Amorphous Geometry

Ellie D'Hondt[1] and Theo D'Hondt[2]

[1] Foundations of Exact Sciences (FUND)
[2] Programming Technology Laboratory
Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
{eldhondt|tjdhondt}@vub.ac.be

Abstract. Amorphous computing is a recently introduced paradigm that favours geometrical configurations. The physical layout of an amorphous computer is based on a possibly irregular and error-prone planar distribution of a large number of simple processing components; this is well-suited for handling spatial structures. It has come to our attention that the well-known and long-since established discipline of computational geometry could benefit from the amorphous computing approach. In this spirit we speculate on the use of having smart surfaces interact with their environment or with autonomous entities such as robots. It seemed natural to refer to this approach by the notion of amorphous geometry. Although at this stage our exploration of this concept is fairly modest, we feel that our experiments are sufficiently convincing and merit further study. Exploring the possibilities of amorphous geometry should show that amorphous computing can deal with various classes of problems from computational geometry and may eventually expand the field considerably while providing a basis for useful applications.

1 Introduction

The context in which this paper is set is a newly emerging domain in computing called *Amorphous Computing* [1]. It was developed in anticipation of the rapidly developing fields of microfabrication and cellular engineering. These support the mass production of small computational units with limited power, provided that there need be no guarantee that each unit works faultlessly. A random distribution of such particles, equiped with a local communication protocol, constitutes the basic model of an amorphous computer. However, while producing a system composed of such units is within our reach, there as yet exist no programming systems applicable to it. Amorphous computing is precisely the paradigm trying to fill the gap between the construction and the programming techniques required for an amorphous computer. More concretely, amorphous computing addresses the important task of identifying the appropriate organising principles and programming methodologies for obtaining predefined global behaviour through local interaction only; individual amorphous computing particles cannot rely on any global knowledge about the system they are a part of. Since disciplines such as biology and physics often operate within this context, they were adopted as a source for metaphors applicable within the field of amorphous

computing, which among others led to the development of the *Growing Point Language* (GPL) [2]. For a more detailed overview of amorphous computing and a comprehensive list of references we refer to [4].

We propose introducing amorphous computing into the venerable field of *Computational Geometry* (see for instance the excellent [3]). Computational geometry serves as a foundation for various disciplines, including but not limited to operations research, artificial intelligence and robotics. Nearly as old as computing itself, it encompasses a vast body of knowledge rooted in mathematics, algorithms and programming. The fact that amorphous computing addresses problems that exist in physical−generally planar−space, led us to the conviction that it might be a useful addition to the domain of computational geometry. The physical layout of an amorphous computer serves as a medium for the representation of geometrical configurations; therefore it seems natural to introduce the notion of *amorphous geometry*. Moreover, we can imagine a complete range of useful applications, such as smart surfaces interacting with their environment in any possible way. Thinking about amorphous computing can be done in terms of representative applications; one of them is an active anechoic wall that reduces noise by using an audio-sensing network of amorphous computing particles dissolved in the paint covering the wall. We like to fix our ideas about amorphous geometry by thinking about examples such as active hospital floors that can compute and visualise paths to be followed by people walking on them.

Exploring the possibilities of amorphous geometry is an immense task. In order to show that amorphous computing can deal with more than a few problems taken from computational geometry, much more needs to be done than we can present here. We will limit ourselves to a number of modest experiments that nevertheless illustrate the point we want to make. The significance of our work is dual: first, it provides an insight in how metaphors from computational geometry are translated into an amorphous context; next, concrete simulations of amorphous geometrical programs using GPL provide us with knowledge of how powerful an environment it is. Both aspects are discussed using the construction of polygons as a case study. Section 2 handles the metaphorical aspect of amorphous geometry, drawing on abstractions already present in GPL. In this way, both the GPL architecture as well as the metaphors themselves are explained. Our experiences with GPL are reported in section 3, where actual simulations of polygon construction and their associated amorphous programs are presented. Additionally, and in order to satisfy the need for more expressiveness, several extensions to GPL and their implementations are presented. We conclude in section 4 with some current and future work.

2 Metaphors for Amorphous Geometry

Ultimately, the objective of amorphous computing is to develop engineering principles and languages to control, organise and exploit the behaviour of a set of programmable computing particles. The first serious attempt to achieve this is GPL [2], a language adopting the construction of complex patterns as a model

for global behaviour. The specification of these patterns relies on biological ingredients such as tropisms and pheromones, which help direct the growth of structures towards or away from others. With the aid of these and other abstractions, complex patterns such as digital circuits can be constructed. In this section however, we use GPL as an aid to describe how metaphors from computational geometry are interpreted in an amorphous style. While the emphasis lies on the actual notion of geometrical metaphors, concretising them with the help of GPL also sheds light on the architecture of an amorphous solution to a particular geometrical problem.

One could say that the two most basic components of geometry are points and lines. Amorphous geometry has no sense without a clear interpretation of these two concepts. Relying on the one-to-one correspondence between points in space and amorphous computing particles, it is clear that a geometrical point is represented trivially by one amorphous particle, while a contiguous set of such particles arranged as a linear array play the role of a line. Since an amorphous computer is necessarily of bounded dimensions, only line segments can be represented. Suppose we are capable of identifying two amorphous computing particles as being the endpoints of a line segment that is to be constructed. Through local communication only, one endpoint has to initiate a "search" for the other endpoint, without knowing the global direction in which the second endpoint is to be found. In order to do this, one can fall back on the biological metaphor of chemical gradients[1]. In order to fix our thoughts, we will look at the concrete representation of a line segment (inspired by [2]) in GPL, which is as follows:

```
(define-growing-point (a-to-b-segment)
  (material a-material)
  (size 0)
  (tropism (ortho+ b-pheromone))
  (actions
  (when ((sensing? b-material)
    (terminate))
    (default
    (propagate)))))
```

The principal concept in GPL is the *growing point*, a locus of activity that describes a path through connected elements in the system. At all times, a growing point resides at a single location in the GPL domain, called its *active site*; for example, the initial active site for the above growing point is the point *A*. A growing point propagates itself by transferring its activity from a particle to one of its neighbours according to its *tropism*. At its active site, a growing point may deposit *material* and it may secrete *pheromones*. A growing point's tropism is specified in terms of the neighbouring pheromone concentrations; in

[1] Biological systems use chemical gradients to determine the positions of cells. A chemical is released from a cell, such that the concentration of this chemical decreases as one moves further away from the cell, hence giving an indication of distance.

this case, `ortho+` directs the growing point towards increasing concentrations of `b-pheromone`.

As the segment between A and B is generated, the generator process itself will require some cooperation from the endpoint B in order to produce the desired line segment. This is indicated by the presence of `b-pheromone` and `b-material` in the code above. Hence, a growing point is installed at B in the following way

```
(define-growing-point (b-point)
  (material b-material)
  (size 0)
  (for-each-step
  (secrete EDGE-LENGTH b-pheromone)))
```

For a schematic representation of the establishment of the line segment see figure 1. This is taken from [2], where the amorphous process generating the segment is described at length. Note that the segment only approximates a straight line.

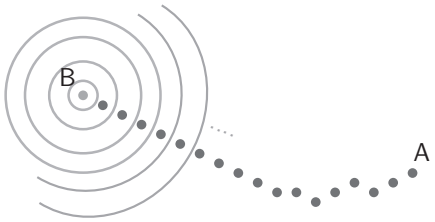

Fig. 1. Construction of a line segment.

The above code extracts are to be viewed as examples of concrete amorphous representations of the geometrical metaphors point and line segment. While we return to the GPL-specific code in the discussion of our experiments below, it should be clear that it is important that we can successfully incorporate geometrical concepts into amorphous computing; the specific code required in order to do so is at this stage of lesser significance.

3 Experiments in Amorphous Geometry

While the first step in developing amorphous geometry is to redefine metaphors in an amorphous style, at one stage a concrete implementation is required in order to carry out experiments. As a typical geometrical problem, we discuss polygon construction within the GPL framework. It is shown that amorphous

geometry is capable of solving more complex problems, while at the same time GPL's power as an amorphous language is investigated.

The problem of drawing a polygon can be divided into the subproblems of constructing line segments from one vertex to another. For example, when drawing a quadrangle with vertices `a`, `b`, `c` and `d`, one starts with establishing the growing points `a-to-b-segment` and `b-point`, as defined in section 2, at the vertices `a` and `b` respectively. Next, one has to define equivalent growing points at the couples of vertices (`b`,`c`), (`c`,`d`) and (`d`, `a`). As a result, each vertex is equipped with two growing points, a *<vertex>*`-to-`*<next-vertex>*`-segment` and a *<vertex>*`-point` growing point. We say "equivalent" and not "equal", since if we were to use the same definitions for the required growing points at every vertex, interference between pheromones would result. As a result, line segments would not grow in a straight way and might not even halt at the correct vertex. Hence, we have to use a unique pheromone and material for each vertex, and define an associated segment-like growing point that grows towards that particular pheromone. The whole process is presented schematically in figure 2.

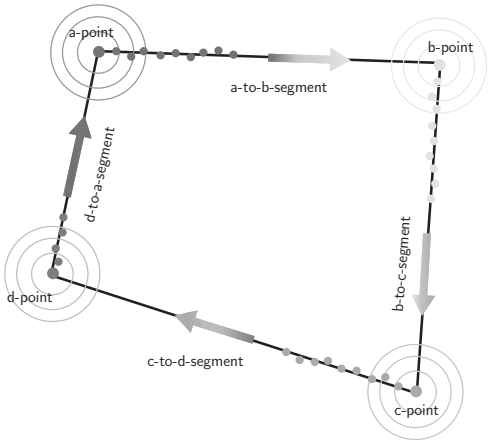

Fig. 2. Drawing a quadrangle.

An interesting side issue concerns the generation of self-intersecting polygons. Detection of edge material immediatly leads to the identification of points where edges intersect; this is typically a much more complex task in conventional geometry.

In the above solution for drawing a quadrangle, we are immediately confronted with the inability of GPL to parametrise pheromones. While material parametrisation is already present, we still need a different growing point definition for every vertex to avoid pheromone interference, resulting in a lot of duplicate code. Since GPL would gain considerably in expressiveness by allowing

pheromone parametrisation, we felt it was useful to include this in the GPL framework. Pheromone parameters can occur either in a `secrete-` or in a `tropism-`expression. In the former case, one has to ensure that the pheromone parameter is explicitly evaluated before carrying out the associated `secrete`-expression. The latter case is a bit more complicated since tropisms influence growing point propagation, a command dependent on communication. In GPL, tropisms are translated beforehand into a combination of a filtering and sorting process through the procedure `analyse-tropism`. Before a growing point propagates, it sends a stream of pairs of neighbouring pheromone concentrations through the analysed tropism. These pairs are first filtered according to the tropism in question, after which they are sorted in order of preference. On the basis of this result a growing point moves its active site towards a specific neighbour. However, when tropisms can contain pheromone parameters they are no longer part of a growing point's static information, since parameters have to be evaluated dynamically at each active site.

Once these additional GPL-features are implemented, the growing point definitions required to draw any polygon are as follows:

```
(define-growing-point (edge next-vertex-pheromone
                                          next-vertex-material)
  (material edge-material)
  (size 0)
  (tropism (ortho+ next-vertex-pheromone))
  (actions
  (when ((sensing? '(: next-vertex-material))
    (terminate))
    (default
    (propagate next-vertex-pheromone)))))

(define-growing-point (vertex vertex-pheromone vertex-material)
  (material '(: vertex-material))
  (size 0)
  (actions
  (secrete+ EDGE-LENGTH vertex-pheromone)))
```

With the help of these growing points, the following code constructs the desired quadrangle:

```
(with-locations (a b c d)
 (at a (start-gp (vertex 'a-pheromone 'a-material))
   (--> (start-gp (edge 'b-pheromone 'b-material))
    (--> (start-gp (edge 'c-pheromone 'c-material))
     (--> (start-gp (edge 'd-pheromone 'd-material))
      (start-gp (edge 'a-pheromone 'a-material))))))
 (at b (start-gp (vertex 'b-pheromone 'b-material)))
 (at c (start-gp (vertex 'c-pheromone 'c-material)))
 (at d (start-gp (vertex 'd-pheromone 'd-material))))
```

Here `-->` is the `connect`-command, which allows us to connect several growing points. Note that in the above, explicit pheromone and material names are required to be quoted, which constitues a change in syntax with respect to the previous version of GPL. Simulation results for the construction of a quadrangle with the GPL-illustrator in the above way, as well as for the analogous construction of a triangle, pentagon and hexagon are shown in figure 3. It may be inferred from these results that arbitrary polygons can be constructed in an amorphous way.

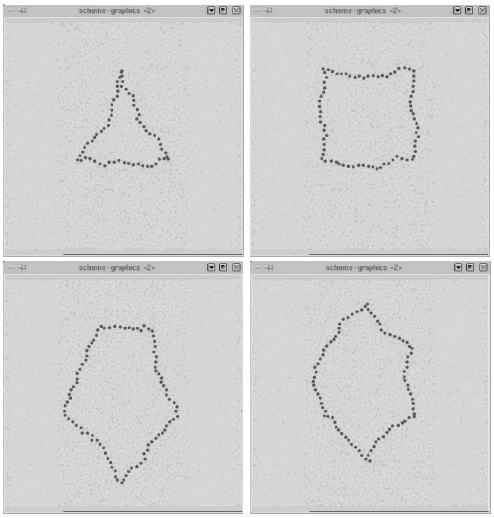

Fig. 3. Several polygons constructed with GPL.

4 Conclusion and Future Work

Amorphous computing is an interesting new programming paradigm that is based on the notion that a large network of computational particles can be embedded in physical surfaces. It seems almost obvious to propose it as an alternative to some of the more conventional computational techniques used to tackle geometrical problems. In this paper, we have suggested the notion of amorphous geometry to explore this idea. Typically, each amorphous processing particle governs a geometrical patch, such that the whole medium cooperates in finding a global solution to a particular problem at hand. We only scratched the surface by considering points, lines and polygons; we showed that they can be simulated by very simple programs, composed in the growing point language. We rapidly discovered that the expressiveness of GPL is too limited for tasks as elementary

as generating the edges of a polygon from the vertices. However, introducing a particular kind of parametrisation in GPL seems to solve the shortcoming, and did not pose any undue problem.

Extending GPL into a sufficiently expressive language so as to cover typical problems from computational geometry would seem to be a worthwhile continuation of the work described in this paper.

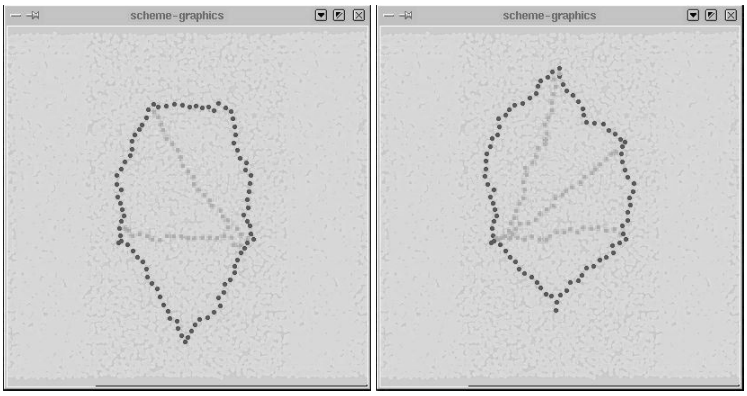

Fig. 4. Triangulating a polygon.

In figure 4 we show some initial results from an experiment to apply amorphous geometry to the triangulation of a polygon. This work is still very preliminary; at this stage the approach is too imprecise to justify reporting on it at length. However, it is sufficiently promising to warrant mentioning it as future work. Triangulating a polygon is an extremely representative case in computational geometry: it covers a whole family of problems. Being able to solve it transforms the idea of intelligent surfaces from sheer speculation into a conceivable−albeit futuristic−reality.

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, and R. Weiss: *Amorphous computing.* Communications of the ACM, 43(5), May 2000.
2. D. Coore: *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computing.* PhD thesis, MIT, 1999.
3. M. de Berg and M. van Kreveld and M. Overmars and O. Schwarzkopf: *Computational Geometry: Algorithms and Applications.* Springer (1998) second edition
4. E. D'Hondt: *Amorphous Computing.* MSc thesis in Computer Science, Vrije Universiteit Brussel (2000); available at `http://student.vub.ac.be/~eldhondt/PDF directory/thesisAmorphous.pdf`