Wall Grammar For Building Generation

Mathieu Larive*
Oktal Synthetic Environment

Veronique Gaildrat†
IRIT

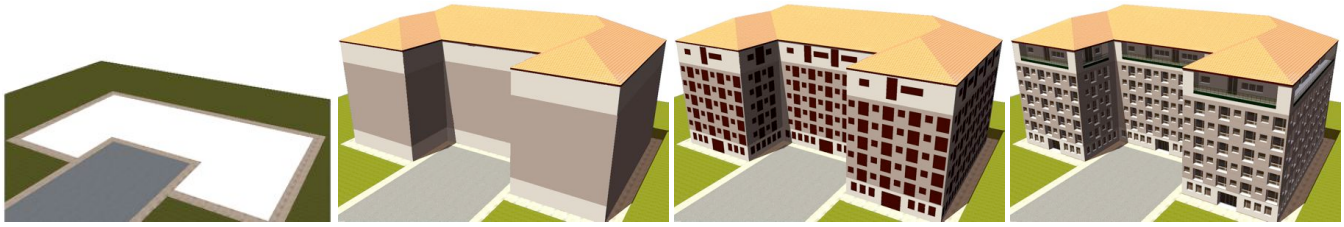

Figure 1: From left to right : Entry data (building footprint), first derivation step, setting up extrusions and final building.

Abstract

We present a new technique to automatically generate building exteriors. Our technique relies on the definition of building templates that will be applied on building description. The building descriptions require the building 3D footprints, its height and roof height. This information can be stored within a GIS (Geographic Information System). Most of the time, computer graphists use the texture repetition method in order to create building frontages. Due to this lack of flexibility, this method often disappoints the users who expect more realism. In this paper, we focus on the description of our frontage generation method. Building frontages are generated using a 2.5D wall grammar based on a set of rules that can be simple or detailed enough to fulfill the users wishes. Our method is as easy to use as the texture repetition but provides a higher level of realism and diversity in the resulting buildings.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.5 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: Procedural Modeling, Architecture, Formal Grammar, Computer-Aided Design

1 Introduction

Recent improvements of applications in virtual reality, video games and simulation of city expansion have increased the suitability of digital mock-ups for urbanism studies. For example, urbanization leads to emerging problems such as the influence of electromagnetic radiations or the forecast of urban transportation networks. The capacity to quickly generate credible digital city models helps the users to achieve their studies.

Detailed modeling of realistic towns is a real challenge for computer graphics. Modeling a virtual city which is detailed enough to be credible for a visitor is a huge task that requires thousands of hours of work. In this context, we think that work on automatic approaches can bring a real added value. These approaches represent a promising research topic which has to be developed because the current results do not satisfy the previously defined needs. In order to distinguish what exists and what remains to be studied, a state of the art of the current techniques can be found in [Larive et al. 2005]. As shown in Figure 2, the process of city generation can be divided into seven stages.

In this paper we focus on the stage that deals with the generation of building exteriors. We suggest handling this stage by the use of building templates. A building template consists of three different sets of templates (roof, frontages and groundwork). The generation of groundworks creates polygons sewing the 3D building footprint to the building walls. The generation of roofs uses a skeleton-based method to create various roof types. The generation of the frontages is based on a 2.5D parametric *wall grammar*. The main topic of this paper is about the frontages generation method that we have developed and its associated wall grammar.

The objective of the work described in this article is to create a system that imports building outlines defined by 3D polygons and building height. Then the system uses this data to create 3D building models as detailed by the user requirements. Typical usage will start with a 3D ground model inside a GIS. The building outlines and heights can be obtained automatically or defined by the user inside the GIS (that was the case for the example shown in Figure 15). Then, building templates have to be assigned to building outlines, this can be done manually, randomly or using any socio-statistical data available in the GIS. Once the setup is performed,

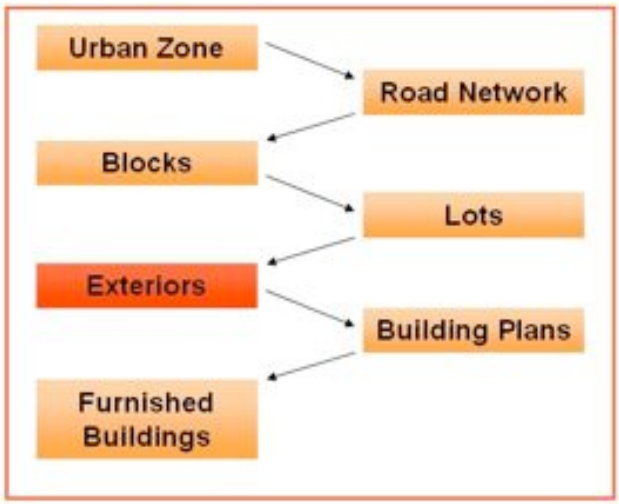

Figure 2: Hierarchical division of city generation

*e-mail: larive@irit.fr
†e-mail: gaildrat@irit.fr

Figure 3: Details of the building described in Figure 1.

our system will create the building models taking into account all this previously defined information.

Our work is mainly inspired by the one found in [Wonka et al. 2003]. Wonka et al. have designed a system able to generate any geometric details found on building frontages. Their system easily handles the vertical and horizontal alignments found in almost every human construction. The mechanism of *split grammar* that they introduced to obtain these results is powerful, even though it is complicated to describe every frontage decoration by their geometric properties. On the other hand, as a generic system, the system does not focus on identifiable patterns that exist inside a building frontage. That is why we wanted to create a grammar based system, able to obtain most of Wonka et al. results. Our grammar is designed only for frontages in order to simplify its usage and take advantage of frontage specificities. Another objective of our work is to facilitate the resort of repetition schemes on every part of our frontages and the usage of previously generated 3D objects (such as balconies or cornices).

1.1 Related Work

Since 1975, grammars, L-System [Prusinkiewicz and Lindenmayer 1991] or shape grammars [Stiny 1975] have been used in architecture to describe building or create design algebra, see [Mitchell 1990]. They have recently been used in various projects for building modeling. Three different approaches can be distinguished.

The first one, described in [RauChaplin et al. 1996], is mainly based on a study of the industrial architectural design in order to create a library of architectural elements that can be combined to visualise working/assembly drawings and 3D scenes. The exhaustive preliminary study led to quite a large database of shapes of vernicular buildings across the La Have river in Canada. Thus the system is able to create buildings that are detailed enough to allow the user to automatically assess the final cost of the building. The main drawback of this system relies on the fact that it is restricted to a highly specialized building type. Hencemore, this system is not able to deal with shapes that were not described during the creation of the

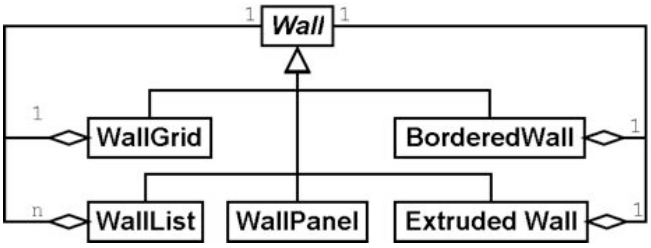

Figure 4: The relations between our classes/derivation rules

library of architectural elements. One of the goals of our work is to be able to define non-existing shapes.

The second one, described in [Wonka et al. 2003], derives buildings using split grammars. These are parametric set grammars based on the concept of shape. The system uses a second kind of grammar the *control grammar*, in order to control the propagation of the split grammar attributes. Using these two types of grammar, the system can create realistic buildings from high-level or precise descriptions. The generated buildings are extremly realistic, the usage of a unique rule base allows the user to create various kinds of buildings, but it can also require learning time in order to handle the complexity of the system. Therefore, this system can easily create too many faces (from 1,000 to 100,000 faces per building), and as previously stated it remains too generic for our goals.

The third one, found in [Parish and Müller 2001], generates simpler buildings using a vertical L-system. It can quickly create a huge number of buildings, but each building frontage is most likely rendered using a shader without any geometric details. Therefore the shape of the generated building is restricted to the skyscraper one.

In [Müller et al. 2006] (to appear), authors of the two previous articles are working together in order to create a grammar-based system compatible with buildings defined by complex mass models. The modeling of their buildings, based on a procedural approach with consistent mass models, creates complex buildings. The different parts of the building are self-sensitive due to additionnal data generated with the building faces. This property enables the split-grammar to take into account overlayed frontages: for example, a window from a given frontage will not be hidden by another frontage. Using the split-grammar and the mass modeling, the authors were able to model complex buildings such as the Petronas tower. As the work described in [Wonka et al. 2003], the methods used in [Müller et al. 2006] deal with the geometry in a multi-purpose way. These two approaches are able to describe roads or trees with the same grammar than the one used for the frontages whereas our work focuses on frontage description in order to make it as simple as possible.

The work described in [Gerth et al. 2005; Haveman 2005] have some properties common to our work. The authors identified realistic castle 3D models as a potential useful tool for researchers, museum curators and exhibition organizers. Their paper focuses on domain-dependant modeling tools. These tools are designed to allow people untrained in 3D modeling to express their ideas in 3D by themselves. Typically, these users are archeologists or art historians that possess knowledge on historical sites and monuments.

1.2 Overview

This paper is organised as follows: Section **??** describes the wall grammar used to generate the frontages, then Section 3 deals with the description of groundwork and roof templates. Our results can be found in Section 4, discussion about advantages and drawbacks of our tool in Section 5, and conclusions and future work in Section 6.

2 FrontagesTemplates

A frontage template contains a list of keywords, a primary wall and potentially a default material. A frontage template can only be chosen if it is geometrically compatible with the frontage (i.e. the minimal and maximal width and height of the frontage template fit the width and height of the frontage). Among the templates that can fit the frontage, the choice depends on the keywords stored in the footprint data by the user. For example, the user can define the *blind* keyword for a frontage without any aperture, or the *entry door* keyword for the main segment of the building footprint. The default material is used by the walls that do not have a background material in order to ensure graphical coherence for a complete frontage.

2.1 Wall Grammar

We have chosen a formal grammar representation to describe our frontages. Taking into account the impressive results found in [Wonka et al. 2003], we wanted to propose a system that would be simpler to understand and use. We decided to work on a *wall grammar* that can be seen as a *split grammar* that uses walls as its elements (instead of shapes). We attain a minimal set of five rules, a terminal one (see 2.1.2), two position rules (see 2.1.4 and 2.1.3) and two repetition rules (see 2.1.6 and 2.1.5). This choice allows us to easily deal with instantiation and repetition of given walls.

Formal grammar: it consists of a finite set of *terminal symbols* (the letters of the words in the formal language), a finite set of *non-terminal symbols*, a finite set of *production rules* with a left- and a right-hand side consisting of a word of these symbols, and a *start symbol*.

Each rule (wall) is able to determine its required space (min/max dimension in vertical and horizontal directions) by a simple computation on itself and its children. Then, among the geometrically possible primary rules (frontage template), one is chosen for the current frontage using the keywords defined in the template and in the building description.

2.1.1 Abstract Wall

The abstract class *Wall* stores the information that is shared by the different rules (see Figure 4 for class diagram). It deals with the *background material* as well as the final dimensions of the wall (horizontal and vertical). The minimal and maximal dimensions are computed depending on the wall's internal element size. Moreover, the user can define preferred dimensions that will automatically be used if they are geometrically valid.

2.1.2 Wall Panel

A *wall panel* is the simplest element of our system, it is our only *terminal symbol*. Besides the data that every wall shares (dimensions and background material), a wall panel can have an external reference to a 3D object. In this case, the 3D object will be instantiated in the center of the wall panel surface. When a 3D object is instantiated, the user can choose to create the background face (or not). For example, this is useful for 3D objects that model entry halls, in this case a background face must not be generated.

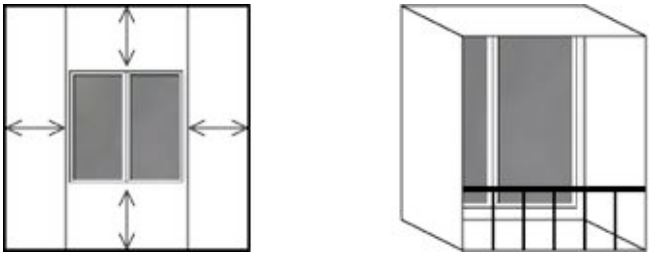

Figure 5: Left : bordered wall. Right : extruded wall with a negative depth variable

2.1.3 Bordered wall

A *bordered wall* (see Figure 5-left) is a wall with four margins (left, right, top and bottom) and a central element that references a wall. Each margin has a size and a resize policy that can be *minimum*, *maximum* or *fixed*. The default value for the resize policy is *minimum*, that means that the border cannot be smaller than the defined size.

$$BW \longrightarrow W$$

2.1.4 Extruded wall

An extruded wall (see Figure 5-right) is a wall that extrudes its child wall according to a given depth. This depth can be positive (respectively negative): the child wall and its border faces are in front of the frontage (respectively the child wall goes inside the frontage). Figure 6 illustrates the usage of the depth. Four booleans are used to define if the depth faces have to be generated or not. Moreover, an *extruded wall* can contain a 3D model, that will be instantied on its surface (i.e. depth = 0). This 3D model is used for decoration purpose as the guardrail in Figure 5.

$$EW \longrightarrow W$$

2.1.5 Wall list

A *wall list* contains several walls and an orientation that can be either horizontal or vertical. The different walls will be created from left to right (for a horizontal wall list) or from bottom to top (for a vertical one) in order to cover the frontage zone assigned to the wall list.

$$WL \longrightarrow W_1 W_2 ... W_n$$

2.1.6 Wall grid

A *wall grid* contains a unique wall that can be repeated in one or two directions (vertically and/or horizontally). For each of these

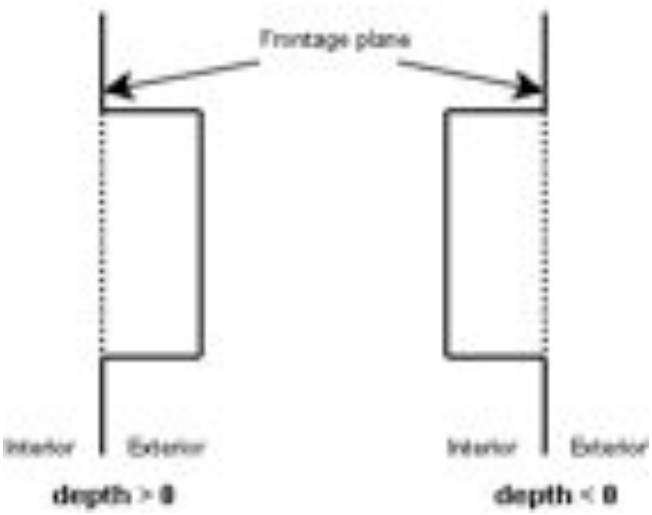

Figure 6: Positive and negative value of the depth of an extruded wall.

directions, the number of repetitions is controlled by the minimum and maximum cardinalities.

$$WG \quad \longrightarrow \quad W^{(1-n)(1-m)}$$

2.2 Implementation

The class diagram in Figure 4 shows the class hierarchy of our rule system. The more complicated functions are the ones that compute the size of the child elements of a given wall. This *resize* function is virtually defined in the *abstract wall* class, then redefined in its children. Figure 7 details the implementation of this function for the *wall list* class.

In this algorithm, R is the *wall list* rule, F is the frontage zone on which we try to instantiate the *wall list* and the ω^i are the walls that are stored in this *wall list*. The ω_{Hmin}, ω_{Hmax} and ω_{Hpref} store the minimal, maximal and preferred dimensions of each child wall. These three dimensions are computed beforehand by each child wall.

3 Groundwork And Roof Templates

3.1 Groundwork Template

A footprint can be a non-planar polygon (i.e. the ground under the building is not flat) but the floor of the building must be flat and horizontal. Groundwork is used to adjust the buildings with the ground. A groundwork template contains a groundwork type and a groundwork material if needed (that could be a texture or a colour).

The different groundwork types currently available are:

Z min the frontage starts from the minimum footprint altitude, no additional faces are generated, but part of the frontages can be buried (some openings could be covered by the ground).

//Test of the horizontal validity
for each $\omega \in R$
  if ($\omega_{Hmin} > F_H$ **and** $\omega_{Hmax} < F_H$) **then return FALSE**
end for

//Test of the vertical validity
if ($\Sigma\omega_{Vmin} > F_V$ **and** $\Sigma\omega_{Vmax} < F_V$) **then return FALSE**

//We try to use the Vpref
if ($\Sigma\omega_{Vmin}F_V - \Sigma\omega_{Vpref}$ **and** $\Sigma\omega_{Vmax}F_V - \Sigma\omega_{Vpref}$) **then**
//if ω ***has a*** $Vpref$***, it is used***
  $\omega_V = \omega_{Vpref}$
//if ω ***has not a*** $Vpref$***, we use a ratio to share the remaining space***
  $p = (F_V - \Sigma\omega_{Vpref})/\Sigma\omega_{Vmin}$
  $\omega_V^i = p \times \Sigma\omega_{Vmin}$
end if

//Else we try to favour the $Vpref$
while $\exists\omega$ **uninstantied**
  if ($\Sigma\omega_{Vmin} > F_V - \Sigma\omega_{Vpref}$ **or** $\Sigma\omega_{Vmax} < F_V - \Sigma\omega_{Vpref}$)
    then $\omega_V = \omega_{Vmin} \times F_V / \Sigma\omega_{Vmin}$
  end if
  else
    if $\exists\omega_{Vpref}$ **then** $\omega_V = \omega_{Vpref}$
    else $\omega_V = \omega_{Vmin} \times F_V / \Sigma\omega_{Vmin}$
  end else
end while

Figure 7: Resize rule for a vertical wall list.

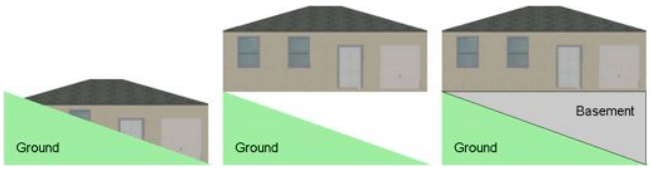

Figure 8: The three different types of groundwork templates. From left to right, *z min*, *z max* and *extruded*.

This groundwork type is mainly used when the ground altitude variations are small or when the template is meant for that case (i.e. chalet template).

Z max the frontage starts from the maximum footprint altitude, no additional faces are generated. Typically, the building is floating above the ground. This groundwork type is interesting when the user wants to take care of the groundwork himself.

Extruded the frontage starts from the maximum footprint altitude and groundwork faces link the frontages to the ground. For each segment of the building footprints, some vertical faces are created between this segment and the original footprint beneath.

Figure 8 shows these groundwork types. The groundwork material will only be used in the case of an extruded groundwork. If no material is given by the template, the groundwork faces will use the default material of the building template.

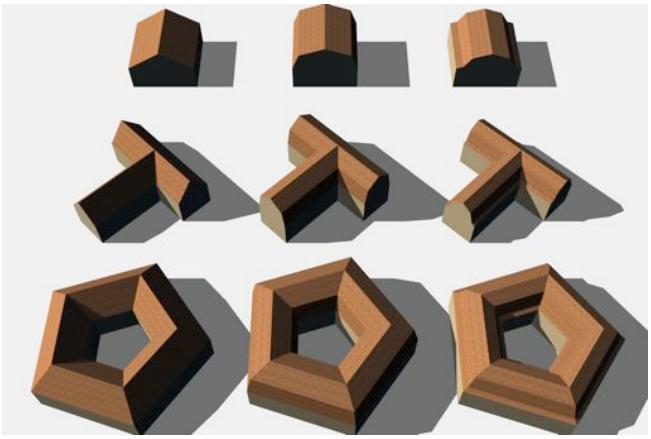

Figure 9: Examples of roofs for various two slope roof templates. From left to right : standard two slopes, gambrel, shed gambrel.

3.2 Roofs

One of the objectives of this work is to be able to accept any non intersecting polygon as entry data for our templates. It must even accept the ones with holes. This leads us to use the *Straight Skeleton* method [Felkel and Obdrzalek 1998; Eppstein and Erickson 1998] on the polygons describing the building footprint. As this algorithm is valid for such polygons, we are able to offer several roof types that can always be applied on our buildings.

The available roof types can be classified into four categories : the *flat roofs*, the *one slope roof*, the *two slope roofs* and the *four slope roofs*. The first two categories contain only one roof. The *flat roofs* can have a border on their side, whose height is the roof height. The *one slope roof* is obtained by putting the longest footprint segment at the roof height and its furthest segment at wall height. As the building outlines are not restricted to quadrilaterals, the main difference between the last two categories is the presence of gables on the generated roofs: the *four slope roof* templates do not create gables when the *two slope roofs* do. We will now detail our roof templates with two or four slopes. These templates were mainly inspired by the work on the two slope roof that can be found in [Laycock and Day 2003]. First we are going to give details on the *two slope roofs* that can be seen in Figure 9 :

standard two slopes : this roof type creates vertical gable faces on the smallest segments of the building outline. These faces often use the default material defined in the building template in order to give a coherent aspect to the building. Furthermore, the gable faces are created by displacing skeleton points to the building outline segment. The *regular faces* (roof faces that are not gables) are created with only one flap.

gambrel : this roof type creates two slope roofs with gables. The regular faces compound two flaps, the lowest one has a steeper slope than the upper ones.

shed gambrel : this roof type creates three flaps for each regular roof face. It can be described as the addition of a quasi horizontal flap at the bottom of a gambrel roof.

Then, we are going to give details on the *four slope roofs* that can be seen in Figure 10 :

standard four slopes : this roof type creates only regular faces, with a constant slope.

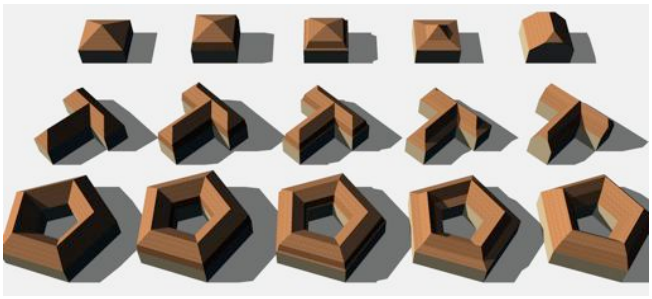

Figure 10: Examples of roofs for various four slope roof templates. From left to right : four slopes, mansard, pagoda, porch, alsatian.

mansard : this roof type can be seen as an extension of the gambrel one with only regular faces. Each slope contains two flaps, the lowest one has a steeper slope than the upper ones.

pagoda : this roof type aims at generating pagoda roofs. It can be seen as an extension of the shed gambrel ones with only regular faces that contain three flaps each.

alsatian : the roofs created by this type are hybrid roofs that have an upper part made by a standard four slope roof that is placed on a standard two slope roof (with gables).

A roof template is defined by a roof type, an overhang type, an overhang size, a slab size and some materials (for the roof, gables, overhang and slabs). When the user wants to create roofs bigger than the building outline, he can use overhang. Then he can decide if they are closed, open or have slabs. Slabs are vertical faces built on the outline of the roof. Overhangs and slabs can be defined on all our roof type.

4 Results

4.1 File Format and Workflow

Our system is implemented in C++, while our building templates are described and stored within a XML framework. This project was developed to create abstract geometry thus we do not depend on a given software or 3D library. Currently, two different implementations are available.

The first one is integrated into a terrain modeler based on the Geoconcept GIS. This integration allows us to import various GIS data, such as VMap, DTED, DFAD SEDRIS or DXF. This implementation is meant to generate large zones (currently up to 240,000 km^2).

On the other hand, we have developed a command line version in order to perform our tests and benchmarks. This version creates scene rendered using a viewer based on OpenGL Performer.

Having designed such a building template system, we would like to offer the user an interface which allows him to create his template (the xml format remains tricky to handle). Figure 14 shows the mockup of our frontage template editor. The upper left panel shows the current state of the building template. The center left panel shows the walls that are available : they can be part of the current frontage, or imported from previously generated frontages. The bottom left panel shows the available materials. In the upper right corner, a 3D window shows a preview of the current frontage.

Figure 11: Three different buildings generated from the same building footprint. Left: a *Hausman* building with an alsatian roof (94 faces). Middle: a glass building, with 2 external objects for the entry hall and a flat roof (350 faces). Right: a highly extruded building with a four slopes roof (5600 faces).

Figure 12: Example of building generated by the frontage template described in 4.3.

4.2 Performance and Complexity

One of our test scenes includes 17362 buildings. These buildings were defined from an aerial picture, each building footprint and height was manually defined by a computer graphist. Then this user created his own building templates and applied them on the buildings. A view of the final scene can be seen in Figure 15, the building generation took 7 minutes and 55 seconds, 920,182 faces were generated.

In order to test our own building templates, we used them on this scene. Using the simplest of our building templates (comparable to the one on the left in Figure 11), we generated this scene in 6 minutes and 51 seconds. The final scene contains 618,050 faces. Using the most complex of our building templates (comparable to the one on the right in Figure 11), we generated this scene in 3 hours and 27 minutes. The final scene contains 25,449,776 faces.

These results were obtained on a 2GHz AMD Athlon 64. The screenshots were generated using a multi-sensor ray casting engine (Figure 1, 9, 10 and 11) and a real time viewer (Figure 3 and 15).

4.3 Simple Building Frontage

Figure 12 shows an example of the application of the grammar shown in Figure 13. The rule (1) derives the first *wall list* from the *start symbol* ω. The rule (2) defines this *wall list* as a vertical one made of three rules: the rule (3) the *wall list* that deals with the first-floor, the rule (6) that handles the remaining floor and an *extruded wall* that creates a cornice at the top of the frontage.

The rule (3) encloses the entry door described by a bordered wall with two instantes of a horizontal *wall grid* described by the rule (4). This *wall grid* contains a vertical *wall list* made of two *bordered walls*, the first one defines a window, whereas the second one defines a cornice.

The rule (6) is a horizontal *wall list* made of four rules. Two instantes of the same vertical *wall list* (rule (7)) encloses the *wall grid* described by the rules (10) and (12). The rule (7) is defined by a bi-directional *wall grid* (rule (8)) and a horizontal *wall grid* that defines the last floor (rule (9)). The rule (10) is a vertical *wall grid* that contains the rule (11), an *extruded wall* that defines the fire stairs and contains a horizontal *wall grid* (rule 12). This one is defined using the same *bordered wall* as the rule (8).

Finally the rule (13) defines a vertical *wall grid* that contains the same bordered wall as the rule (9).

5 Discussions

Number of rules. As the user can define the complexity of his templates, our system can be used to generate credible digital city models as well as cities used for aircraft simulation (less complex models). Even though it is theoritically possible to create a unique template including all of our rules, this system was designed in order to enable the user to describe one distinct kind of building. A distinct approach, based on the use of a larger rule database allows the user to describe his buildings only using high level attributes (see [Wonka et al. 2003]). Future work on our rule selection mechanism will allow us to create this kind of complex building templates even if it was not our primary goal.

Usability of the system. Several of the templates presented in this paper were created by a computer graphist. Once the user has understood a given frontage template, he can begin to create his own templates by modifying the materials. Then, he can try to change the wall template parameters (dimensions, cardinality for a wall grid, border size and policy for a bordered wall etc...). After these two learning phases are performed, the user knows the five different

$$\omega \longrightarrow WL_1 \quad (1)$$
$$WL_1^{\uparrow} \longrightarrow WL_2 \quad WL_3 \quad WG_1 \quad (2)$$
$$WL_2^{\rightarrow} \longrightarrow WG_2 \quad BW_1 \quad WG_2 \quad (3)$$
$$WG_2^{\rightarrow} \longrightarrow WL_4 \quad (4)$$
$$WL_4^{\uparrow} \longrightarrow BW_2 \quad BW_3 \quad (5)$$
$$WL_3^{\rightarrow} \longrightarrow WL_5 \quad WG_3 \quad WG_4 \quad WL_5 \quad (6)$$
$$WL_5^{\uparrow} \longrightarrow WG_5 \quad WG_6 \quad (7)$$
$$WG_5^{\uparrow\rightarrow} \longrightarrow BW_4 \quad (8)$$
$$WG_6^{\rightarrow} \longrightarrow BW_5 \quad (9)$$
$$WG_3^{\uparrow} \longrightarrow EW_3 \quad (10)$$
$$EW_3 \longrightarrow WG_7 \quad (11)$$
$$WG_7^{\rightarrow} \longrightarrow BW_4 \quad (12)$$
$$WG_4^{\uparrow} \longrightarrow BW_5 \quad (13)$$
$$WG_1^{\rightarrow} \longrightarrow WP_1 \quad (14)$$

Figure 13: Example of the grammar used to generate the building shown in Figure 12.

kinds of rules and their parameters. From this moment, he's able to create new templates by himself from scratch.

Geometric complexity. As shown in Figure 11, our buildings can contain a limited number of faces. We are able to control the complexity of the generated buildings, using LODs, textures, geometry simplification (merging coplanar faces) and appropriate templates. It is also possible to control the complexity of buildings by instantiating more often the biggest frontage part (i.e. the ones with less polygons per surface unit).

6 Conclusions And Future Work

In this paper, we have introduced a new system for generating buildings from their footprints. The main advantages of our work are:

- Generation for any building footprint, convex or not, even with holes.
- Creation of simple or complex frontages, according to the user requirements.
- Availability of various kinds of roofs independantly of the footprint complexity.
- It has already been integrated to (and used in) a commercial terrain modeler.

A potential use of our system is the generation of city 3D models for the calibration of SAR (Synthetic Aperture Radar) simulation such as the ones described in [Soergel et al. 2005]. Our system is able to create cities of a given complexity which helps to validate the results of these kinds of simulation. We have to enhance our roof templates in order to enable the user to place 3D models on the roofs, because roof details are difficult to deal with in SAR simulation.

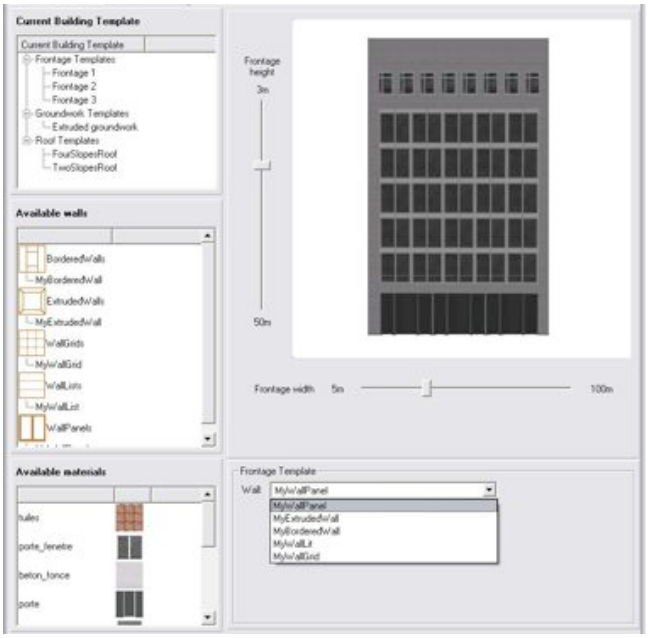

Figure 14: Current state of our frontage template editor.

The final goal of our research is to propose a complete city creation system. As shown in Figure 2, the generation of city models can be divided into seven stages. As we are already capable of producing furnished buildings (see [Larive et al. 2004; Xu et al. 2002]), our next research topic will deal with the generation of road networks, lots and building footprints. Once these stages will be completed, we will propose a complete city creation system.

References

EPPSTEIN, D., AND ERICKSON, J. G. 1998. Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions. In *Proc. 14th Symp. Computational Geometry*, ACM, 58–67.

FELKEL, P., AND OBDRZALEK, S. 1998. Straight skeleton implementation. In *SCCG 98: Proceedings of the 14th Spring Conference on Computer Graphics*, Comenius University. Bratislava, L. S. Kalos, Ed., 210–218.

GERTH, B., BERNDT, R., HAVEMANN, S., AND FELLNER, D. W. 2005. 3d modeling for non-expert users with the castle construction kit v0.5. In *Proceedings of ACM/EG VAST 2005*, R. e. S. Mudge, Ed., 49–57.

HAVEMAN, S. 2005. *Generative Mesh Modeling*. PhD thesis, TU Braunshweig, Germany.

LARIVE, M., LE ROUX, O., AND GAILDRAT, V. 2004. Using meta-heuristics for constraint-based 3d objects layout. *3IA'04*.

LARIVE, M., DUPUY, Y., AND GAILDRAT, V. 2005. Automatic generation of urban zones. *WSCG'2005*, 9–12.

Figure 15: Urban area made of 17 362 buildings, the building generation took 7mn and 55sec, 920 182 faces were generated.

LAYCOCK, R. G., AND DAY, A. M. 2003. Automatically generating roof models from building footprints. In *WSCG*.

MITCHELL, W. J. 1990. *The Logic of Architecture: Design, Computation,and Cognition.* MIT Press.

MÜLLER, P., WONKA, P., SIMON, H., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. In *SIGGRAPH*.

PARISH, Y., AND MÜLLER, P. 2001. Procedural modeling of cities. *ACM SIGGRAPH*.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1991. *The Algorithmic Beauty of Plants.* Springer Verlag.

RAUCHAPLIN, A., MACKAYLYONS, B., AND SPIERENBURG, P. 1996. The lahave house project: Towards an automated architectural design service. *Cadex'96, pp. 24-31*.

SOERGEL, U., MICHAELSEN, E., THOENNESSEN, U., AND STILLA, U. 2005. Potential of high-resolution sar images for urban analysis. In *URBAN*.

STINY, G. 1975. *Pictorial and formal aspects of shape and shape grammars : on computer generation of aesthetic objects*. Birkhauser.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Transactions on Graphics 4*, 22 (july), 669–677. Proceedings of ACM SIGGRAPH 2003.

XU, K., STEWART, J., AND FIUME, E. 2002. Constraint-based automatic placement for scene composition. *Graphics Interface'02*.