Representation of complex façades using typed graphs

Dieter Finkenzeller, Alfred Schmitt

Universität Karlsruhe
Institut für Betriebs- und Dialogsysteme
Am Fasanengarten 5
76131 Karlsruhe
Tel: +497216087505
Fax: +497216088330
E-mail : {dfinken, schmitt}@ira.uka.de

Abstract: In this paper we present a method for representing complex façades. For this purpose we developed a hierarchical decomposition for the façade. The steps of the decomposition are represented as nodes in a typed graph where the leaves are the atomic façade elements. Every node includes necessary attributes for its façade structure. The edges in the graph depict adjacent façade elements providing spatial information. With this approach complex architectural structures can be defined easily. A geometry engine traverses the graph in order to generate the detailed façade geometry but can stop at any given depth to produce exactly fitting geometry.

Key words: virtual architecture, procedural modeling, computer graphics.

1- Introduction

The manual modeling of detailed façade structures with tools like Maya [AM1] or 3D Studio Max [A3D1] is a very time-consuming and tedious task. Another problem is that models once created can not be changed easily.

This paper contributes to a solution that avoids these problems. We developed a method to represent a detailed façade with the following features. The façade is represented in a symbolic way with necessary geometrical information. Changes in the façade can be done easily by this symbolic representation. Finally, a geometry engine creates the entire façade automatically. The generated geometry fits together exactly so that adjacent structures do not overlap. This also simplifies texture generation because the surface occupied by a texture is exactly given with its geometry. Unwanted effects such as overlapping textures and geometry are excluded by definition.

With such a representation the modeling task takes place on an abstract level. This relieves the designer of the burden of difficult modeling tasks and gives him a high level of control. The façade representation and the geometry engine are part of a bigger system we are currently developing (see Figure 2).

The goal is to allow the designer to provide only a coarse outline of the building and some parameters about the type and style of the building and a kind of database is queried to produce the details for the façade (see Figure 1). With this information the graph is built and the geometry engine generates the detailed façade geometry.

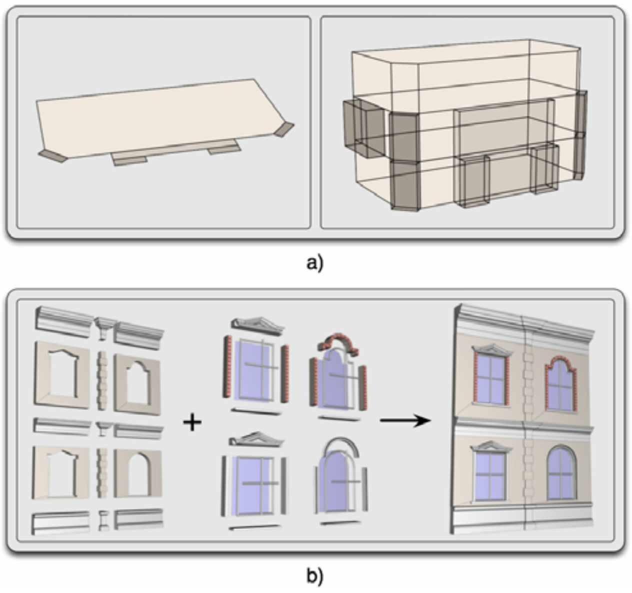

Figure 1: Modeling process for a) the user and b) the system.

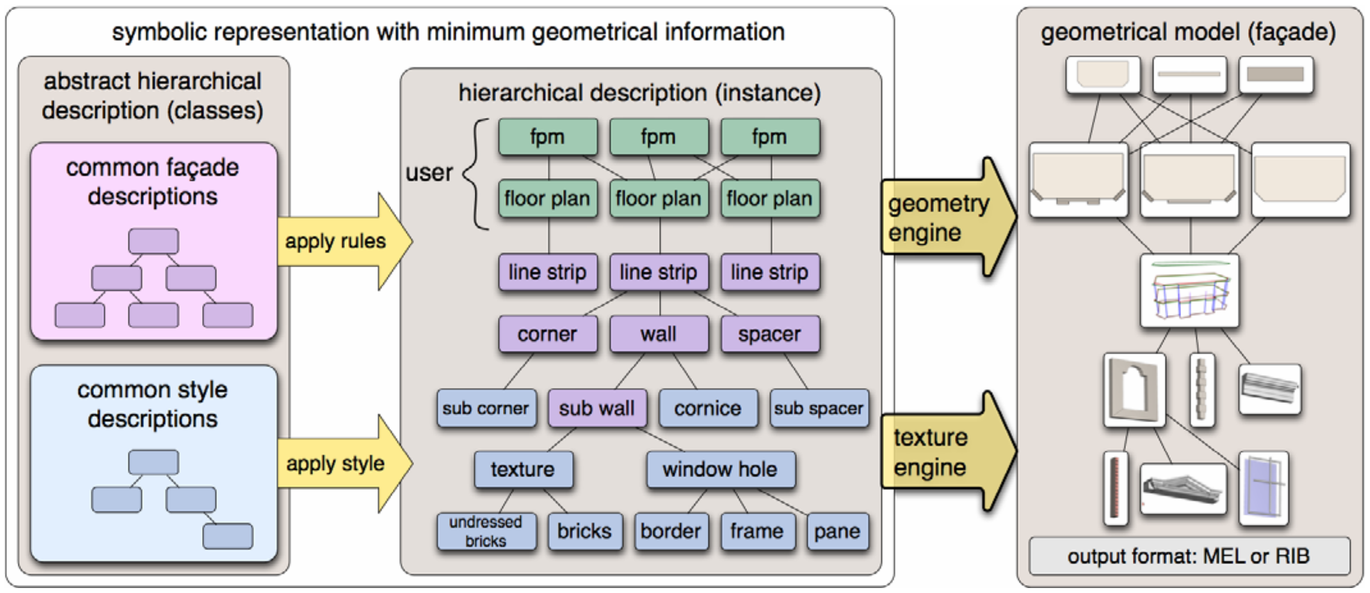
symbolic representation with minimum geometrical information
abstract hierarchical description (classes)
common façade descriptions
common style descriptions
apply rules
apply style
hierarchical description (instance)
user
fpm
fpm
fpm
floor plan
floor plan
floor plan
line strip
line strip
line strip
corner
wall
spacer
sub corner
sub wall
cornice
sub spacer
texture
window hole
undressed bricks
bricks
border
frame
pane
geometry engine
texture engine
geometrical model (façade)
output format: MEL or RIB

Figure 2: System overview.

2- Related work

There are a lot of different research areas which contribute to the task of automated building and façade generation which are mainly subsumed in the task of procedural modeling of cities, buildings, façades, and architectural structures. In the following, we concentrate on the representation of façades.

For procedural modeling there are basically two common techniques. The first technique is rewriting which operates on strings. Especially Chomsky's work on formal grammars caused a huge interest in rewriting systems. A condensed introduction is given in [S1]. The second technique applies L-systems. The biologist Aristid Lindenmayer proposed this mathematical formalism–which also uses rewriting–in 1968 as a foundation for an axiomatic theory of biological development [PL1]. The basic idea is to use a set of rewriting rules or productions to create complex objects by successively replacing parts of a simple object. In Chomsky grammars productions are applied sequentially, whereas in L-systems they are applied in parallel, replacing simultaneously all letters in a given word.

There are several research papers concerning the procedural modeling of whole cities using grammars and L-system like [PM1, MW1], [GP1, GP2], and [A1]. Parish and Müller [PM1] create the geometry for the buildings with a parametric stochastic L-system. Their buildings are generated by manipulating an arbitrary ground plan. When producing the building they use several modules for the L-system, e. g. transformation modules, extrusion modules, branching and termination modules, etc. Textures are created for the façades. Therefore they designed a tool for the semi-automatic generation of façades which they call *layered grids*. In [GP1] and [GP2] Greuter et al. focus on a real-time generation of procedural cities. They construct their two-dimensional floor plans from level to level by adding and merging a randomly selected regular polygon or rectangle. For texturing the buildings they use a fixed number of predefined texture patterns to give the buildings an appropriate look.

The authors of the papers [BB1, BJ1, LD1, HF1] present specialized methods for the description and generation of architectural structures. Birch et al. [BB1, BJ1] present techniques for the interactive modeling of buildings and architectural structures. They focus on the reduction of the number of parameters to a manageable size and to generate details procedurally. In order to have different windows a basic rectangular window can be successively subdivided into smaller windows. On these windows various window styles can be applied. They also have the possibility to generate bay windows. Legakis et al. [LD1] present a method for cellular texturing of architectural models. Upon the underlying geometry they perform texturing operations considering the interdependencies between cells for vertices (corners), edges and faces. Havemann et al. [HF1] use the combination of polygonal mesh modeling and subdivision surfaces, which they call *Combined BRep*, for modeling architectural structures like ornaments and window frames. They only need a coarse model of the structure and a view dependent refinement is done at runtime. Wonka et al. [WW1] introduce split grammars for the generation of buildings. They represent the façade as a non-terminal shape which can be further split into smaller non-terminal shapes leading in the last step of splitting to terminal shapes like windows, wall elements, etc.

Müller et al. [MW1] present a combination of the work from Parish et al. [PM1] and Wonka et al. [WW1] with astonishing results. But they do not provide any information for texturing the buildings. For architectural elements they use predefined models. A disadvantage of those three works is that the building description is not parameterized. After a building is created it can not be modified easy. They also recognized that spatial information upon adjacent elements is crucial. For resolving spatial queries they subdivide the building with an octree where intersection tests are computed. With the method presented in this paper spatial queries can be answered by hierarchical description of the building. Intersection tests are not necessary.

The last paper mentioned leads us to the important objective of adequate geometry representation. Mäntyä [M1] introduces different boundary models. Faces, edges and vertices are represented in a tree like structure not only describing the geometry model but also connection information between faces, edges and vertices.

3- Floor plan outline

Our main goal is to have arbitrary floor plan outlines for every level in a building. For this purpose we follow the aspect of a vector oriented approach. Additionally we have to take two aspects into account. First, we discovered that it is necessary to have basic architectural information, e. g. the location of projections, balconies, etc., on an early stage of modeling. The second is that we need spatial information of adjacent architectural structures both on a single level (*intra level*) and between two adjacent levels (*inter level*).

In the next two subsections we describe the methods which fulfill our demands.

3.1 - Single level

To obtain arbitrary floor plan outlines with necessary information about basic architectural structures, we compose convex 2D polygons to a floor plan outline. We refer to a convex polygon as a *floor plan module* or *fpm*. Each fpm represents an architectural structure in the façade with necessary information, e. g. type, material, basic geometric information, etc. To receive a single floor plan outline several fpms are combined. The basic method is given in [FB1]. In this method floor plan modules are connected via their edges. But floor plans like the Petronas Towers (see the outline in Figure 3) ground level were not possible. We extended this method to cope with this problem. Now two connected fpms can share a line (or a convex polygon) between two edges which build a concave form. This line can again have connections to other fpms. In Figure 3 we have a big rectangular fpm (fpm_1) where four triangular shaped fpms (fpm_2, …, fpm_5, depicted with dashed lines) are connected, each on one edge. The vertices where the relevant edges build a concave form are marked with a circle. The eight thick lines between the four triangles and the big rectangle form the new connection lines. Each of the eight lines has an arc fpm connected. Now a Petronas Tower like floor plan can be generated.

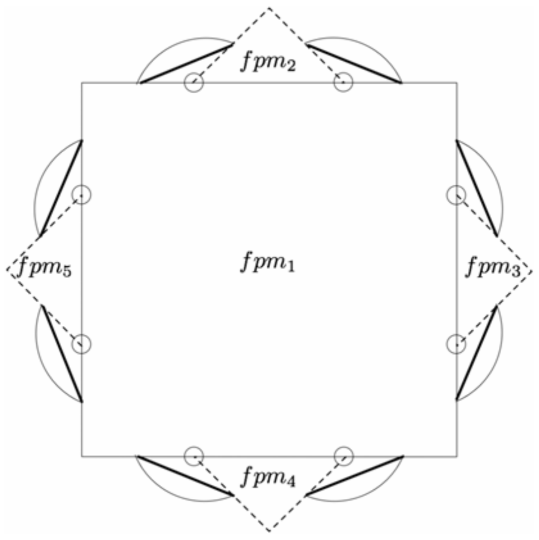

Figure 3: fpms representing a Petronas Towers like floor plan.

3.2 - Multiple levels

When extending a single floor plan to buildings with multiple levels, we start upon the fpms on the first level. The fpms are extended in a way that spatial information between different levels can be generated easily. For this purpose we support the following options for each fpm in the level to be extended. It can be:

- omitted, so no such fpm will be on the next level,
- fully extended to the next level or
- subdivided as shown in Figure 4, each convex combination of the subdivisions can be extended to the next level.

In [FB1] the last method (mentioned as “free”) for extending a fpm to a new level caused intersection operations. This led to the problem that spatial information was hard to compute. Therefore we replaced it with subdividing the fpm. The new method works as follows. The fpm in Figure 4 consists of the closed polygon ($\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_5$). The edges e_1, e_5, e_4, and e_3 are subdivided, generating the vertices $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$, $\mathbf{q}_4$. The fpm is then subdivided by the dotted edges ($\mathbf{q}_1$, $\mathbf{q}_2$) and ($\mathbf{q}_3$, $\mathbf{q}_4$) thus producing the following convex polygons: ($\mathbf{p}_1$, $\mathbf{q}_3$, $\mathbf{q}_5$, $\mathbf{q}_2$), ($\mathbf{q}_3$, $\mathbf{p}_5$, $\mathbf{q}_1$, $\mathbf{q}_5$), ($\mathbf{q}_5$, $\mathbf{q}_1$, $\mathbf{p}_4$, $\mathbf{q}_4$), and ($\mathbf{q}_2$, $\mathbf{q}_5$, $\mathbf{q}_4$, $\mathbf{p}_3$, $\mathbf{p}_2$). Any convex combination can be used for next level fpms. It is also possible to have multiple but disjoint convex combinations for next level fpms.

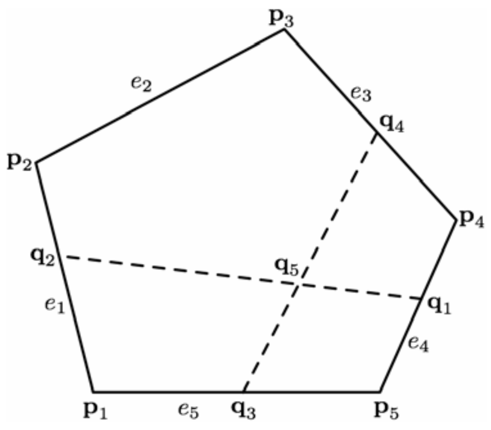

Figure 4: Subdivision of a floor plan module.

During the extension process, necessary connections will be retained. To generate structures like oriels or balconies, etc. the newly created fpms receive connections to fpms representing these structures, like the oriel on the left side on the second level shown in Figure 13. With our representation of the floor plan outlines the seams of adjacent structures in a single level and between two consecutive levels are emphasized clearly. This information is crucial if the geometry of interdependent structures has to be adapted at their seams. Also depicted in Figure 5 are the seams of adjacent structures. The lines in shades of red represent different types of floor connections to the previous level's ceiling. Ceiling connections are coded in shades of green. Connections between elements on the same level are blue colored.

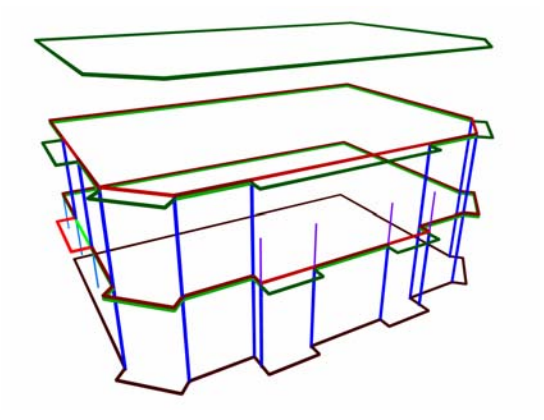

Figure 5: Different connection types between adjacent structures.

4- Façade description

Now that we have a representation for the coarse outline of a building, the next step is to generate the façade details. In the next sections we describe how we successively refine the coarse outline. From step to step we add more details to the façade.

4.1 - Corners and walls

The *basic walls* and *corners* are built from the coarse outline of every level's floor plan. Each vertex of the outline represents a corner whereas each edge is taken as a basic wall. Basic walls are further subdivided into *walls* and optionally *wall spacers*. As shown in Figure 13 the front wall (a basic wall) of the top level is subdivided into five walls and four spacers. Wall spacers and corners can be refined to stacks of arbitrary blocks. In Figure 13 the quoins consist of single blocks as well as the elements between the walls.

4.2 - Cornices

Additionally cornices can be applied to corners, walls, and wall spacers. The contour of a cornice is defined via a Logo-like [MA1] description language. We support two simple commands *angle* and *arc*. Angle draws a straight line (initially starting at the origin) of a given length and direction whereas arc draws an arc with a given radius and sweep angle. Figure 13 shows several different cornice types on every level of the building. Level one and two have cornices at the bottom and at the top. Level three has only a cornice at the top. The example in Figure 6 describes the cornice on the third floor of the façade in Figure 13. The profile of the cornice is shown in Figure 7.

```
Modifications
{
  angle, rel,  90, 0.3;
  angle, rel, -90, 10.0;
  angle, rel,  90, 0.6;
  angle, rel, -90, 11.0;
  angle, rel,  90, 0.6;
  angle, rel, -90, 0.6;
  angle, rel,  90, 0.6;
  angle, rel, -90, 2.0;
  angle, rel,  90, 2.0;
  arc, rel, 0, 4.0, -90;
  angle, rel,  90, 5.0;
  angle, rel, -90, 5.0;
  angle, rel,  90, 0.6;
  angle, rel, -90, 0.6;
  angle, rel,  45, 7.0;
  angle, rel, -45, 0.6;
  angle, rel,  90, 0.6;
  angle, rel, -90, 2.0;
  angle, rel, -84, 22.0;
}
```

Figure 6: Example cornice description.

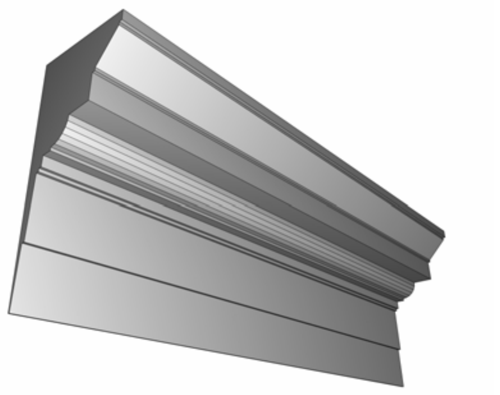

Figure 7: Example cornice profile.

4.3 - Doors and windows

Doors and windows are defined for walls only. First a rectangular hole is defined relative to a given wall. In the next step the four edges of the hole can be refined. Each edge can be replaced with a polygon defined by the description language we use for the cornices. The dotted line in Figure 8 shows a refined edge. Additional parameters for offsets to the left d_l, right d_r, and top d_t can be applied. In Figure 8 the original edge ($\mathbf{f}_l$, $\mathbf{f}_r$) is refined to the dotted line between the edge (**l**, **r**).

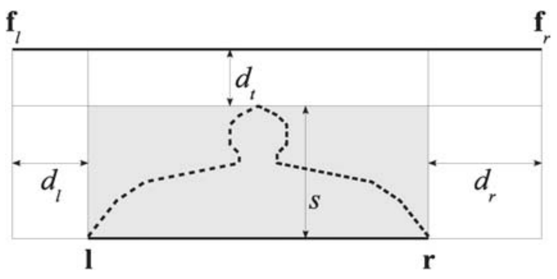

Figure 8: Refinement for a top edge of a window or a door.

The inner part of the hole is now filled with a frame for each refined edge. Figure 9 shows different frame types that can be used: single and double cornice, bricks, and simple. The ledges (the bottom edges) of all four window frames are simple types. The top edges of the two frames on the left side have a special cornice refinement. It is a combination of two different cornices. Both top frames are additionally refined with bricks and the bottom right frame has a single cornice refinement.

The frames for the window panes are kept very simple. They run along the inner edges determined by the above defined frames, have a central cross, and are of type cornice. All windows in Figure 9 and Figure 13 have such window pane frames.

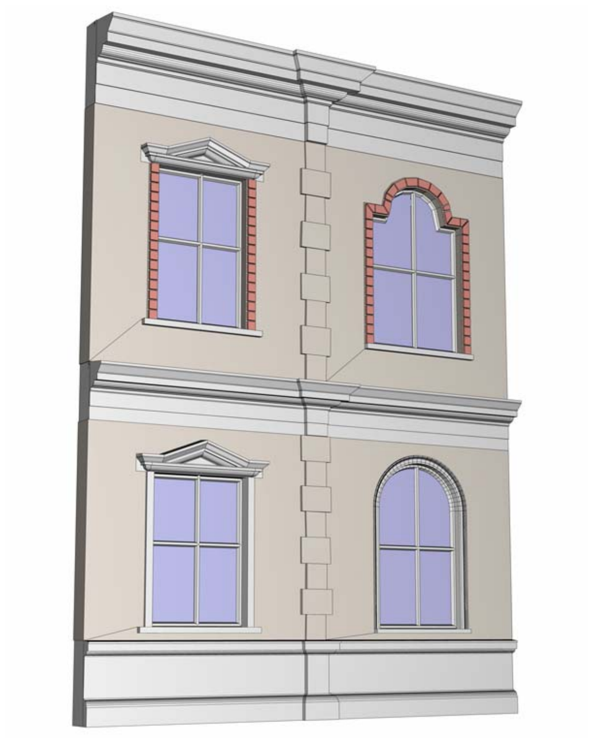

Figure 9: Refined window borders with different styles.

5- Roof description

Most algorithms for roof generation work on arbitrary floor plan outlines. Felkel et al. [FO1] present a robust algorithm for automatic roof generation. The floor plan polygon is shrunk to a kind of skeleton which is used to create the gables. Laycock et al. [LD2] and Müller et al. [MW1] present another method for the automatic generation of roof models. They build the roofs according to the building footprint. Laycock et al. partition a building footprint into rectangular pieces where main parts may also overlap. Then each piece is given a roof type and adjacent roofs are connected appropriately. The limitation is the need for rectangularity and that the roofs all take place at the same level. Müller et al. merge main building parts where each part has its own roof type. Then they also generate appropriate roofs. A major problem of both approaches is the intersection of the building pieces forming the roof.

In our case we use the individual floor plan modules to control the appearance of the roof. The appearance of the entire roof is determined by the single floor plan modules. Every top level fpm has its own roof type information. Actually the three roof types *flat*, *pent*, and *gable* roof are possible. Figure 10 shows the basic procedure for pent and gable roofs. The fpm represented by ($\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_5$) is subdivided by the line ($\mathbf{q}_1$, $\mathbf{q}_2$). In this example $\mathbf{q}_2$ equals $\mathbf{p}_2$. For the pent roof an additional angle $\boldsymbol{\alpha}$ at point $\mathbf{q}_2$ or $\mathbf{p}_2$ describes the roof inclination. For the gable roof the line represents the gable at a given height h. Additionally the line can be subdivided by the points $\mathbf{r}_1=(\mathbf{q}_1-\mathbf{q}_2)\,a$ and $\mathbf{r}_2=(\mathbf{q}_1-\mathbf{q}_2)\,b$ to receive a hipped roof with height h_1 and h_2.

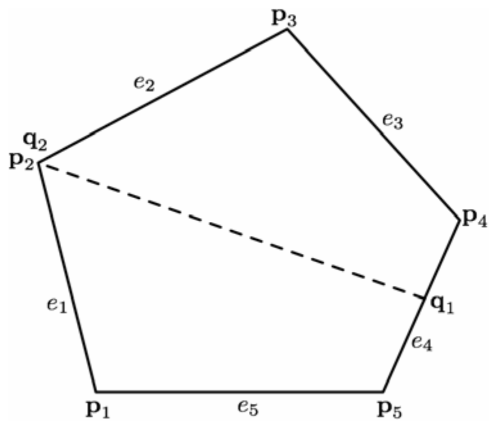

Figure 10: Start for pent and gable roof.

When the entire roof is generated, the roof information of adjacent fpms are combined into a single roof. The step from single fpm roofs to a combined roof is shown in Figure 11. With our method no complicated intersections have to be computed and we are not limited to roofs on a single level.

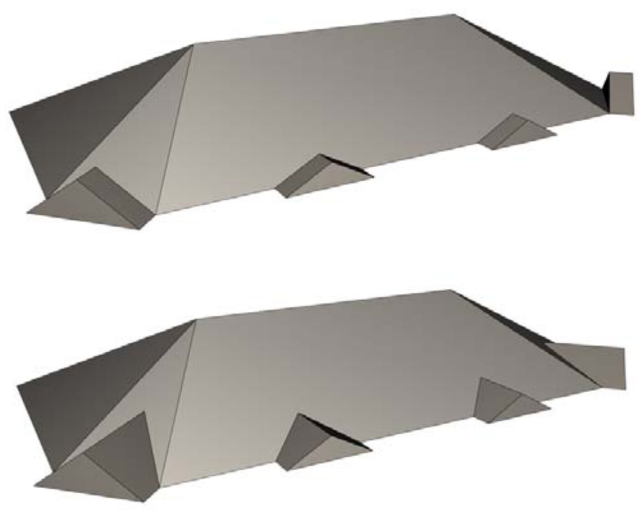

Figure 11: Step from roof parts to the entire roof.

6- Geometry engine

In the previous sections, we explained our method of representing building outline and façade elements, including fpms, walls, corners, windows, etc. In this section we put all information together and introduce our hierarchical description of the façade in a typed graph. We developed a tool, the geometry engine, which takes the data and produces the detailed geometry.

6.1 - Hierarchy

An entire hierarchy combining floor plan, roof generation and façade representation for a particular building façade is depicted in Figure 12. This depicts the relationships between adjacent façade elements. The nodes also include essential geometric information.

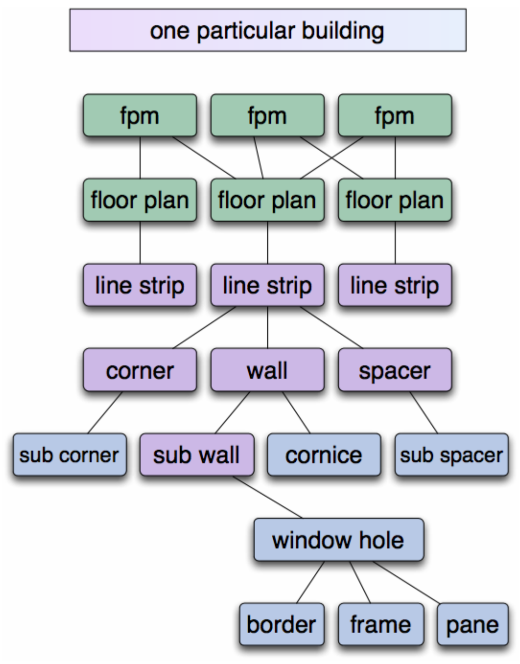

Figure 12: Hierarchy for a particular building façade.

6.2 - Hierarchy traversal

Now we generate the geometry upon the information given by the hierarchy. For that reason we start at the root of our hierarchy (which represents the entire building) and traverse it down to its leaves. While traversing the graph we gather information about the geometry in every step which is equivalent to a step by step refinement of the geometry.

6.3 - Depth order relation

A major aspect in our concept is that adjacent structures do not overlap, so they have to be adapted. Our *depth order relation* arises directly from the graph. Deeper steps in the hierarchy have a higher priority and hence they have the permission to modify the geometry of previous steps, e. g. the door or window holes modify the walls.

6.4 - Same level order relation

The nodes in the hierarchy at the same depth also have to be taken into account. For instance walls, corners and spacers are all at the same level. Here the corners and spacers have a higher priority and the wall geometry is modified to fit the corners' or spacers' geometry.

Finally we build the roofs but they are treated in a special manner because they do not fit in the relations described before. If in the same level they are adapted according to the order given by the connection tree of the fpms, e. g. if one roof is adjacent to another roof then the roof with lower priority is adapted to fit the adjacent roof. If one roof is adjacent to other structures such as walls, corners, spacers, doors or windows then these structures are adapted to fit the roof. For example in Figure 13 the projection on the left side is only present in the first two levels. A roof on top of this projection will interfere with the wall on the third level and therefore the wall has to be adapted.

The above mentioned hierarchical information gathering process can be interrupted at any depth and the geometry will be created that far. This allows us to produce a more or less detailed geometry for a façade which can be used for a level of detail visualization, which is useful for real-time applications.

At the moment we can produce the geometry in either MEL (Maya Embedded Language) or RenderMan format. This allows the user to import the model into Maya or render it with a RenderMan compliant renderer.

7- Results

For the typed graph and the geometry engine we implemented a prototype in Python. At the moment the coarse building outline has to be provided as Python code. Very simple methods take the role of the style library. They subdivide the coarse building and produce the detailed façade description. The prototype has been tested on an AMD Athlon XP 2000+ with 1 GB memory.

The examples in Figure 13 and Figure 14 were generated with our prototype and rendered in Maya. The first example consists of 81983 triangles and 3439 individual objects. The creation time for the example was 26 seconds. The second example consists of 556736 triangles and 7971 individual objects. The creation time was 107 seconds.

8- Conclusion and future work

In this paper we proposed a representation of a detailed façade description in a hierarchical typed graph. We presented the versatility of this representation regarding the ability to change the façade appearance quickly and to automatically generate exact fitting geometry. The method is applied in a framework for rapid modeling of building façades which is still under development.

A limitation of the presented method is that organic looking structures can not be described. Also the floor plan description is limited to polygons. This could be solved with splines but would increase the time for development.

Different façade styles, its proportions, etc. will be represented as rules in a graph structure. With a graphical user interface the designer draws the coarse building outline and the graph is queried to produce the detailed façade geometry. To have a more realistic appearance for the walls, we are developing a texture generator to create adapted brickworks. Additionally, new roof types and balconies will be added.

7- References

[A1] D. Arnold. Economic reconstructions of populated environments - progress with the charismatic project, 2000.

[A3D1] Autodesk. 3D Studio Max, 2006. http://www.autodesk.com/3dsmax.

[AM1] Autodesk. Maya, 2006. http://www.autodesk.com/maya.

[BB1] P. J. Birch, S. P. Browne, V. J. Jennings, A. M. Day, and D. B. Arnold. Rapid procedural-modelling of architectural structures. In Proc. of the 2001 conference on Virtual reality, archeology, and cultural heritage, pages 187-196. ACM Press, 2001.

[BJ1] P. Birch, V. Jennings, A. M. Day, and D. B. Arnold. Procedural modelling of vernacular housing for virtual heritage environments, 2001.

[FB1] Dieter Finkenzeller, Jan Bender, and Alfred Schmitt. Feature-based decomposition of façades. Proc. Virtual Concept 2005.

[FO1] Petr Felkel and Stepan Obdrzalek. Straight skeleton implementation. In Laszlo Szirmay Kalos, editor, 14th Spring Conference on Computer Graphics (SCCG'98), pages 210-218, 1998.

[GP1] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Undiscovered worlds - towards a framework for real-time procedural world generation. In Proc. of the Fifth Intern. Digital Arts and Culture Conference, 2003.

[GP2] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Real-time procedural generation of 'pseudo infinite' cities. In GRAPHITE '03: Proc. of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, pages 87-ff, New York, NY, USA, 2003. ACM Press.

[HF1] Sven Havemann and Dieter Fellner. A versatile 3d model representation for cultural reconstruction. In Proc. of the 2001 conference on Virtual reality, archeology, and cultural heritage, pages 205-212. ACM Press, 2001.

[LD1] Justin Legakis, Julie Dorsey, and Steven Gortler. Feature-based cellular texturing for architectural models. In Proc. of the 28th annual conference on Computer graphics and interactive techniques, pages 309-316. ACM Press, 2001.

[LD2] R. G. Laycock and A. M. Day. Automatically Generating Roof Models from Building Footprints. The 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2003.

[M1] Martti Mäntyä. An Introduction to Solid Modeling. Computer Science Press, Maryland, 1988.

[MA1] Anne MacDougall, Tony Adams, and Pauline Adams. Learning Logo on the Apple II. Simon & Schuster, 1984.

[**MW1**] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. ACM Trans. Graph., 25(3):614-623, 2006.

[PL1] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. The algorithmic beauty of plants. Springer-Verlag New York, Inc., New York, USA, 1996.

[PM1] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In Proc. of the 28th annual conference on Computer graphics and interactive techniques, pages 301-308. ACM Press, 2001.

[S1] Uwe Schöning. Theoretische Informatik kurz gefasst. BI Wissenschaftsverlag, 1992.

[WW1] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. ACM Trans. Graph., 22(3):669-677, 2003.

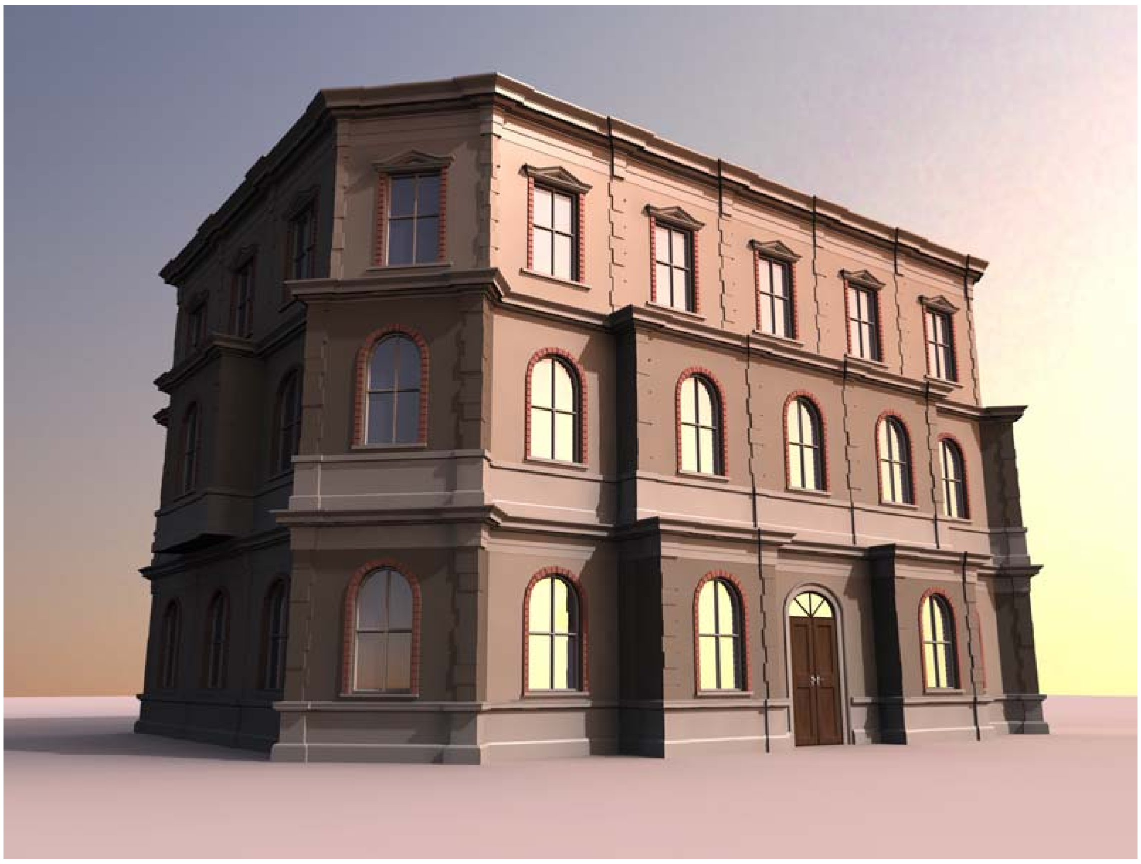

Figure 13: Example façade.

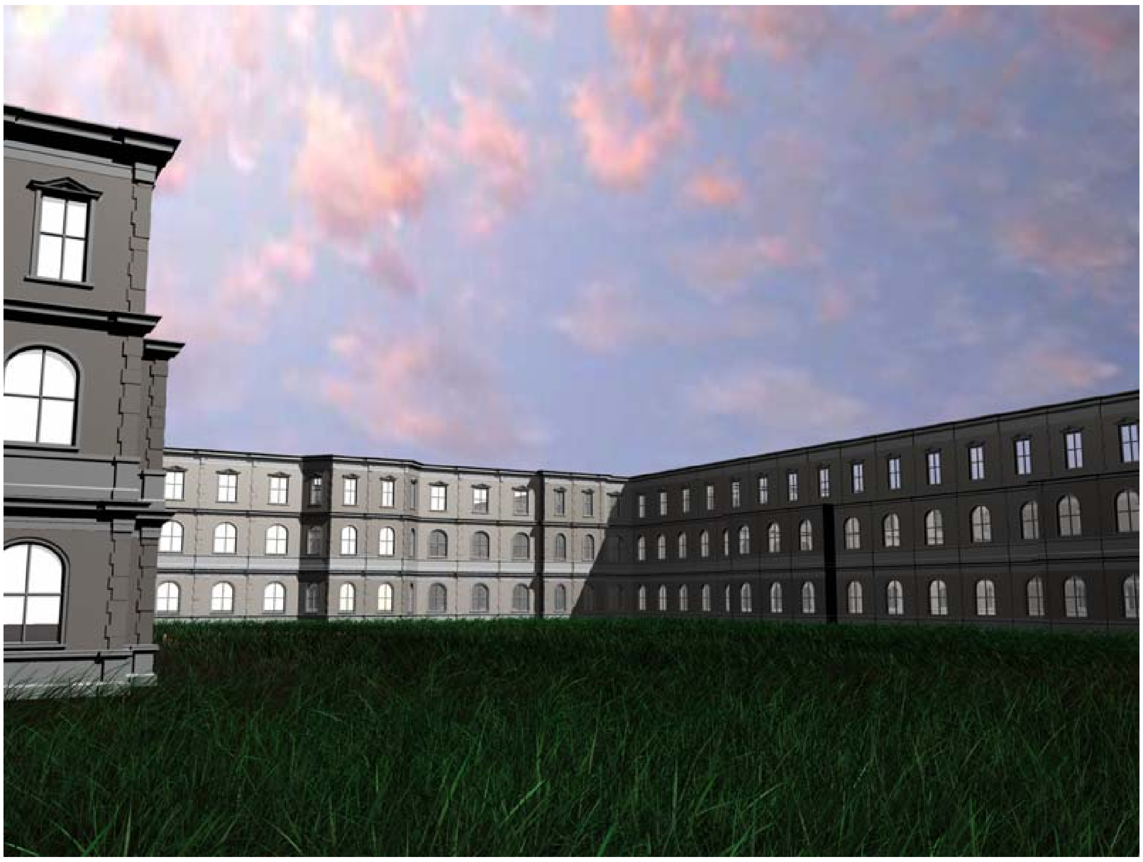

Figure 14: Example façade of a large building.