: (5.6 6.4)
Cluster 4: (5.6 6.2)
Cluster 5: (5.2 6.4)
Cluster 6: (5.2 6.2)
Cluster 7: (4.8 6.4)
Cluster 8: (4.8 6.2)
Cluster 9: (4.4 6.6)
Cluster 10: (4.4 6.4)
Cluster 11: (4.4 6.2)
Cluster 12: (4 6.2)
Cluster 13: (3.6 6.2)
Cluster 14: (3.2 6)
Cluster 15: (2.8 6.4)
Cluster 16: (4.4 6) (3.6 5.8) (2.8 5.4)
Cluster 17: (3.6 6.4) (3.2 6.4) (3.6 2.6.2) (2.8 6.2) (2.8 6) (2.4 6.2)
(2.4 5.6)
Cluster 18: (2.4 5.4)
Cluster 19: (4.8 6) (3.2 5.6) (2.4 5.2)
Cluster 20: (4 6) (4 5.8) (2.4 5)
Cluster 21: (2 5.6)
Cluster 22: (4 6.4) (2.4 6) (2.4 5.8) (2 5.8) (2 5.4)
Cluster 23: (2 5)
Cluster 24: (2 4.8)
Cluster 25: (2 4.6)
Cluster 26: (3.6 6) (3.2 5.8) (2.8 5.6) (1.6 4.2)
Cluster 27: (2.8 5.8) (2 5.2) (1.6 4.4) (1.2 5.4) (1.2 5.2) (1.2 5) (0.8 4.6)
Cluster 28: (4 6.6) (2 6) (1.6 4.6) (1.2 4.8) (0.8 4.8) (0.8 4.4)
Cluster 29: (0.8 4.2)
Cluster 30: (1.2 4.6) (1.2 4.4) (1.2 4.2) (1.2 4) (0.8 4) (0.8 3.8)

Table 4.7: Output of history aggregation program
Figure 1.16: Pearson's classification scheme for a Gray-Scott system. Reproduced from [21].

observation of a multitude of simulations. Representatives of his classes are shown in Figure 1.16. Each representative is labeled by a Greek letter.

Figure 1.17 shows the location in parameter space of the classes that Pearson found. Compare with Figures 1.12 through 1.15 and Table 1.7.

Cluster 17 corresponds to Pearson's pattern $\lambda$, and clusters 16, 19 and 20 correspond to $\kappa$. Notice that cluster 19 includes a pattern at parameters $(-.021,0.052)$, which lies outside of the region where Pearson locates $\kappa$; however, visual inspection reveals that this pattern is similar to the others in the cluster. Cluster 22 corresponds to pattern $\epsilon$ and 26 corresponds to $\theta$, even though 26 contains an element that would be better placed in $\beta$. Cluster 27 includes $\alpha$ and $\beta$, but also contains transient patterns that Pearson did not discover. Most transient patterns (consisting of a single body
that grows and eventually dissipates) are grouped into cluster 30; Pearson did not classify those patterns because he only looked at systems in steady state. We include these transient patterns for the sake of completeness, and as an additional test for the history classifier. Other rapidly dissipating patterns were placed by the algorithm into cluster 29.

Some of Pearson's classes were not discovered by the system. Pattern $\zeta$, for instance, was found by the algorithm to be too similar to $\epsilon$ to merit a class of its own. Many instances of pattern $\epsilon$ were found, but were grouped as distinct classes.
In all, the algorithm discovered behavioral classes for the Gray-Scott system that overlap with Pearson's hand-made classification. The degree of agreement is significant. We can conclude that this algorithm succeeds in emulating the human understanding of difference when applied to these coherent patterns in a diffusion-reaction system.

The reason why the approach taken to solve this problem succeeds is that the domain of the problem does not include any substantial amount of background knowledge. People excel at tasks such as recognizing faces and other patterns for which we have a naturally-selected advantage. Diffusion-reaction systems, although existing in nature, are phenomena where much of our inborn knowledge is not of much use. When a researcher like Pearson wants to classify these patterns, he or she has to do it based on simple criteria such as overall shape of coherent objects and detection of typical transformations. The computer program developed in this thesis encodes those simple criteria, and hence it exhibits an approach that we expect would be successful wherever domain knowledge is not necessary or available for the task of recognition. On the other hand, if the task in hand requires identifying complex objects or events such as changes in facial expressions, we expect this approach to be inappropriate.
CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Thesis Summary

This thesis has introduced a new conceptual model, STA, intended to reason about time varying fields such as diffusion-reaction systems. STA was achieved by extending Spatial Aggregation so that it makes explicit representation of time and temporal change. For this purpose, various abstract operations were introduced to represent the notions of persistence and change. Common qualitative events such as birth, death, collision, separation, acquisition or loss of components or properties were identified for objects in spatio-temporal domains.

The ideas of STA were applied to a case study: the classification of behaviors in diffusion-reaction systems that exhibit coherent spatio-temporal patterns. Several techniques were applied for this purpose, namely, operations of abstraction of change, kinetic data structures and geometric shape classification.

The results of these study match previous observations made by other researchers that studied some of these phenomena [24], and these results are based on explicit representations, which are explainable to humans. What this thesis contributes that
had not been done before is the *automatic* differentiation of pattern classes by behavior. Pearson accomplished such a classification using a combination of analytic and empirical methods: from standard bifurcation theory, he located a region in parameter space where behavior shifts may happen, and then used computer simulation to see those behaviors. His account of what he saw is as empirical as that of a chemist providing a qualitative description of a never-before-seen phenomenon. He classifies the systems based on the shapes he observed, and on the typical transformations that he saw. The programs developed in this thesis accomplish a similar classification in a similar way: the challenge was to formalize certain notions that are intuitive for humans, such as that of similarity of shape.

5.2 Applicability

The methods described in this thesis have been applied to a particular class of phenomena. The approach used leverages two pieces of knowledge:

- Diffusion-reaction systems, with all their complexity of behavior, produce patterns that contain objects whose descriptions are relatively simple:

- Variations among behaviors of systems are qualitatively significant; for instance, one class of patterns may consist of small round spots, while other may consist of elongated, twisted objects.

This knowledge confers a distinct advantage: there is not much of a need for subtlety. The distances between shapes in shape space will be large if the shapes are qualitatively different, and the typical sequences observed will also be considerably different. This approach might also apply to other problems that also have these
characteristics, such as tracking vortices in fluids or identifying and tracking particular weather phenomena (such as hurricanes). On the other hand, this approach remains to be tested for situations where subtler behavioral differences need to be detected.

More in general, how well would this technique do if applied outside of this domain? We expect them not to perform as well when applied to problems where extensive domain knowledge is required. For example, tracking objects for computer vision requires solving problems such as that of object occlusion and representation from incomplete information, not to mention the existence of multiple perspectives, different levels of illumination and reflectance, etc. While it is certainly possible to address such problems taking from a STA perspective, the actual techniques to deal with the various issues would have to be developed from scratch or adapted from existing methods from the computer vision field. The technique of shape classification used in this thesis would probably be insufficient, for example. Similarly, the problem of examining weather patterns also requires extensive domain knowledge. While this problem seems more amenable to treatment from a STA perspective, it would still require specific techniques to be developed.

Although focused to a particular domain, the concepts of STA should fare well when applied to a variety of problems requiring spatio-temporal reasoning. The basic vocabulary used by STA to describe discrete events involving coherent objects borrows ideas from qualitative physics applied to fluid dynamics, and, with minor extensions, could be applicable to distant domains such as distributed temperature control, the study and prediction of traffic patterns, and realistic models of population dynamics. All those domains have in common the existence of high-level structures that undergo
transformations such as those that STA can represent. Similarly, all those domains could benefit from the use of a sampling particle system.

On the other hand, many techniques used in this thesis are rather specific. The use of Delaunay triangulations is limited to two-dimensional fields (but not restricted to diffusion-reaction systems). The curvature-based shape matching algorithms are also specific to two-dimensional domains, and are mostly applicable when overall shape matters more than specific detail.

The main contribution of STA is not a set of specific methods to solve particular problems, but rather a novel, unifying perspective to begin to reason about problems that involve temporal variation of fields, to identify the main sub-problems typically found in such domains, and to devise approaches to solutions to such problems.

5.3 Future Work

STA has ample room for the development of specific implementations to deal with different data structures and ways to change them efficiently, while abstracting away details of change that are not relevant at higher levels of description. This will be achieved through the incorporation and extension of further techniques from the kinetic data structures field.

Some of the algorithms developed in this thesis exploit the fact that the case study is of fields in two dimensions. For instance, the modifications introduced to Witkin and Heckbert’s algorithm take advantage from the fact that Delaunay triangulations are efficiently constructed and altered in two dimensions. It remains to be determined how well some of these techniques generalize. One possible solution to the need for
efficient neighborhood graphs would be to use Relative Neighborhood Graphs, which have been shown to be cheap to construct in three or more dimensions [34].

Other problems, such as that of shape classification, seem harder to generalize. Smoothing objects in higher dimensions is straightforward; however, segmenting them into regions that are well described by single curvature parameters is not. Objects in two dimensions have one-dimensional borders, and segmenting those borders consists of finding a beginning and an end for such segments. Furthermore, the segments have a natural order, that goes around the border of the objects. None of these properties are preserved in higher dimensions. This is further complicated by the fact that shapes in higher dimensions may have complex topologies. Of course, when the objects are relatively simple and they can be fitted to parametric curves, their shape can be abstracted as a set of parameters. Unfortunately such an approach is suitable to compare objects that were already similar to begin with, since they could all be fit to the same model. In general, the problem of classifying shapes in three or more dimensions remains wide open.

One possible way to approach the shape classification problem for the three dimensional case is to segment surfaces by drawing closed iso-lines (each for a uniform curvature value) and cutting the surfaces through those iso-lines of maximal curvature; the surface would be then divided into a collection of surfaces that have one single distinguishing characteristic, such as that of being a bump or a concavity. For the purpose of matching, these segments could be aligned following a fixed order, and trying out all combinations for rotational alignment. We expect such naive approaches to be computationally expensive, so looking for more efficient approaches to the problem seems like an interesting line of future research.
Another interesting topic of research is the possibility of addressing the "common fate" problem (see Section 3.2.5). STA in its current status relies on the ability to find spatial structure from the spatial domain alone. The question raised by the "common fate" problem is: can STA techniques and concepts be generalized so that spatial aggregation can stem from temporal change?

Finally, a worthy goal to pursue will be the integration of STA with the SAI programming environment developed by Bailey-Kellogg [1]. This would result in a programming environment where rapid prototyping of STA applications could be achieved, freeing time to consider design issues. The integration is two-fold:

- Creating extensions to the Spatial Aggregation Library, a C++ portable library that supports the development of C++ SA applications and that is at the core of the SAI programming environment. This step has already been partially achieved by providing the required new abstract interfaces and by implementing the algorithms required to support the application developed in this thesis. More tools need to be programmed to support a wider variety of data structures, as noted at the beginning of this section.

- Adding syntactical support to the language itself and to its interpreter. This may require conceptual revisions to the language, possibly including its implicit programming methodology.

Carrying out this extensive integration work is challenging. It requires the programmatic implementation of the notion of update, which requires an abstraction of the notion of change, including the information that codes such change. The challenge lies in the need to have a syntax that is not only general and powerful enough,
but that can be easily understood by users without having to worry about too much
detail, such as the data types of the objects that constitute the change detail.
BIBLIOGRAPHY


