Street Generation for City Modelling

Xavier Décoret, François X. Sillion

Street generation for city modelling

Xavier Décoret, François Sillion

iMAGIS GRAVIR/IMAG - INRIA

Abstract

In this paper, we present a complete solution for automatically retrieving the street graph of an urban model. Given a set of 2.5D polygons representing the buildings footprints and their heights, the algorithm constructs a graph that represents the street network (a node for a crossing, an edge for a street, each associated with a set of surrounding buildings) along with geometric information such as the width of the streets. We demonstrate how this graph can be used to analyze the city structure and give an example of its use with an automatic geometric modeler for city streets.

Keywords: Street generation, Voronoï, Median axis, City modelling

1 Introduction

City modeling is a growing field of interest. With the development and democratization of machines able to run high-quality simulations, more applications focus on building virtual environments. Whereas the first large environments were mainly poorly detailed terrains, todays machine capacities facilitate the computation of both large and complex 3D models. Cities are such models. They can be modeled at various, and possibly infinite, level of details (large blocks when viewed from above, detailed shops when walked through). They can have both large open spaces (squares, or avenues) and densely obstructed ones (narrow streets). Applications covers a large spectrum, from virtual tourism and video games to city planning and rescue or military training.

However, modeling such environments can be tedious. When the model has to faithfully represent an existing city, modeling every building by hand is inefficient. Even simple blocks textured with real photos requires a great deal of work and raises storage problems, as shown by the experience of the UCLA[1]. Acquiring the data automatically is also very difficult[Tel98] and can not yet be applied at large scale.

At the other end of the spectrum are imaginary, or semi-realistic cities such as the one generated for video-games like *Crazy Taxi 2*[2]. Such cities usually want to *look like* Paris or New-York. Some important monuments must then be modeled exactly but others just need to approximate the look and feel (facade style and color, street furniture,etc...) of the different areas. This leaves room for procedural modeling. Recent work [PM01] uses L-systems to automatically generate large cities by constructing a street network and populating it with buildings.

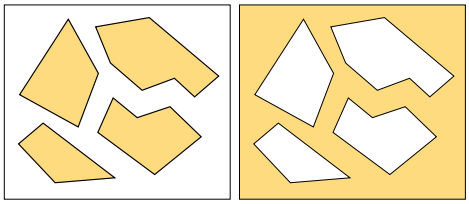
One interesting problem is the generation of streets. If the exact appearance of the buildings is required only for significant landmarks, the location of other buildings should always be correct. These locations can no longer be represented by L-systems since they involve complex social and historical interactions. However they can be retrieved automatically from aerial photographs and used to fix the position of procedurally generated buildings. Streets must then be generated *after* the buildings are placed,

In this paper, we present a solution to this problem of street retrieval where buildings are given. In section 2 we present the general idea, sections 3 and 4 explains the two phases of the process. Sections 5 and 6 show applications and results.

2 Overview

The input of our system is a set of building footprints, that are described as counter clockwise oriented 2D polygons with an altitude attached to each vertex. The altitude can be treated independently and will be discussed in section 4.1. The output is a set of center lines as this is the most common source of road data[3]. These lines are enhanced with extra information describing local and global properties.

Motivated by the idea that roads go in between buildings, more or less at equal distance of the ones on the left and right hand sides, we define the street network as a modified *medial axis* of a non simple polygon. The polygon we consider is obtained by subtracting all footprints from a large container polygon. In our tests we use a slightly enlarged bounding box of the buildings, as shown on figure 1. Other polygons



(a) footprints (b) difference

Figure 1: Difference polygon

(a) Buildings (inside is colored), and an enlarged bounding box. (b) the difference polygon whose median axis is the street graph.

could be used, such as a polygon shaped according to the terrain boundary for example. We will discuss in section 6 the role of this polygon. It is specified to the system as a building footprint with a *reversed* (clockwise) orientation.

The medial axis (also known as the *skeleton*) of a shape is formally defined as the set of interior points whose closest point on the boundary is not unique [Pav82]. More intuitively, if the shape represents a prairie whose boundary has been set on fire, it is the points were the flames will meet [Blu67].

[1] `http://www.ust.ucla.edu/ustweb/ust.html`

[2] `http://www.sega.com`

[3] `http://www.vterrain.org/Culture/RRF/index.html`

The skeleton of a polygonal shape is known to be consisting of segments of straight lines and parabola[OI92]. Algorithms to compute it can be found in [MR96, OI92]. However, we can not use them straightforwardly. First of all, a mixed representation with straight lines and parabolas is hard to manipulate and we would rather have a graph structure made only of lines. [AAAG95] proposes a novel type of skeleton for polygons that is not a medial axis but is only composed of straight lines. Unfortunately, it still suffers from the main two problems of skeleton computations methods :

input sensitive : if the polygon boundaries are slightly changed the skeleton shape is completely modified and may become unnecessarily complex (for our uses) as shown on figure 2. This is due to the fact that skeleton has an exact definition in term of distance to boundary, encoding far more information than we would need.

artifacts : the skeleton have protrusions everywhere the boundary has bulges or perturbations, as shown on figure 3. In many skeleton-based methods, filters and post process are used to prune them. In our case, we want to ignore them.

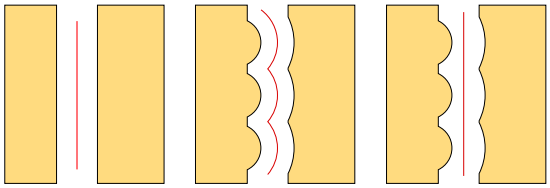


Figure 2: Input sensitivity

(a) A skeleton (in red) for two ideally aligned buildings. (b) With more detailed facades, the skeleton changes to fit, whereas a simpler road as shown on (c) would be sufficient.

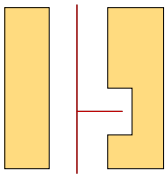


Figure 3: Artifacts

Bulges and noise on the polygon boundary produces extra branches of the median axis.

To account for the above mentioned issues, we propose a modified medial axis based on Voronoïdiagrams.

2.1 Our approach

We first approximate roughly the skeleton, and then filter the solution. This is a two-phase process with :

a topological phase which generates a graph whose arcs represent *streets*, and whose nodes represent *crossings* (junction of 3 or more streets).

a geometric phase The arcs of the graph are *shaped* to a suitable position in between the surrounding buildings. The shape is formed with only straight lines.

The advantage of this approach is that it reduces the overall problem to local filtering. Indeed, the first phase gives a collection of paths (arcs of the graph) and corners (nodes of the graph). As we will see later, each path is bound to exactly one building on its left and one on its right; and each corner is associated with a list of surrounding buildings. These bindings have the property of partitioning the space: every point on the 2D map that is not inside a building is uniquely attached to either a path or a corner. Therefore, the second phase is a local optimization process that fits each path or corner to the subset of the map bound to it. One then need only specify how a path should be placed in between 2 building facades and how a corner should be placed at the "junction" of several buildings.

The first phase uses Voronoïdiagram, that is a distance-to-boundary based construction of the skeleton but the generated collection of path and corners, together with the bindings and partitioning is strongly *insensitive* to the detailed shape of buildings footprints. On the contrary, the second phase depends highly on these detailed shapes but is no longer based on distance to boundary. Instead it maximizes an objective function under constraints. This decoupling is the reason why our approach is very robust relative to the input. It also naturally handles protrusion and dead-ends.

The next sections explain each phase in details.

3 Construction of a graph

3.1 Delaunay Triangulation

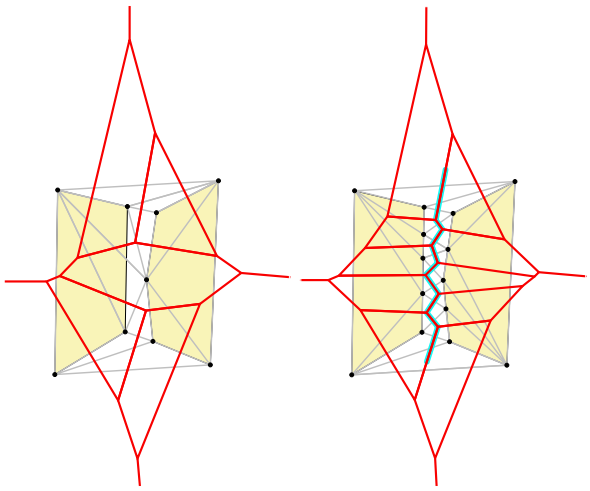
We sample the buildings' boundaries by adding vertices along their footprints' edges so that two consecutive vertices on a boundary have a distance of less than ε. The vertices are then Delaunay triangulated using the CGAL library[4]. One can intuitively see that if ε is less than the smallest distance between any two footprints (known as the *separation* of the two polygons in literature), then all the boundaries are covered with edges of the triangulation. Figure 4 shows an example where it is not true when not enough samples are used (ε is too large).

Currently, the user supplies an estimation of ε as the "thinnest expected street". Typically, for our test databases where the unit is to represent one meter of the real world, we chose a value of 1.0. Note that we could compute exactly this value by measuring the minimum distance between 2 polygons using for example [Ama94], with a hierarchical spatial structure to accelerate an otherwise quadratic search. However, this value is not critical. Choosing too small a value would yield the same results with simply an increase of the computation time during the Delaunay triangulation.

3.2 Dualization

The Voronoïdiagram is known to be the dual of the Delaunay triangulation. In CGAL, it can be obtained by taking the dual edge of every edge in the triangulation. However, we will not consider all the edges. Indeed, we ignore the ones that cover a footprint since their dual would be an edge crossing the buildings boundaries. To identify such edges, each building has a unique Building Identifier (BID), which

[4] `http://www.cgal.org`



(a) (b)

Figure 4: Sampling limit

Two footprints are Delaunay triangulated (gray lines) using too few vertices (a). The footprints are not covered by the triangulation and the corresponding Voronoïdiagram (in red) is not a relevant street. With our choice of ε, the problem dissapears (b) and the Voronoïdiagram gives a better street (emphasized in blue).

propagates to all vertices placed on this building's footprint. An edge $[AB]$ is ignored if $bid(A) = bid(B)$.

Otherwise, its dual edge $[ab]$ is retrieved together with the two (different) BIDs, $bid(A)$ and $bid(B)$. It is then inserted into a graph structure. The graph nodes hold a list of BIDs, and the edges hold a pair of BID, called *left* and *right*. Here is the pseudo-code for the edge insertion :

1. search if a graph node n_a is placed at position a, otherwise create a new node;
2. add $bid(A), bid(B)$ to the list of BIDs of n_a;
3. repeat for b;
4. create an oriented edge joining n_a and n_b, and set its right and left BIDs to $bid(A)$ and $bid(B)$.

Due to the orientation of the footprints and the way CGAL returns triangulation edges and computes dual, step 4 correctly gives left and right buildings. With another library one may have to check this property and eventually swap the edge to guarantee it. This will be important in step 3.4. Notice also that edges are oriented and that we do not insert the reverse edges in the graph.

3.3 Merging nodes

Each node of the graph has either 2,3 or more connected edges (what we call the *dimension* of the node). Nodes of dimension greater than or equal to 3 are called *corners*. A dimension greater than 3 happens in cases where rectangular shapes are involved such as in an orthogonal crossing depicted on figure 5(a). However, these cases are rare. The input polygons are generally not perfectly aligned and the result usually looks like figure 5(b), with two close nodes of dimension 3.

In such cases, we do not want to consider the tiny edge (emphasized in blue) between the two corners as a street. So every time we found two nodes of dimension greater than 2 that are directly linked by an edge, we contract the edge, thus merging the corners as shown on figure 5(c). Note that the configuration of buildings required to have such tiny edges make them independant of ε. With smaller values (more samples) such edges are still present.

3.4 Finding paths

By construction, the sequence of edges linking two different corners is made of edges with the same left and right BIDs, and of nodes whose dimension is 2. Such a sequence defines a *path*, with a left and a right BID.

At the end of this first phase, we are left with :

- a list of *corners*, each one associated with a list of the BIDs of the surrounding buildings;
- a list of *paths* connecting corners, each one associated with the BIDs of the building on the left, and of the building on the right.

Paths are oriented from a *start* corner to an *end* corner. Each corners knows the list of outgoing paths.

Figure 6 shows the result for a model of the Boston Financial District (courtesy of the Laboratory Of Computer Science, MIT).

3.5 Partitioning space

The corners and pathes are used to partition the 2D map. Each corner is projected on the surrounding footprints[5]. This defines a polygonal area attached to the corner (figure 7(a) shows an example). For each path, we also consider the polygonal area defined by the two sections of the left and right footprints in between the projections of the start and end corners (figure 7(b) illustrates this). These sections are called left and right *borders*. The two segments that join these borders are called the *start line* and the *end line*.

We also define the *width* of a path as the distance w between the left and right borders (*i.e.* the minimum distance of a point on one border to its projection on the other one).

4 Construction of schematic network

For each path, we construct a simplified centerline, the *skeleton*, with the following properties :

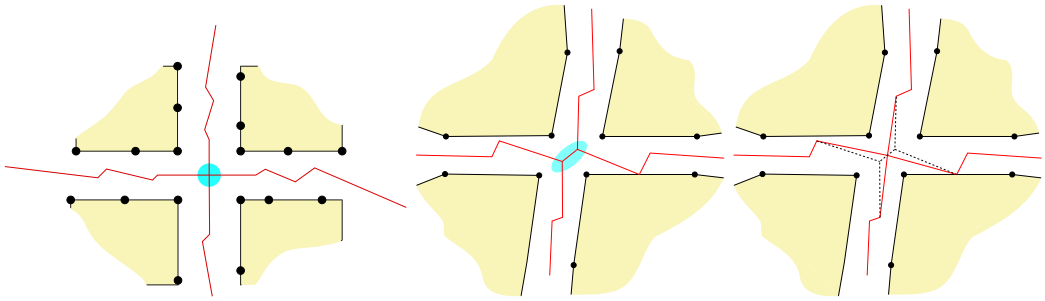
1. it is made of straight lines;
2. every point on the left and right border is further than half the width of the path from the skeleton[6].

To do so, we generate a set of sample median points in the following manner : for each vertex P on one border, we consider its projection P' on the other border and add the middle M of $[PP']$ to the set. We also add two special samples, the middle points of the start and end lines.

We then search for the longest valid segment joining two such samples. A valid segment is one that satisfies property 2. Once this seed segment is found, we successively

[5] We consider the orthogonal projections. They are not necessarily vertices of the footprints.

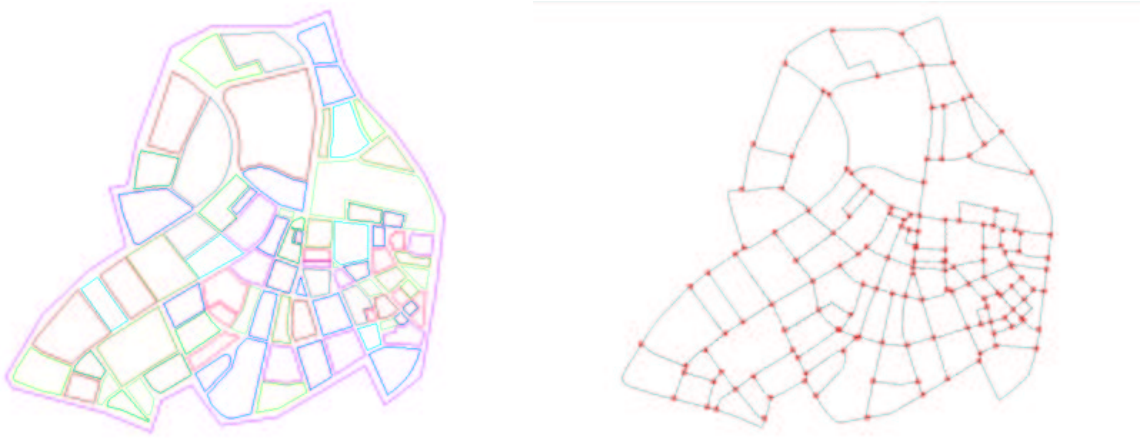
[6] In practice, to avoid numerical issues, we ask it to be 10% further.



(a) (b) (c)

Figure 5: Merging corners

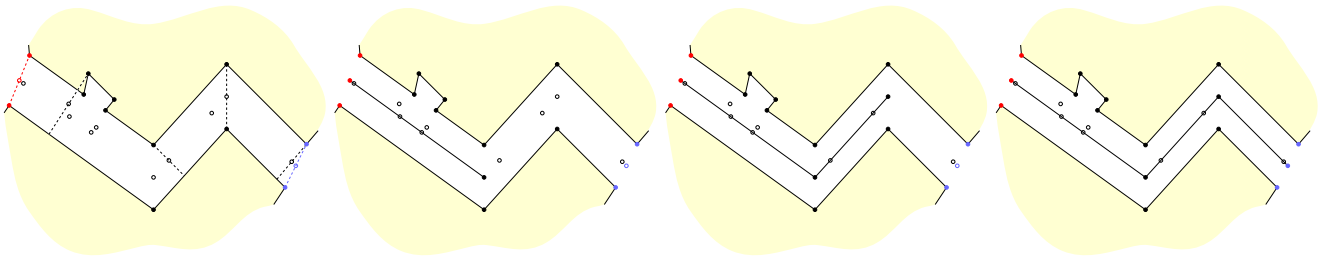
The Voronoïmethod generates (b) two close corners (a corner is a node with 3 or more edges) where we would like only one such as in the ideal configuration of (a). Such corners are merged into a median position as illustrated by (c).



(a) (b)

Figure 6: Corners and pathes

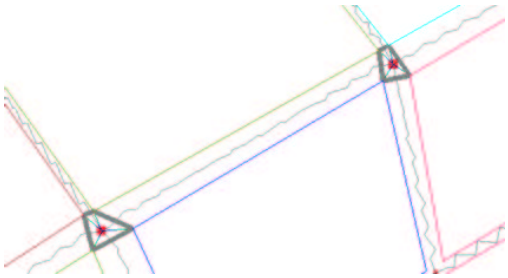
(a) A set of footprints, color indicates the unique building id (BID). (b) The resulting graph, with corners in red and pathes in grey. The jagged shape of the graph is not visible due to resolution, see figure 7(a) for a zoomed version.



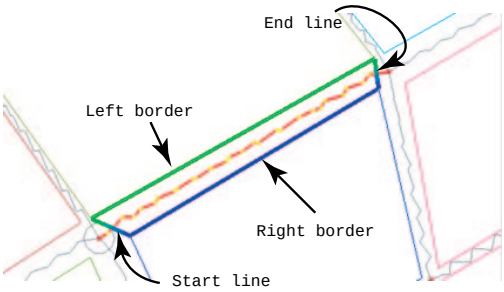
(a) (b) (c) (d)

Figure 8: Construction of skeleton

(a) shows the set of sample (hollow circles) with the construction indicated for some of them (dashed lines). A longest valid segment is first chosen (b) and then extended with other segments (c) until it reaches the end line (d).



(a)



(b)

Figure 7: Partitioning with the graph

(a) Bold grey polygons are the areas attached to the corners in red. (b) Emphasized is the polygonal area attached to the path in red. Blue line is the right border, green line the left one.

extend its extremities by adding maximal segments until we reach the samples at the middle of the start and end lines.

Figure 8 illustrates the process. Notice how the bulge in the upper building is filtered out; it adds more sample median points that are ignored at the benefit of relevant ones. The same thing would happen to extra samples generated by noisy, jagged footprints. In a certain manner, the process only retains samples corresponding to interesting features of the path shape.

4.1 Adding height

The whole process was done in 2D. The height of skeleton points can be retrieved simply. When a border's vertex at height h is projected onto the other border, it gives a point whose height h' is either the height of the vertex if the projection happens to fall exactly on a vertex, or the average of the heights of the segment extremities on which the projection lies. The height of the generated sample is then simply $\frac{h+h'}{2}$.

5 Applications

The data we build, corners and skeletons gives much information. Obviously, the graph can be used for many applications such as traffic planning or evaluation of the distance between two locations, for which much work has been done. However more information is available and can be used in a context of automatic modeling (models can be geometric models, models of moving pedestrians, etc...). The minimum and maximum distance of surrounding buildings to a skeleton helps finding narrow streets which can be useful, for example to decide if a street should be one-way. Corners' dimensions can be used to classify the type of crossings in the city :

dimension 3 means a secondary street joins a main one. The main one is constituted by the two skeletons who form the smallest angle. These skeletons themselves can be connected to corners of dimension 3, extending the main street. This way, avenues can be identified in the city.

dimension 4 means two streets cross each other. Width of the streets can be used to find out which one should have the priority and how traffic signs should be placed.

higher dimension means there is an important traffic junction. Maybe bridges or a roundabout should be placed here.

When modeling street geometry, the partitioning of space allow to generate geometry that exactly fits the space between buildings. This is what the next sections show.

5.1 Road geometry

Our solution for modeling is based on the following observation. It is easy to define a parametric model of street geometry for an ideal configuration. If one has for example to fill a rectangular area of $W \times L$, he can define the model of the figure 9. What is difficult is to define the model when

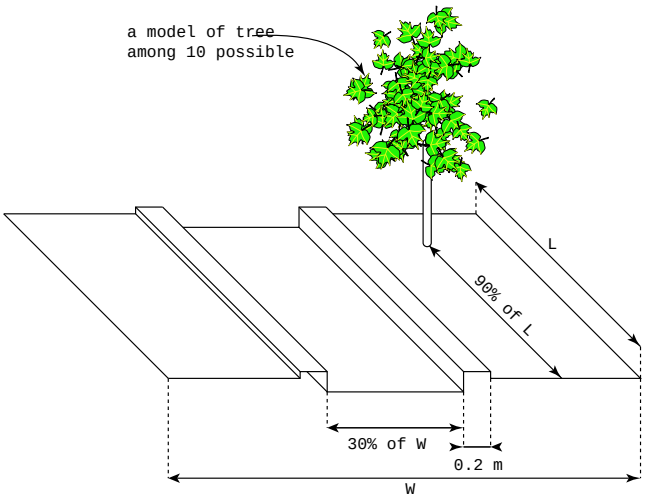


Figure 9: A parametric model of rectangular road

it is no longer rectangular, but has an arbitrary non convex shape, with perhaps jaggy boundaries. For a given path, what should be the parametric model to generate a geometry filling the area attached to the path?

Our approach is to cover such an area with simple shapes centered on the skeleton. We use 2 kind of shapes, rectangular shapes called *segments* and pie slices called *curves*.

At each of the inner vertices of the skeleton, we place the points O, M and N such as :

- the bisectrix of the skeleton segments joining at an inner vertex is also the bissectrix of $O_i\widehat{M_iN_i}$;

- the middles of $[OM]$ and $[MN]$ are on the skeleton;
- $\max(OM, MN)$ is maximum.

Curves are the pie slices $O_i\widehat{M_iN_i}$, and segments are the rectangles $M_iO_iN_{i+1}M_{i+1}$. Figure 10 illustrates this procedure.

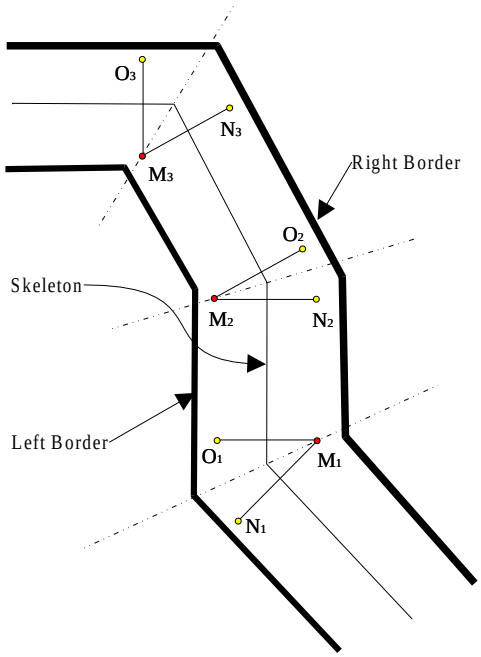


Figure 10: Covering with segments and curves

Parametric models of segments and curves are then used to generate geometry. The remainder of the path's area that is not cover by the generated geometry is triangulated and textured with a default texture.

What this scheme does is therefore to remove the burden of modeling from the modeler. The modeler simply defines parametric models for segments and curves (which is quite easy). These models specify position and shape of the road, sidewalks and any other relevant elements. The modeler then gives strategies for choosing a parametric model for a given path in a given city. These strategies can be based on information available in our system as described earlier. They can also use extra information from a GIS, such as style layers painted on a map of the city. The system then takes care of constructing a "clean" road using these models and strategies and fills the "dirty" remaining part around the buildings boundaries with triangles.

5.2 Crossing geometry

In the same manner, the user provides models to build geometry for corners. The parameters available to the model are : the the parametric models that have been used for each connected path.

6 Results

We tested our algorithm on two databases : the Financial District of Boston and a model of Vienna (courtesy of Peter Wonka). The table 1 and 2 sum up the informations and results. The computation were done on a Pentium 800MHz.

	Number of		
	footprints	corners	pathes
Boston	87	146	247
Vienna	458	740	1196

Table 1: Test databases

	phase 1	phase 2	total
Boston	246	3207	3453
Vienna	2756	9452	12208

Table 2: Computation times (in msecs

The whole computation time, even for the complex Vienna model is low. The "phase 1" time indicates the time to build the Voronoi diagram. The "phase 2" time is the time to build skeletons and geometry. We used very basic parametric models similar to the one of figure 9. With more complex models, the computation time could slighlty increase.

Figure 11 shows the resulting graph for the complex Vienna model. Note the peripheral roads. They are due to the bounding polygon we substracted the footprints to. They are necessary to have a graph with no leaf. However, they can easily be identified using the BID of the bounding footprint. This gives extra information since streets that connect to these road can be seen as entrances of the city.

Figure 12 shows the geometry generated for a curve. Notice the nice turn. Figure 13 shows the robustness of the method for a complex case where the building footprint has a non-trivial shape.

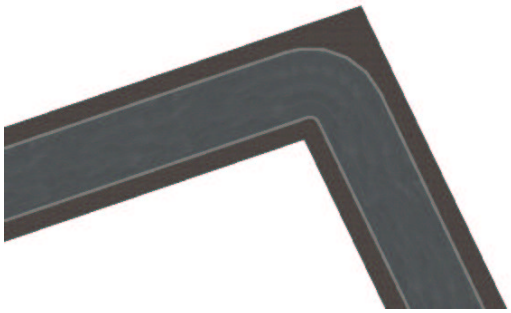


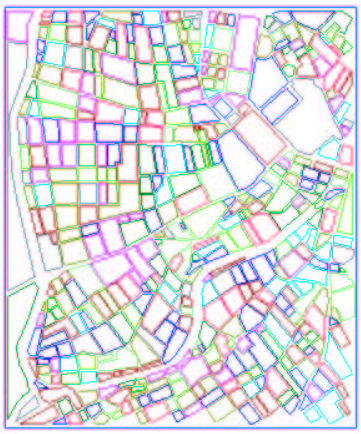
Figure 12: Geometry for a curve

6.1 Peripheric

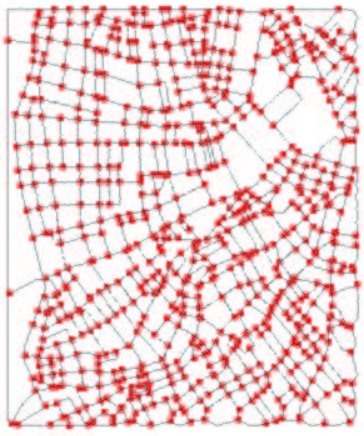
7 Conclusions and future work

We present an algorithm to retrieve the street graph of a city given the footprints of its buildings. The algorithm is very robust, relative to the input and does not suffer of the extreme sensibility of the skeleton algorithms it is inspired of. It therefore can be used on footprints scanned from aerial photographs.

It is fast and does not require obscure parameters to tune for each case. We believe that this algorithm can have



(a)



(b)

Figure 11: The street graph for Vienna

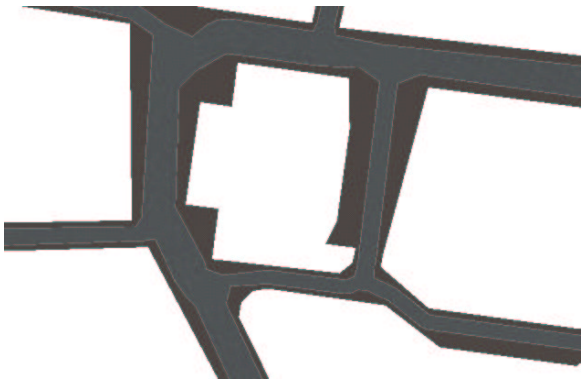


Figure 13: A complex case handel

many applications for the automatic modeling of a city. We demonstrate one use, where the modeler designs a library of simple geometric elements (segments and curves) and allows the system to pave the streets with them. The modeller no longer has to draw the street network himself as with today's tools such as the *RoadPro* module of *Multigen*©[7].

Our library is simple for the moment and was used to demonstrate the plausibility of our algorithm, however we plan to develop a more complex one. This raises some interesting problem regarding texturing of the roads. We also plan to explore automatic placement of street signs and urban furnitures. As we partition space and connect exactly to building footprints, we expect our system to handle connections between the road and buildings, for example with garage entrances. We finally want to use the graph structure to demonstrate another kind of modeling : animation of moving objects in a city.

[7] www.multigen.com

References

[AAAG95] Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. A novel type of skeleton for polygons. *J.UCS: Journal of Universal Computer Science*, 1(12):752–761, 1995.

[Ama94] N. Amato. Determining the separation of simple polygons, 1994.

[Blu67] H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380, Cambridge, MA, 1967. M.I.T. Press.

[MR96] N. Mayya and V.T. Rajan. Voronoi diagrams of polygons: A framework for shape representation. *JMIV*, 6:355–378, 1996.

[OI92] R. Ogniewicz and M. Ilg. Voronoi skeletons: Theory and applications. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition, CVPR*, pages 63–69, Los Alamitos, California, 15–18 1992. IEEE Press.

[Pav82] Theo Pavlidis. *Algorithms for Graphics and Image Processing.* Computer Science Press, Rockvill MD, 1982.

[PM01] Yoav I.H. Parish and Pascal Müller. Procedural modeling of cities. In *Computer Graphics*, Annual Conference Series, pages 301–308. ACM SIGGRAPH, August 2001.

[Tel98] Seth Teller. Automated urban model acquisition: Project rationale and status. In *Proceedings of the Image Understanding Workshop*, pages 455–462, Monterey, CA, 1998.