The Visual Computer (2012) 28:809-818

Procedural Generation of Villages on Arbitrary Terrains

Arnaud Emilien · Adrien Bernhardt · Adrien Peytavie
Marie-Paule Cani · Eric Galin

Published online: 18 April 2012

Abstract Although procedural modeling of cities has attracted a lot of attention for the past decade, populating arbitrary landscapes with non-urban settlements remains an open problem. In this work, we focus on the modeling of small, European villages that took benefit of terrain features to settle in safe, sunny or simply convenient places. We introduce a three step procedural generation method. First, an iterative process based on interest maps is used to progressively generate settlement seeds and the roads that connect them. The fact that a new road attracts settlers while a new house often leads to some extension of the road network is taken into account. Then, an anisotropic conquest method is introduced to segment the land into parcels around settlement seeds. Finally, we introduce *open shape grammar* to generate 3D geometry that adapts to the local slope. We demonstrate the effectiveness of our method by generating different kinds of villages on arbitrary terrains, from a mountain hamlet to a fisherman village, and validate through comparison with real data.

Keywords Procedural generation · Open Shape Grammars · Villages · 3D modeling

Arnaud Emilien · Adrien Bernhardt · Marie-Paule Cani
E-mail: firstname.surname@inria.fr
LJK (U. Grenoble, CNRS) and Inria, Grenoble, France

Adrien Peytavie
E-mail: adrien.peytavie@liris.cnrs.fr
Université Lyon 1, LIRIS, UMR5205, F-69622, France

Eric Galin
E-mail: eric.galin@liris.cnrs.fr
Université Lyon 2, LIRIS, UMR5205, F-69622, France

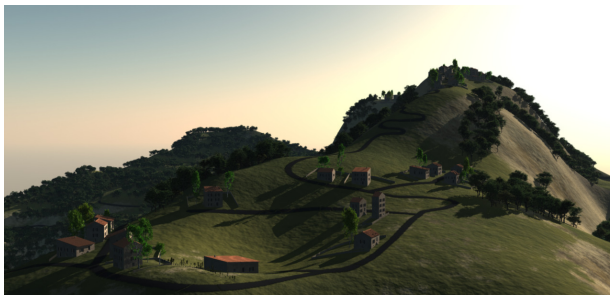


Fig. 1 A highland settlement generated by our system

1 Introduction

Offering the richest possible user experience in flight simulators, video games and 3D movies, requires the creation of an increasing amount of complex, textured 3D models. Modeling a large number of these by hand using standard software is a tedious task. This is especially true for landscapes, which should ideally combine arbitrary terrains with rivers, lakes, forests, and adapted human settlement. Several procedural modeling techniques have been developed to automatically generate such complex content. However, the modeling of human settlement has been restricted, up to now, to the generation of large cities, where building blocks are used to populate regular street networks. Adaptation to rough terrain has been scarcely studied, and no previous work was conducted, to our knowledge, on the generation sparser settlements.

This paper addresses the procedural modeling of small villages on arbitrary terrains, where bodies of water have possibly been predefined. Our method generates all the elements defining a village, from the road network to the individual parcels of land and to 3D houses adapted to the local slope.

Contrary to the large cities usually studied in Computer Graphics, most human settlements did not result from some pre-defined land-use plan, but from people progressively settling in safe, well served, sunny or convenient location for farming or fishing. Meanwhile, the road networks progressively grew and in turn attracted new settlers [2]. The result of such progressive settlement can still be observed in many regions over the world. For instance, it is the cause of the unique look of typical highland hamlets in the European Alps or of ancient villages on the banks of the Mediterranean Sea. Generating scattered settlements is a challenging problem that requires stepping away from the standard modeling paradigm used for cities [15]. In addition, modeling villages on hilly terrains requires generating complex land parcels, driven by both curved roads and un-even terrain slopes, and houses with non-regular doors and windows positions.

Our main contributions are as follows. First, we propose a new, hybrid settlement/road generation process that progressively creates a village layout on arbitrary terrain based on a growth scenario and on dynamic interest maps (Section 4). Secondly, we introduce an anisotropic conquest process that creates plausible individual parcels of land (Section 5). Finally, we present an open shape grammar able to adapt the geometry of houses to the local terrain slope (Section 6). Our method is illustrated by the generation of a variety of typical villages, validated through comparison with real layouts and pictures (Section 7).

2 Related work

Generating villages requires the generation of road networks, land tessellation into parcels and the creation of 3D buildings. This section reviews previous work in these domains. A complete survey on procedural city modeling can be found in [17].

Road networks Several interactive techniques have been introduced for editing or sketching road networks on arbitrary terrains. Roads were either represented by Bezier curves [3], or by clothoids enabling to express curvature constraints [12]. Closer to our concerns, automatic generation of roads on rough terrains was addressed by Galin [7,6]. Finding a path from a starting point to an end point subject to maximal slope and curvature constraints was expressed as an anisotropic shortest path problem. It enabled to take into account the environment, such as the presence of vegetation or water. Our method extends this approach to the case of road networks between hamlets or houses, leading us to a new, road re-use strategy.

Procedural modeling of cities Existing methods for modeling cities start by generating a street network. The cycles formed by neighboring streets are tessellated into blocks serving as footprints for buildings. Inspired from L-Systems [14], Parish pioneer work for street network generation was fully automatic [15]. User control through interactive editing was later allowed [9,11]. In parallel, other approaches were introduced such as tensor fields based or example-based city layout generation [4, 1]. Those methods first define the road network and then create the parcels in a second step. Therefore, they cannot be applied in the case of scattered settlement, where modeling the interaction between progressive settlement and road network extension is mandatory. Although not addressing the same problem, our progressive generation approach is closer to resource management methods introduced to simulate city growth or optimize land use [18,16].

When a city layout defining footprints for buildings is set, grammars such as L-Systems [15], split grammars [19] or shape grammars [13] are used for automatically generating their geometry. Our method for generating houses on hilly terrains extends existing grammar-based approaches: we define an open shape grammar enabling façade elements to self adapt to external constraints.

Non-urban settlements To our knowledge, the only work addressing the generation of non-urban settlements was dedicated to South African informal settlements [8]. After using a particle system to generate settlement seeds, different combinations of Voronoï diagrams were used to tessellate the terrain. This inspired our work on land parcels generation, although we had to develop a new, anisotropic land conquest method to account for alignments with road paths and terrain features, observed on real village layouts.

Lastly, progressive growth of non-urban settlements based on environmental constraints is somewhat similar to the spread of biological species. Inspiring from a model for lichen spreading on a support [5], we rely on particles to progressively seed settlement based on interest maps. However, villages develop in more structured ways, requiring us to take the road network into account in the simulation loop.

3 Overview and Notations

In this work, we call *village* any non-urban, sparse settlement, for instance a group of terraced houses around a church, a couple of remote hamlets and a few isolated farms between them. We define the *region of interest* on which the village is to be created as a compact $\Omega \in \mathbb{R}^2$. Ω is supposed to be connected to the outside world

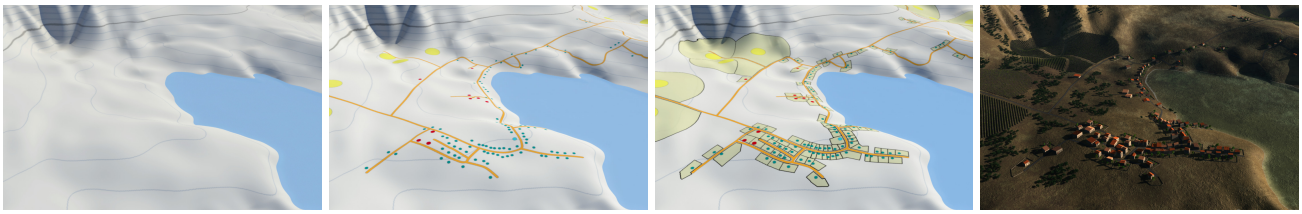


Fig. 2 Overview of our method: given an input terrain, we first generate the skeleton of the village (roads and settlement seeds), then we add parcels to the village layout, and finally generate 3D geometry for houses.

through a set of *connection points* Ψ, located at the edges of Ω, and that will serve as extremities for the future road network. The nature of the environment is pre-defined using functions over Ω. $h(\mathbf{p})$, $w(\mathbf{p})$ and $v(\mathbf{p})$ respectively denote the elevation, the water height and the vegetation density at a given point $\mathbf{p}$.

To enable the creation on the same terrain, of various villages, we use a village type $\mathcal{V}$ (e.g. high-land settlement, defensive village, fisherman village). A *growth scenario* sets $\mathcal{V}$ over time and launches the creation of *settlement seeds* B_i – marking the future locations of buildings, and of the roads R_j that serve them. We call *Village Skeleton* $\mathcal{S} = (\{B_i\}, \{R_j\})$ the result of this step. A settlement seed is defined as $B_i = (\mathcal{B}, \mathbf{p})$, where $\mathcal{B}$ is the building type (e.g. castle, church, terraced house or farm), and $\mathbf{p} \in \Omega$ is a position. A *road* R_j is defined by a set of nodes positions $\{\mathbf{p_k}\}$ controlling its central curve. During the generation process, we call *building encyclopedia* a function that returns, for each pair $(\mathcal{V}, \mathcal{B})$, the set of parameters used to seed a building.

The village layout we need to compute is not only composed of the village skeleton, but also includes a tessellation of Ω into individual land parcels $\mathcal{P}_i$ around buildings. We call F_i the footprint of the building B_i, defined as a sub-part of $\mathcal{P}_i$. This footprint will serve as foundation for the geometry of the building.

Our algorithm for generating villages on arbitrary terrains is summarized in Figure 2. Given Ω, a few environment maps and a user-defined growth scenario, we first grow the village skeleton, then generate land parcels and building footprints to get the village layout, and finally create 3D geometry. These three steps are detailed in Sections 4 to 6.

4 Growth of a village skeleton

In this section, we explain how we generate the building seeds and the road network that form the village skeleton. We use an iterative particle-based method for seeding buildings while considering environmental constraints and interest functions depending on the building type. The key feature of our simulation loop is to alternate seeding steps with creation of road segments connecting newly created buildings to the network (Figure 3). This approach allows for dynamic update of interest regions when new roads are created.

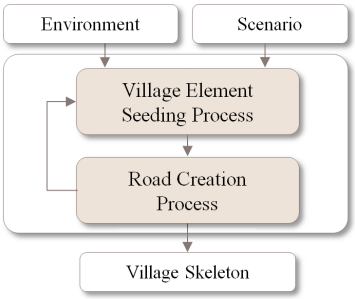


Fig. 3 Algorithm for the village skeleton growth.

4.1 Growth scenario

The growth scenario allows the user to control the evolution of a village by defining a list of temporal events which can either be a change of village type or the seeding of several buildings of a given type.

Change of village type The village type $\mathcal{V}$ sets some of the parameters used for seeding buildings, and thus affects the way a village will grow. Modeling a change of the environment or a change of the population needs over time is done by changing the village type. For instance, we can simulate a peaceful period following war by first choosing a defensive village type, followed by a prosperous, farming type.

Generation of building lots A vast majority of events in the growth scenario concern the creation of n buildings of a given type $\mathcal{B}$. During the execution of the scenario, the creation of a new building seed is immediately followed by its connection to the road network.

4.2 Building seeding

The way human settlements spread over a terrain is somewhat similar to the growth of natural species.

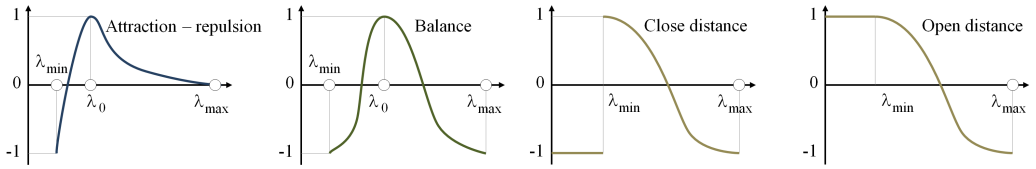


Fig. 4 Functions used to compute interests. From left to right: Attraction-repulsion function (sociability and worship), balance function (roads and slope), close distance function (water), open distance function (fortifications).

Our approach resembles the Open Diffuse Limited Aggregation model presented in [5] and adapts it in several original ways. Instead of moving particles over the land randomly to set the location of a new building, we rely on a stochastic positioning process followed by a local interest-based aggregation.

Seeding algorithm A position $\mathbf{p}$ for the building B to be positioned is randomly selected, and the conditions for constructing at $\mathbf{p}$ are checked (for instance, a farm cannot be built in the middle of a lake). If construction is possible, we compute a local interest value $\mathcal{I}(B)$ that measures the advantage for the building to be at its current location. The parameters of the interest function, extracted from the building encyclopedia, depend on the village and building types. Then, we perform a random choice, called the *aggregation test*, for deciding whether the position $\mathbf{p}$ should be kept or not for B, with a probability of success depending on $\mathcal{I}(B)$. In case of failure, we randomly select a new position and iterate the process until a good position is found.

Interest function When a building $B = (\mathcal{B}, \mathbf{p})$ tries to seed at a given location, we need to analyze the interest of this location in a consistent manner, while taking the village and building types into account and checking constructibility conditions. This is done by combining, with coefficients depending on the village and building types, n independent functions $f_i(B) \in [-1, 1]$ representing different interest criteria. A negative value is given if the location is undesired (-1 if impossible), while a positive value indicates a positive evaluation of the criterion at $\mathbf{p}$. The combination is controlled by the building encyclopedia, where a set of n weighing factors $\{w_i\}$ is predefined for each couple $(\mathcal{V}, \mathcal{B})$. The interest function is then given by:

$$\mathcal{I}(B) \begin{cases} 0 \text{ if } \exists i \text{ such that } \ f_i(\mathcal{B}) = -1 \\ \max\left(0, \sum w_i.f_i(B)\right) \quad \text{in the other cases.} \end{cases}$$

This method is general: it can combine a variety of criteria according to the desired result. We list below a few relevant functions for f_i (Figure 4). Some of them, stored as static 2D maps, are pre-computed before village generation starts. Others, such as sociability and accessibility, are dynamically updated when a new building (resp. road) is created. The values of these interest functions can be displayed on the ground by the user using a color-map based visualization, as shown in Figure 5.

Sociability This criterion measures the interest of buildings to be clustered. Since being too close to a neighbor is generally less attractive than being at a short distance, we thus use a sum of attraction-repulsion functions $f_{att}(d)$ where d is the distance between B and the surrounding buildings. The parameters λ_{min},λ_0 and λ_{max} are predefined for each couple of building types, and stored in the building encyclopedia. This enables us to set distinct preferred distances values between terraced houses and between farms.

Worship This function models the attraction of houses to religious elements such as temples, churches, statues, or monasteries. These elements, often at the very center of villages, are those around which the houses were initially constructed. We use the same kind of attraction function than for sociability, but computed only for the surrounding buildings of religious type.

Accessibility This function expresses both that building close to an existing road is easier, and that settlers usually prefer to create their house in a well served place. We use a non-symmetric bell-shaped function (Figure 4) of the distance d to the closest road. The parameters λ_0 (preferred distance to a road) and $\lambda_{\min}$, $\lambda_{\max}$ (defining the interval outside which construction is prohibited), depend on the building type.

Slope We use a bell-shaped function depending on building type to express preference to a given slope value, and to prohibit construction outside of a given slope range. This can be used to attract corn farms to flat areas and vineyards to hills.

Water Being able to attract houses to the sea shore, a lake or a river is important. We use a *close distance* function (Figure 4), a decreasing function of distance to the nearest water body. Minimum and maximum distances $\lambda_{\min}$ and $\lambda_{\max}$ depend on the building type.

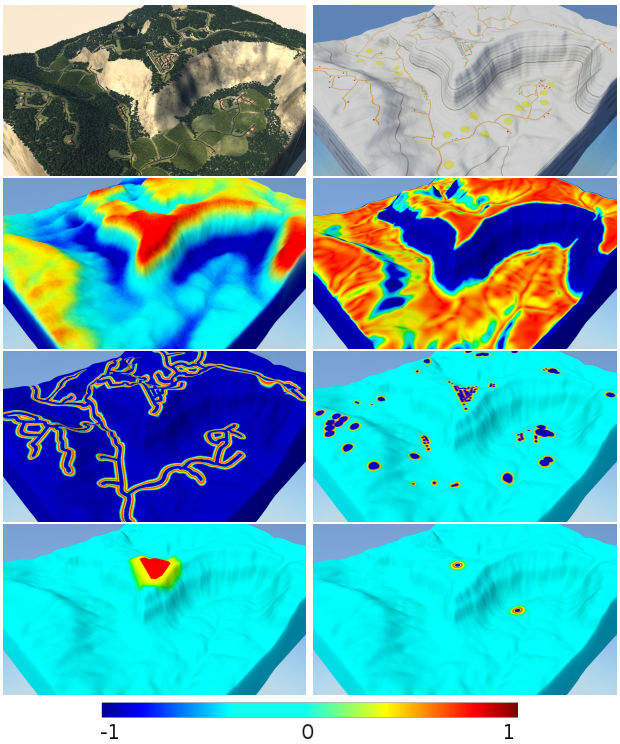


Fig. 5 Visualization of interest maps for the village type $\mathcal{V}$ = fortified and the building type $\mathcal{B}$ = house. Current village, current village skeleton, geographical domination, slope, accessibility, sociability, fortification, worship.

Fortification During wartime, building within a fortification or close enough to a castle, is important for houses. This interest is computed using an *open distance* function (Figure 4) (with $\lambda_{min} = 0$), a decreasing function of the shortest distance to the nearest fortified enclosure. f_{open} is equal to 1 inside fortifications.

Geographical domination Either an indicator of social superiority or as necessity for defense, being at a higher spot than surrounding buildings is important factor. Churches and monasteries are often built in overlooking places so that they can be seen from afar. The importance of height decreases with the distance to the point of interest. We use the following function:

$$f(\mathbf{x}) = \sum_{\mathbf{p}|\|\mathbf{x}-\mathbf{p}\|<r} \frac{h(\mathbf{x}) - h(\mathbf{p})}{1 + \|\mathbf{x}-\mathbf{p}\|^2}$$

$\mathbf{p}$ denotes sample points on the terrain, $\|\mathbf{x}-\mathbf{p}\|$ the Euclidean distance between $\mathbf{p}$ and $\mathbf{x}$, r the influence radius and $h(\mathbf{p})$ the height of the terrain at $\mathbf{p}$.

4.3 Connection to the road network

For the accessibility criteria to be correctly updated, we need to create the roads that connect a new building to the network just after placing the later.

Road construction We use a shortest path algorithm similar to the one in [7] to connect every new building to Ψ, the set of predefined connections to the outside world. The construction cost of a road is the sum of the costs of its road segments R, expressed as a weighted sum of different cost functions g_i:

$$\mathcal{C}(R) = \sum_{j=0}^{m} w_j.g_j(R)$$

In addition to slope, curvature and water costs, we use another function g expressing the cost for a road segment of crossing an existing building. This cost is set to a high constant to prevent collisions. To prevent the method from generating a fully ramified road networks, we note that, as in real life, the less costly connection to the existing network should be looked for (road re-use coming for free). This is modeled by introducing a new *re-use weight* $w_{ex} \ll 1$, used to reduce the cost of traversing existing road segments:

$$\mathcal{C}'(\mathcal{R}) = \begin{cases} w_{ex}.\mathcal{C}(R) & \text{if } R \text{ belongs to a road} \\ \mathcal{C}(\mathcal{R}) & \text{otherwise} \end{cases}$$

This way, a new segment is correctly connected to the network, as illustrated in Figure 6.

Road cycles Real road networks often include cycles providing shortcuts. We thus add a cycle construction step (Figure 6). Once a first road to a new building is computed, we try to extend it by looking for the closest road node p in a cone of angle θ from B, centered on the current road direction. We then use of usual method for generating a road between B and p. This road is created, leading to a new cycle, whenever it is not too close to the other road serving B (the later may occur due to slope constraints forcing roads to turn around obstacles).

5 Land parcels generation

Computing a village skeleton (roads trajectories and building seeds) is not sufficient for generating the layout of a village: we also need to tessellate the terrain into individual parcels of land, where houses, gardens or fields will be defined. A first approach, investigated in [8], is to rely on Voronoi diagrams to define a parcel of land around each building seed. This approach leads to rather isotropic parcels that lack structure and are not aligned with roads.

After carefully analyzing the layout of parcels in real villages, we observed that most parcels have one side neighboring a road and two sides perpendicular to it, the shape of the last side being driven by other

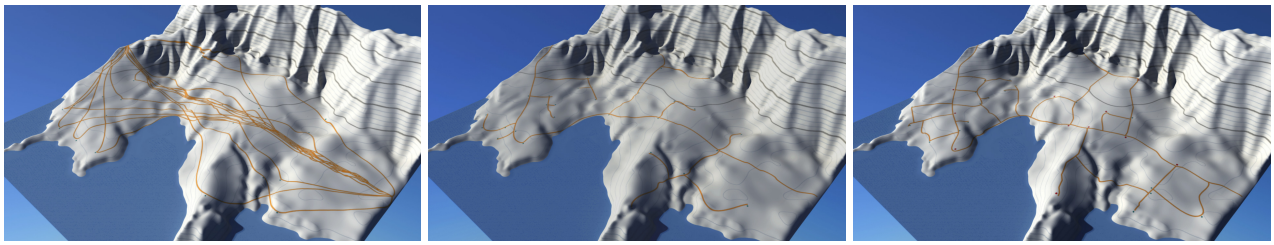


Fig. 6 Creating new roads. From left to right: without and with road re-use, with road cycle generation.

constraints such as the presence of neighbors or large changes in terrain slope. Therefore, we rely on a three steps, anisotropic land conquest method to define adapted land parcels. First seeds conquer their road territory. Then, they expand from the road using anisotropic conquest. Lastly, the resulting parcel is simplified to avoid sharp angles (Figure 7).

5.1 Road conquest

Since each building seed is served by roads, we first define the part of the roads belonging to each parcel (Figure 7). Let the source point S_i of parcel P_i be the projections of B_i on the closest road. We perform the road conquest by propagating P_i on both sides of S_i along the road, until collision with a neighboring parcel or until a maximum distance from S_i is reached.

5.2 Corner conquest

When two building are at the same distance from a junction, road conquest leads to a collision at the angle, resulting into non-plausible land parcels with sharp angles (Figure 8). Observing from real layouts that land at a corner between two roads generally belongs to a single owner, we use a corner conquest pass to resolve these conflicts: we allow the parcel P_i arrived first at a junction to annex a part of its neighborhood. Building seeds that lose access to roads are suppressed.

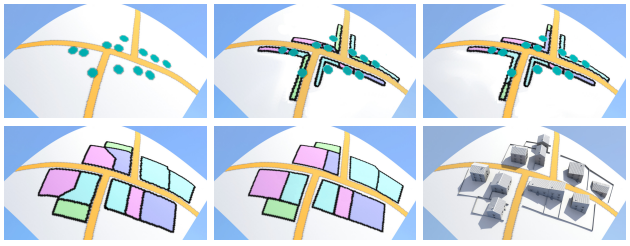


Fig. 7 Land parcels generation algorithm. From left to right: village skeleton, road conquest, corner conquest, region conquest, parcel simplification, building generation.

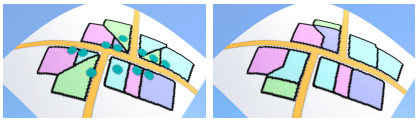


Fig. 8 Without (left) and with (right) corner conquest pass.

5.3 Anisotropic land conquest

Once building seeds own parts of the roads, their land parcel is grown using grid-based propagation. Each road cell belonging to P_i is marked in the grid as a source S, and each of these sources is associated a fund $c_{\max}$. Then the sources iteratively spread. The process stops when the total cost of conquest from S reaches $c_{\max}$.

In general, land parcels are orthogonal to the road, with a shape depending on other constraints such as slope. We thus use an anisotropic function to model spreading cost. The cost dc_s for conquering a non-occupied cell d_s is set to:

$$dc_s(d_s, \mathbf{n}) = \sum_{i=0}^{n-1} \omega_i \, c_i(d_s, \mathbf{n})$$

where c_i are independent cost functions modeling external constraints, all depending on the conquest direction $\mathbf{n}$, defined as the normal to the road at the source. The weights ω_i, set through the building encyclopedia, depend on the village and building types $(\mathcal{V}, \mathcal{B})$. They enable us to ensure, for instance, that a castle or a farm will get more land than a terraced house. We present below several useful cost functions.

Conquest cost This cost is set to the distance from the current cell to the source S. We use the Euclidean distance for farming fields, and the infinite distance $d(\mathbf{p}, \mathbf{n}, \mathbf{t}) = \mid \max(\mathbf{p} \cdot \mathbf{n}, \mathbf{p} \cdot \mathbf{t}) \mid$ for the houses and villas, to get quasi-quadrilateral parcels for them.

Water, wall and road cost The conquest cost for water, wall or roads cells is $+\infty$. This allows us to constraint the parcel shape with the road curvature, and to prevent the parcel to cross a wall or water bodies.

Slope cost As observed in real village layouts, the shape of land parcels (especially those with fields) is sensible to local slope, almost enabling to guess the main terrain features from them. We model this using an anisotropic cost function, for which spread in the main slope direction is difficult. Slope cost is computed as a quadratic function of the directional gradient of the terrain height, to reduce the influence of little slope variations and increase those of bigger ones.

5.4 Parcels simplification

Once the grid cells belonging to P_i are computed, we extract a poly-line representing its contour. Meanwhile, the shape of P_i is simplified, to account for the fact that even in small villages, parcel boundaries mainly consist of straight lines. This simplification is done in two steps, inspired from mesh simplification methods:

First, we remove vertices that have a little influence on the contour shape. Let $e_0 = (\mathbf{p}_0, \mathbf{p}_1)$ and $e_1 = (\mathbf{p}_1, \mathbf{p}_2)$ denote two edges. If the angle $\angle(\mathbf{n}_0, \mathbf{n}_1)$, where $\mathbf{n}_0 = \mathbf{p}_1 - \mathbf{p}_0$ and $\mathbf{n}_1 = \mathbf{p}_2 - \mathbf{p}_1$, is lower than a constant threshold ϵ we replace the two edges by $e_2 = (\mathbf{p}_0, \mathbf{p}_2)$. In the second step, we removing non-plausible acute angles that appear at some T-vertices of the parcel boundary mesh (Figure 7).

5.5 Building footprints computation

Because it is the most frequently observed shape for buildings (Figure 12), we decided to illustrate our method with only quadrilateral footprints for houses and villas. We initialize a quad in each parcel, at the closest position to the road, and oriented according to the closest normal to the road. The quad grows until it either reaches the maximal size for its building type, or collides with the contour of the parcel. If one of the segments is close to the contour, its vertices are projected onto it, enabling the generation of terraced houses when several neighboring houses use this strategy.

6 Geometry generation

The final step of our method is the creation of the three-dimensional geometry of the village, including roads, buildings and vegetation. While existing methods such as [7] can be used to generate accurate road geometry, existing methods for generating houses from their footprint [10] need to be extended to allow the generation of plausible houses on hilly terrain. In small mountain villages, windows and doors often have unusual shapes and façades often have complex layouts so as to conform to architectural constraints such as non-collision with the ground, alignment with floors whenever possible, and guaranteeing at least a door and a window per room. In this section, we introduce *Open Shape Grammar* to adapt geometry generation to such constraints.

6.1 Open shape grammar

We extend the concept of CGA shape grammar rules [13] by enabling the on the flight adaptation of newly created façade elements so that various plausibility constraints are met. In our work, this process is implemented for doors, windows and stairs.

We define an Open Shape Grammar as a grammar where the application of a selected rule can be canceled if some external constraints are not met, such as non-collision with the ground or with other buildings. Open Shape Grammar rules also incorporate adaptation mechanisms: each rule selection yields a series of attempts to create the output shape according to constraints of the environment. The element is created (and then the rule outputs success) as soon as a valid configuration is found. The application of the rule is canceled (and it outputs failure) if a maximum number of attempts is reached, and all have failed. See Appendix A for an example of Open Shape Grammar rule, showing the compatibility with standard grammars.

6.2 Geometry generation algorithm

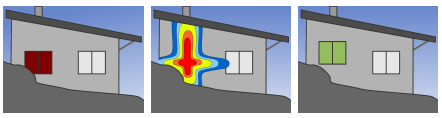
The geometry of a building is generated from the building footprint using a standard method [13]: first we generate the floors and the roof, then we add façade elements, such as doors and windows. The latter is done using an open shape grammar, with the two kinds of adaptations detailed below.

Position adaptation Each element is first positioned at the most plausible location: aligned with a floor and centered on the wall for a window; centered horizontally on the first floor for a door. Next, we move the element on the wall surface with a *displacement cost kernel* $\mathcal{K}$ until the element has a valid position (Figure 10). The kernel enables us to set preferences on the correction direction, such as favoring horizontal displacements over vertical ones.

Let $\mathbf{p}$ be the position of the element on the surface and $c(\mathbf{p})$ its cost. If the position is not valid, the unexplored neighborhood of the current position $\mathbf{p}$ is added to a priority queue with a cost equal to $\mathcal{K}(dx, dy) + c(\mathbf{p})$. The position of lower cost is evaluated next. The creation of

Fig. 9 Example of houses created by our system using Open Shape Grammars

the element is canceled after a user controlled number of failed tests. Figure 10 depicts the priority map for a window.



$$\mathcal{K} = \begin{pmatrix} 2.0 & 0.1 & 2.0 \\ 0.5 & \infty & 0.5 \\ 2.0 & 0.1 & 2.0 \end{pmatrix}$$

Fig. 10 Adaptation of window location: collision, priority map, result, displacement cost kernel.

Shape adaptation The second level of adaptation is to change the geometry of the façade element that we try to create (Figure 11). The candidate shapes are stored in a predefined priority queue, depending on the building type. To position an object, we initialize its shape to the top of the priority queue. If all positioning attempts fail, the shape is changed and the process starts again. If no shape is appropriate, the object is not built.

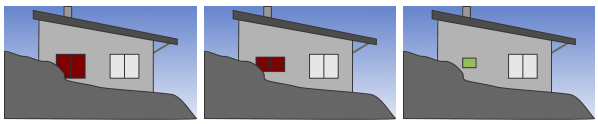


Fig. 11 Shape adaptation using an open shape grammar. For each window type we test for position (here, with only horizontal moves and a minimum distance between windows), and change shape if construction is not possible.

7 Results

Our village modeling system is coded in C++. Renderings were performed by using Mental Ray on the textured meshes we output.

Parcels generation To validate our parcel generation method, we compared the shapes of the parcels we create with those of real village layouts, with similar building distributions and road networks. One of our results is depicted in Figure 12. Parcels have similar shapes and the mean number of neighbors (2.873 with our model, 2.812 in the real data) and of contour edges (4.29 with our model, 4.068 in the real data) are similar.

Fig. 12 Comparison of real versus generated land parcels on terrains with similar roads and building distributions.

Geometry generation Our method of buildings generation with an Open Shape Grammar allows us to create homes on steep slopes without the doors and windows that are in collision with the ground. Note on the Figure 9 the change of position and shape of these elements. Appendix A details the rules we use to generate windows on building façades, and compare them to standard grammar rules.

Village diversity Figures 14, 13, 15 and 1 show different kinds of villages generated by our method. Figure 14 shows mountain villages, for which geographical domination was the main factor influencing seeding. Figure 13 shows a fortified village on a top of cliff where the main criteria were geographical domination and being protected by fortifications. These results demonstrate the effectiveness of our approach for generating settlements that conform to European layout styles. We believe that non-European village types could be created as well by modifying and tuning the growth scenario and the cost functions. A complete comparison and validation is beyond the scope of this paper.

Influence of parameters Figure 15 shows the influence of parameters during seeding. Distance to the sea was the main criterion for the creation of the first village, whereas the second was seeking for domination.

Performance Table 1 gives the time spent in each phase of the generation process. Land parcel generation is the most compute intensive part of our algorithm, due to the size of the grid used to perform the spread (we used a 4096×4096 grid). Note that depending on the desired output (with or without individual gardens around houses), this step can be skipped.

Fig. 13 Fortified village at the top of a cliff, using a war-time growth scenario followed by farming style settlement.

Fig. 14 A real (top left) and a procedurally generated highland hamlet (top right, bottom).

	Fisherman	Mountain	Fortified
Skeleton	4:00	5:00	7:00
Parcels	7:00	11:00	13:00
Geometry	0:20	0:30	0:30

Table 1 Computation time (in minutes) for generating the villages shown in Figure 13, 14 and 15

Limitations The main limitation of our method is the number of user-set parameters, currently 150 per village type. Fortunately, these parameters, stored in the building encyclopedia, can be reused to create a large variety of villages, depending on the terrain and on easily specified growth scenario typically created in 2 minutes. Displaying the interest values on the terrain helps users understand and parameterize the method, although their goals may still be obtained after a long series of trials and errors, as in every procedural generation method.

8 Conclusion

This paper presented an original method for generating scattered settlements on arbitrary terrains, enabling villages and hamlets, with the associated roads, forests and fields to be built on arbitrary landscapes. We demonstrated that our method can generate different types of villages with a coherent and adapted geometry. We validated our results through comparison with real layouts and pictures. In the future, we would like to focus on a user-controlled generation framework allowing real-time editing of villages. Our target application is an interactive system to enabling quick authoring of landscapes rather than a fully automated system. As it is, our method provides a very good starting point to develop such a system.

Acknowledgements This work was funded by the ERC advanced grant EXPRESSIVE.

References

1. Aliaga, D.G., Vanegas, C.A., Beneš, B.: Interactive example-based urban layout synthesis. SIGGRAPH Asia, pp. 160:1–160:10 (2008)
2. Barry, T. (ed.): A story of settlement in Ireland. Routledge (1999)
3. Bruneton, E., Neyret, F.: Real-time rendering and editing of vector-based terrains. Computer Graphics Forum (Eurographics) (2008)
4. Chen, G., Esch, G., Wonka, P., Müller, P., Zhang, E.: Interactive procedural street modeling. ACM Trans. Graph. **27**(3) (2008)
5. Desbenoit, B., Galin, E., Akkouche, S.: Simulating and modeling lichen growth. Computer Graphics Forum (Eurograhics) **23**(3), 341–350 (2004)
6. Galin, E., Peytavie, A., Guérin, E., Benes, B.: Authoring hierachical road networks. Computer Graphics Forum (Pacific Graphics) **29**(7), 2021–2030 (2011)
7. Galin, E., Peytavie, A., Guérin, E., Marechal, N.: Procedural Generation of Roads . Computer Graphics Forum (Eurographics) **29**(2), 429–438 (2010)
8. Glass, K.R., Morkel, C., Bangay, S.D.: Duplicating road patterns in south african informal settlements using procedural techniques. In: Proceedings AFRIGRAPH (2006)
9. Kelly, G., McCabe, H.: Citygen: An interactive system for procedural city generation. In: Game Design & Technology Workshop (2006)
10. Kelly, T., Wonka, P.: Interactive architectural modeling with procedural extrusions. ACM Trans. Graph. **30**, 14:1–14:15 (2011)
11. Lipp, M., Scherzer, D., Wonka, P., Wimmer, M.: Interactive modeling of city layouts using layers of procedural content. Computer Graphics Forum (Eurographics) **30**(2), 345–354 (2011)
12. McCrae, J., Singh, K.: Sketch-based path design. In: Proceedings of Graphics Interface 2009, pp. 95–102 (2009)
13. Müller, P., Wonka, P., Haegler, S., Ulmer, A., Van Gool, L.: Procedural modeling of buildings. In: Proceedings of SIGGRAPH, pp. 614–623 (2006)

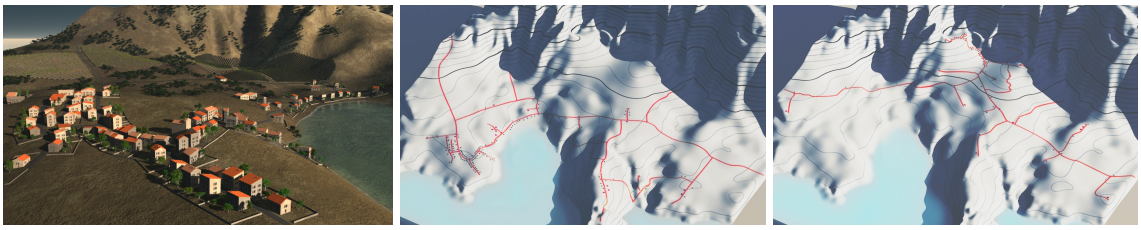


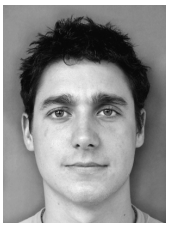
Fig. 15 Settlement on a coast. The left and middle pictures show a fisherman village, favoring distance to the sea; on right picture is a defensive village, which prefer geographical domination.

14. Měch, R., Prusinkiewicz, P.: Visual models of plants interacting with their environment. SIGGRAPH, pp. 397–410 (1996)
15. Parish, Y.I.H., Müller, P.: Procedural modeling of cities. In: E. Fiume (ed.) Proceedings of SIGGRAPH, pp. 301–308 (2001)
16. Vanegas, C.A., Aliaga, D.G., Beneš, B., Waddell, P.A.: Interactive design of urban spaces using geometrical and behavioral modeling. ACM Trans. Graph. **28**, 111:1–111:10 (2009)
17. Vanegas, C.A., Aliaga, D.G., Wonka, P., Müller, P., Waddell, P., Watson, B.: Modeling the appearance and behavior of urban spaces. Computer Graphics forum **29**(1), 25–42 (2010)
18. Weber, B., Müller, P., Wonka, P., Gross, M.H.: Interactive geometric simulation of 4d cities. Comput. Graph. Forum **28**(2), 481–492 (2009)
19. Wonka, P., Wimmer, M., Sillion, F., Ribarsky, W.: Instant architecture. ACM Trans. Graph. **22**, 669–677 (2003)

Appendix A: Example of Open Shape Grammar rules

```
WindowFacade → while (∃ shape in shapes priority queue)
                 while (∃ pos in positions priority queue)
                   if (try (Window(pos, shape))
                     {Walls} = { extract Window
                                 from WindowFacade }
                     throw success
               throw failure

Window (pos, shape) → if(external constraints (pos, shape))
                        {Woodent parts | Shutters | ... }
                        throw success
                      else
                        throw failure
```



Arnaud Emilien is a doctoral student at LJK, University of Grenoble, France, and at LIGUM, University of Montreal, Canada. He completed his Master degree in Computer Science at Grenoble Institute of Technology - Ensimag in 2011. His research interests range from real-time rendering to the procedural modeling of virtual worlds.

Adrien Bernhardt is a doctoral student at LJK, University of Grenoble, France. He received a Master degree in Computer Science at Grenoble Institute of Technology - Ensimag in 2006. His research interests include implicit modeling, sketch-based interfaces, and terrain modeling software.

Adrien Peytavie is an Assistant Professor of Computer Science at the Université Claude Bernard Lyon 1, France. He received a PhD in Computer Science from Université Claude Bernard Lyon 1 in Computer Science in 2010. His research interests include procedural modeling of virtual worlds and simulating natural phenomena.

Eric Galin is Professor of Computer Science at the Université Lumière Lyon 2, France. He received an engineering degree from Ecole Centrale de Lyon in 1993 and a PhD in Computer Science from Université Claude Bernard Lyon 1 in 1997. His research interests include procedural modelling of virtual worlds, simulating natural phenomena and modelling with implicit surfaces.

Marie-Paule Cani is Professor of Computer Science at Grenoble Institute of Technology. A graduate from the Ecole Normale Supérieure, she received a PhD from the University Paris 11 in 1990. Her research focus is making the creation of animated virtual worlds more intuitive, thanks to sketch-based interfaces, procedural models and interactive animation methods.