Feature-based decomposition of façades

Dieter Finkenzeller, Jan Bender, Alfred Schmitt

University of Karlsruhe
Institut für Betriebs- und Dialogsysteme
Am Fasanengarten 5
76128 Karlsruhe
Germany
Phone: +49(0)721 608 3965
Fax: +49(0)721 608 8330
E-mail : {dfinken|jbender|aschmitt}@ira.uka.de

Abstract: Due to advances in computer hardware, virtual environments become significantly larger and more complex. Therefore the modeling of virtual worlds, e.g. for computer animation and games becomes increasingly time and resource consuming.

In architectural settings façade features are influenced by the underlying geometrical structure or even by other façade structures, e.g. façade edges made of large stones influence the adjacent walls. To achieve an aesthetic look of the façade adjacent structures must be seamlessly aligned.

The modeling of such structures is a tedious work. With our approach only a few basic parameters are needed to create highly detailed façades. This relieves the designer of the burden of difficult modeling tasks and gives him more high level control.

In this paper we present a strategy for a floor plan representation that permits arbitrary floor plan outlines. This simplifies the roof generation for different roof types in an easy way to achieve an aesthetic goal. Based on the floor plan representation we describe a hierarchical decomposition of architectural façade features. With an order relation on it we represent the interdependencies between the façade features and introduce a geometry generator for them.

With our approach every building in a large VR city will look different but can have a high level on architectural details.

Keywords: virtual architecture; procedural modeling; geometrical algorithms and data structures; computer-aided design; virtual reality.

1- Introduction

Nowadays, the powerful computer hardware, especially in the field of graphics hardware, allows complex and highly detailed scenes in computer games, animations and virtual environments. In massively multiplayer online (role-playing) games, like 'World of Warcraft' [1], there are many different locations all with a different design. In such virtual environments we will concentrate on generating highly detailed building façades.

Creating a building using a 3D modeling program, e.g. Maya [2] or 3D Studio Max [3] is a very time consuming work. To obtain a realistic and pleasant appearance adjacent architectural structures have to be adapted in a way that their connecting borderlines match. The modeling of such structures is extremely complex, time demanding and requires extensive human resources. Hence the desire arises to automate the process as much as possible. One possibility to achieve this goal is to use procedural methods which help the designer to generate complex and detailed structures. Also repetitive modeling tasks, e.g. window frames with similar design patterns, are automatically arranged by these methods.

Consequently these methods will relieve the designer of the burden of the tedious recurrent and difficult modeling task. Hence it allows him to spend more time to model on a higher level. But surely the designer should always have the ability to modify the details as needed.

The main problem is to detect the connecting borderlines of adjacent architectural structures. As we have seen they appear between walls and doors or windows or quoins – this all happens in a single floor (intra floor connections). But also the "inter floor connections", like balconies with their sustainers, oriels or bay windows, have to be considered. To solve this problem we use a hierarchical decomposition representing the architectural structures. To optimize this decomposition, it is based on a special floor plan representation which also allows arbitrary floor plan outlines and an easy way to create different roof types.

The outline of this paper is as follows. We discuss related building and pattern generation in section 2. Then we present our method for floor plan representation in section 3. Based on the results of the previous section we describe the hierarchical decomposition of building façades in section 4. In section 5 we present how the floor plan representation can be applied in creating different roof types. We then introduce

a geometry factory to produce façades and show results achieved with our method in section 6. In section 7 we give an overview of the implemented prototype. We conclude in section 8 and discuss future work in section 9.

2- Related Work

There are several research papers describing the procedural modeling of cities [4], [5], [6] and [7]. They are mainly focused on generating whole cities on the basis of rule systems. Parish and Müller [4] distinguish between several parts of a city, e.g. commercial and residential regions. With a set of rules and constraints they build motorways between these parts connecting them. Roads are leaving the motorways and branch to several directions providing access to the buildings. The generation of single buildings is also rule based, e.g. the façades are procedurally textured. In [5] and [6] Greuter et al. focus on a real-time generation of procedural cities. In these papers the main aspect concentrates on building a large amount of low to medium detailed buildings.

In contrast, the authors of the papers [8], [9], [10] and [11] present specialized methods for the description and generation of architectural structures. Birch et al. [8] describe an interactive procedural modeling technique for buildings. Hence their main goal is to reduce the number of parameters needed for the description to a manageable size and to generate the details procedurally. Legakis et al. [10] present a method for cellular texturing for architectural models. They extract features of an underlying geometry. Upon these features they perform texturing operations in an order that considers the interdependencies between cells of adjacent patterns. In [11] Havemann et al. describe the combination of polygonal mesh modeling with subdivision surfaces; which they call *Combined BRep*. They use the Combined BRep for modeling architectural structures like ornaments and window frames. Their power is that they only need a coarse model of the structure and the view dependent refinement is done at runtime.

One of the most important objectives is the adequate representation of the geometry. This is directly reflected by the choice of the underlying data structure. Mäntylä [12] and Corney et al. [13] describe similar techniques representing geometry. Mäntylä [12] introduces different boundary models which represent faces in terms of explicit nodes of a boundary data structure. The polygon-based boundary model represents faces as polygons, each consisting of a set of coordinate triples. A solid is defined by its faces and their interdependencies are only implicitly given. A disadvantage is the redundant use of vertex coordinates, e.g. a corner of a cube has a single vertex but it reappears in every edge of that corner. This leads to the improved representation of vertex-based boundary models. The problem of redundant vertices is solved as vertices are introduced as independent entities. Now a face is defined by a set of vertex identifiers given either in clock wise or counter clock wise order to distinguish the two sides of a face. Next Mäntylä introduces edge-based boundary models. Again vertices are represented as single entities. But now they are used to define edges consisting of two vertices. A face boundary is given by a closed loop of edges. The edges are given in a certain order to distinguish the two sides of a face. The vertices of a certain face are implicitly given by its edges. The winged-edge data structure extends the edge-based boundary model. Baumgart [14] was the first to use this data structure which also takes the loop information in edge nodes into account. Corney et al. [13] describe vertex-edge graphs where the vertices and edges of the geometry are represented as nodes and edges in a graph. An alternate graph dual to the previous graph is the face-edge graph. Now the faces are considered as the nodes and the edges are the edges connecting two faces. With these representations they cover neighborhood relations between the faces, the edges and the vertices.

3- Floor plan representation

In this section we describe our representation of floor plans and its impact for our purpose.

Figure 1 : A façade with a middle projection.

3.1 – Floor plan definition for one level

When constructing building façades we want to be able to have arbitrary floor plan polygons. For our purpose we define that a floor plan for one level is exactly one polygon; disjoint polygons are not allowed. If we work with a single concave polygon as a floor plan additional information for some sets of connected edges is necessary. For instance the building façade in figure 1 has a projection in the middle. The walls to the left and to the right show a different design. So it is crucial to have the ability to distinguish different façade structures in an early phase. We solve the problem in composing several convex 2D shapes to build the floor plan. As shown in figure 2 we have the floor plan (the thick black surrounding line) consisting of several convex 2D polygons (the hatched polygons). Such a single convex polygon is stored in an entity called *floorPlanModule (fpm)* which also contains additional data, e.g. façade type, height, material etc. The polygon itself is represented by an edge-based format comparable to the one of Mäntylä [12]. A set of connected fpms together form a floor plan. The connections are given in a tree like order, starting at a distinguished fpm. We will need this *connection tree* later when generating

geometry. This allows us to create arbitrary concave floor plans.

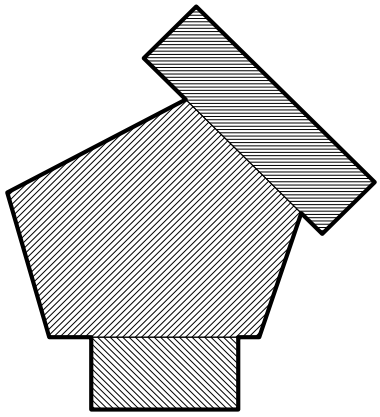

Figure 2 : Floor plan consisting of convex polygons.

A connection between two fpms is a single entity. This incorporates the following:

- the two connected fpms;
- their connecting edges;
- the definition of the connecting interval on one of the fpm's edge as a start and end point of the normalized interval [0, 1].

In figure 3 two fpms are shown. The interval where fpm2 is connected starts at 0.1 (pS) and ends at 0.8 (pE) on the edge e0 of fpm1. The connected fpm (i.e. fpm2 in figure 3) is transformed (scaled, rotated and translated) to match the connection.

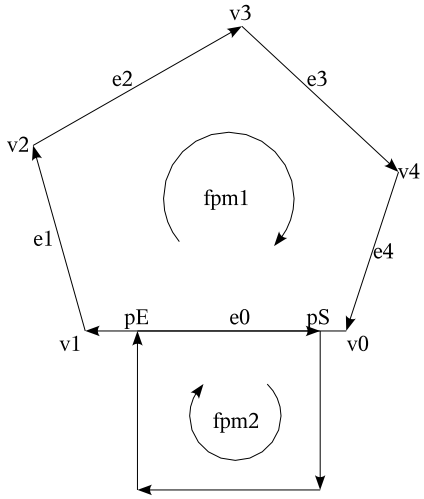

Figure 3 : Two connected floor plan modules.

The information about the connection is once stored in the polygon's edge of each fpm and even in the fpms themselves. As a result every edge knows the edges of all fpms connecting to this edge and every fpm knows all direct neighbors.
Now we can join the fpms to a build a union and as a result we get one boundary polygon. This polygon consists of entire or partial edges of the original fpms. Connected edges belonging to a fpm are classified as features. As we can see in figure 4 a fpm can have several features. The big polygon in the middle has two features (blue or solid lines) whereas the others just have one.

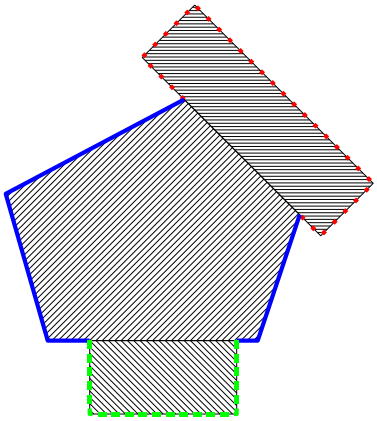

Figure 4 : Features of fpms.

This decomposition enables us to extend the floor plan arbitrarily, e.g. with features like projections, oriels, bay windows etc. A detailed overview of different façade structures for buildings – especially in Germany – of the late 19th century is given in [15]. As we will see next this is very important if we will have different floor plans for different levels.

3.2 – Floor plan definition for adjacent levels

Now that we can describe an arbitrary floor plan for a single level we advance one step further and define the connection between two adjacent levels. The connection starts at a single fpm of the lowest level. There are three distinguished types of connections:

- *full*: the entire polygon of the current fpm is exactly the same in the upper level.
- *aligned*: partial edges of the current fpm polygon are used in the upper level and are extended.
- *free*: the upper level consists of arbitrary polygons inside the current fpm polygon.

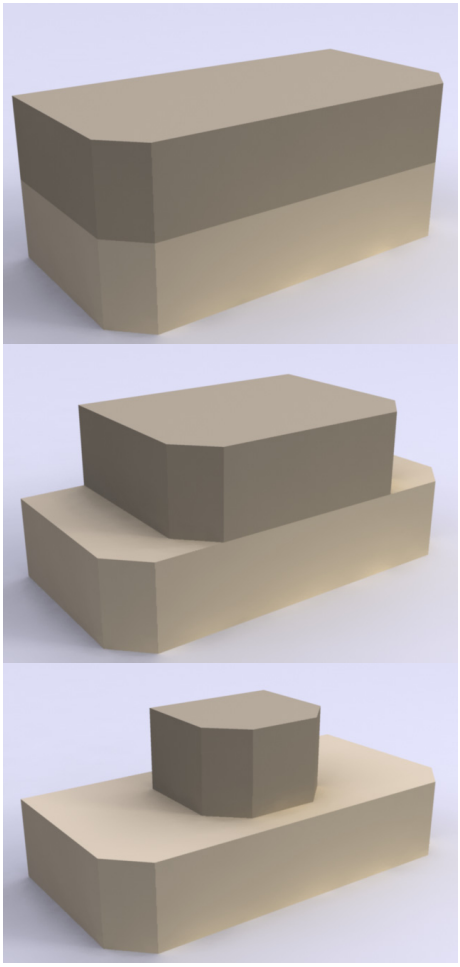

Figure 5 : Connections types between fpms (full, aligned, free).

Figure 5 depicts the three connection types.
It is possible to combine different connection types at the same level. The types *aligned* and *free* can be combined if the polygons do not overlap. The type *full* can not be combined with one of the others. A result of the floor plan representation is presented in figure 6. It shows arbitrary floor plans for different levels.

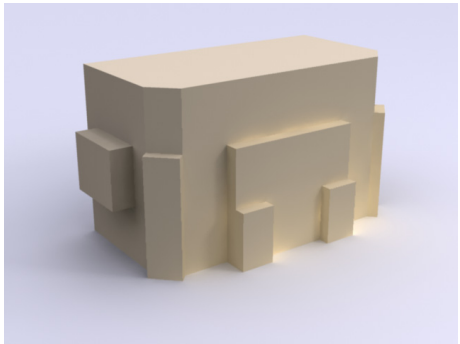

Figure 6 : Arbitrary floor plans for different levels.

3.3 – Adjacent structures

With our representation of floor plans the seams of adjacent structures in a single level (intra adjacent structures) and between two consecutive levels (inter adjacent structures) are emphasized clearly. This information is crucial if the geometry of interdependent structures at their seam is adapted. Figure 7 shows an example of interdependent structures with all potential critical seams marked. The intra level dependencies are marked red whereas the inter level dependencies are blue.

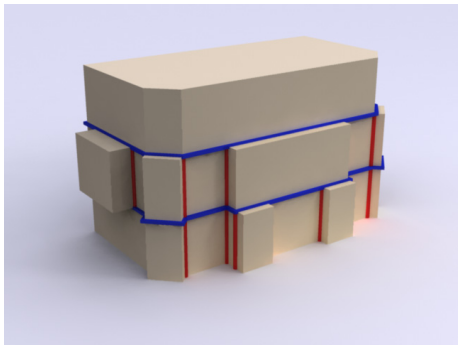

Figure 7 : Seams of adjacent structures.

These rectangular structures are our *coarse features* of the façade. In the next section we describe the refinement of these coarse features to achieve finer and more detailed features.

4- Façade representation

In the previous section we presented the description for arbitrary floor plans and a method for connecting them. With this technique we are able to distinguish different adjacent rectangular structures on the façade. This is quite a coarse classification of the façade. In this section we present a step by step refinement of the coarse structures.
In the following sections we introduce our procedure of subdividing the coarse features to build detailed façade elements (detailed features) as corners, walls, doors with frames and windows with frames. First, basic corners and walls are built directly from the coarse features of the floor plan. Then these corners and walls can be subdivided. Afterwards, doors and windows with their frames are integrated into the walls.

4.1 – Corners and walls

For every line strip of a fpm simple corners and walls are created with user adjustable parameters like thickness, color etc. The edges of the line strips are taken as the basis for the walls with a certain thickness. The corners are treated equally. Metaphorically speaking, the line strip and its vertices are given a certain thickness and extruded along the axis perpendicular to the line strip. After this first step we have a horizontally subdivided structure like the first illustration in figure 8.
In a consecutive step a wall between two corners can be further subdivided. The subdivided walls can have optional spacers in between. The middle illustration in figure 8 shows this issue.
In a last step the spacers, corners and walls can be vertically independent subdivided (see last illustration in figure 8). Every subdivision has its own parameters. If not overwritten it inherits common parameters from its successor.

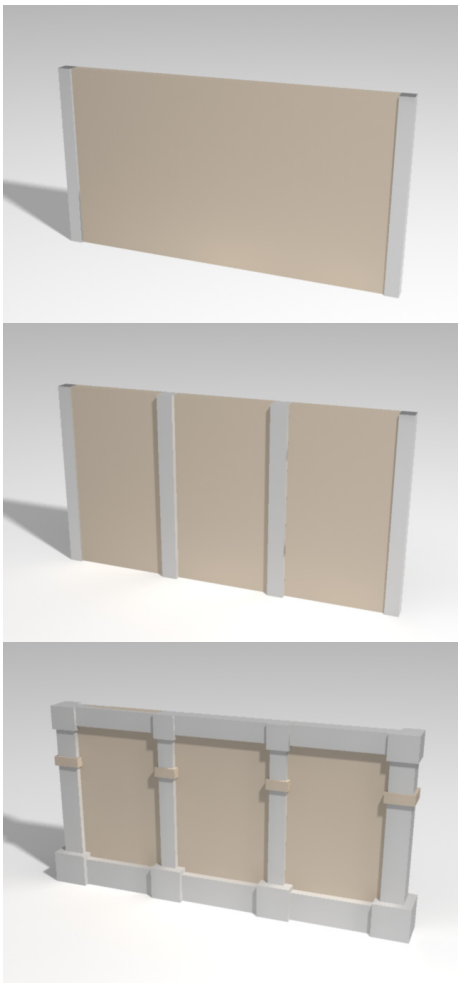

Figure 8 : The subdivision process.

With every step the parameters for the resulting finer structure can be adjusted, e.g. the material, the thickness etc. can be changed.

4.2 – Doors and windows

The next big step in the façade refinement process is the integration of doors and windows. Every wall, e.g. the three subdivided walls in the last illustration of figure 8, can have a door or a window.
Before creating a door or a window, a rectangular hole is positioned in relation to the left bottom corner of a single wall. Its parameters are the distances in percentage to the left, right, bottom and top. An example is given in the first illustration in figure 9 where we defined holes for three windows and a door.
Next the four edges of the hole can be refined with open polygons or NURBS (Non-Uniform Rational B-Splines). This process can be applied to each edge independently allowing us to achieve a broad variety of door and window holes. In the second image in figure 9 the top edges of all four holes were refined with an arc.
In the following step edges or refined edges can be further refined with borders to build frames for doors and windows. Geometrical parameters for the frames are thickness and depth. The thickness describes the amount the frame penetrates the surrounding wall and the depth parameter represents the depth of the border perpendicular to the wall. Figure 10 is a schematic drawing of that issue. As a result the frame can also project the wall.

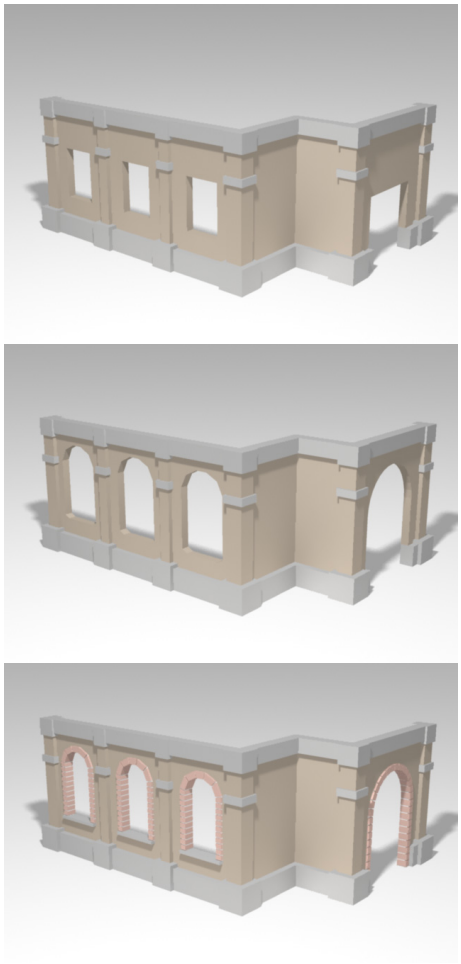

Figure 9 : Door and window refinement.

The frames themselves can be individually subdivided. The distance between adjacent frame tiles, their depth and their thickness can be adjusted. Because of this it is possible to create frames consisting of bricks with a larger stone in the middle. To achieve more realism the mortar between the frame tiles is also created with appropriate parameters. The last image in figure 9 shows subdivided frames with mortar in between.

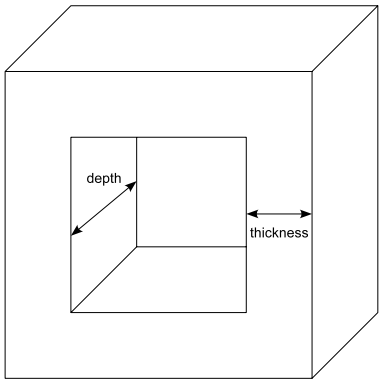

Figure 10 : Thickness and depth of a frame.

Now the doors and windows filling the holes have to be created. This is not implemented yet but scheduled for the future.
These techniques allow a vast amount of different styles for door and window frames.

5- Roof generation

Roof plan generation is quite a difficult process. Most roof algorithms compute roofs for arbitrary floor plan polygons. The common goal is to create the gables for the roof. For this purpose the floor plan polygon is used to determine the gable. It is shrunk to a kind of skeleton which is used for the gable. The algorithms mainly differ in the methods used to shrink the polygon towards the skeleton. Felkel et al. [16] present a robust algorithm for automatic roof generation.
But for our purpose we do not have and we do not need to work on arbitrary polygons because fortunately we can make use of the floor plan representation for the roof generation. The necessary information is already given in the floor plan's fpms we have defined in section 3. Every fpm has an orientation. It is represented as a vector positioned inside the polygon. The vector defines the gable's direction and the magnitude of the vector defines its length. Additional information of the fpm gives the height of the roof. We also know about the connections between the fpms and use this information to build aesthetic roofs of different types; we can even mix different roof types. The types we are currently working on are flat, pent and gable roofs. Next we present the information needed to build one of the roof types:

- For the flat roof we directly use the fpm's line strips to build small surrounding walls for instance.
- The pent roof makes use of the vector of its fpm and additionally needs an angle for the inclination.
- The gable roof also considers the fpm's vector, its length and the roof height.

Figure 11 shows the three roof types and the information used for building the roof (vector, angle, height).
This method gives us the ability to build roofs on every fpm which has no connections to an upper level. These fpms do not need to be on a single level.

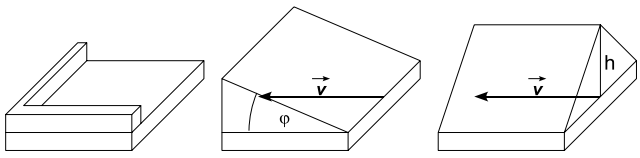

Figure 11 : Three roof types: flat, pent and gable roof.

6- Geometry factory

In the previous sections we explained our method of representing features to generate façades. In these sections we developed as a basis a symbolic representation for describing façades. But at least our goal is to create geometry. In this section we present a mapping from the symbolic representation to the geometry.

Hierarchy

An entire hierarchy combining floor plan, roof generation and façade representation is reflected in figure 12. This depicts the symbolic description about the relationships between adjacent features including essential geometric information.

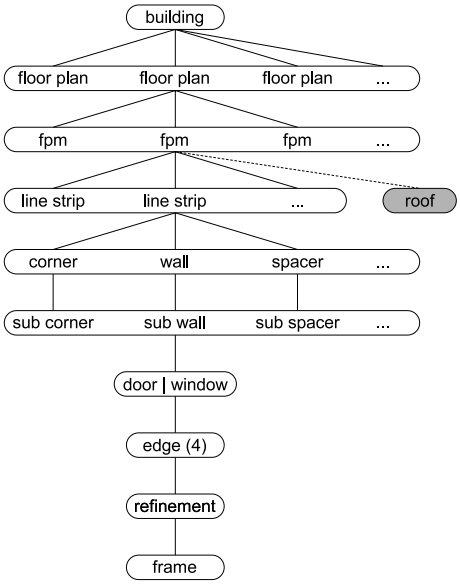

Figure 12 : Hierarchy.

Hierarchy traversal

Now we generate geometry upon this information. For that reason we start at the root of our hierarchy, the building, and traverse it to its leaves. While traversing the hierarchy tree we gather information about the geometry in every step which is equivalent to a step by step refinement of the geometry.

Depth order relation

We characterize this order relation as a depth order relation because deeper steps in the hierarchy have a higher priority and hence they have the permission to modify the geometry of lower priority steps.

Same level order relation

Nodes in the hierarchy at the same depth are also 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.
As an example we consider the process of applying a window hole into a wall. The window has the following parameters: width, height, position in the wall; a refinement

for all edges and a frame with a certain thickness, consisting of a number of bricks. First the geometrical information about the wall, its horizontal subdivisions respectively, is taken. Then the appropriate window hole is computed, taking into account the window hole rectangle, the refinement of its edges and the frame. In the following step we build the geometrical difference between the wall and the computed window hole. This ensures that we have enough space to apply the window frame into the wall, so geometry will not overlap. When building the frame for the window every single brick is created and positioned automatically.
The above mentioned hierarchical information gathering 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 building façade which can be used for level of detail visualization. This is useful for real-time applications.

7- Implementation

For the implementation of our prototype and also for the final version we want to have the software platform independent. So we use Python as the main programming language. To keep the software fast we chose to implement time consuming parts in C/C++ using platform independent classes and functions. Then with the help of Boost [17] we achieve a seamless interoperability between C/C++ and Python [18]. For the graphical user interface we rely on wxPython [19] which is the Python version of wxWidgets [20].
Currently we developed Python modules for the floor plan (single and layered floor plans) and the façade representation implementing the geometry factory with the ability to generate different levels of detail. At the moment the modules are controlled by a simple command line tool. We can produce the geometry in either MEL (Maya Embedded Language) or RenderMan format. This allows the user to process his work with the powerful modeler Maya and renderer RenderMan.
Figure 14 shows a model of a façade for a single level with detailed door and window frames, rendered with the RenderMan compliant renderer 3Delight [21].

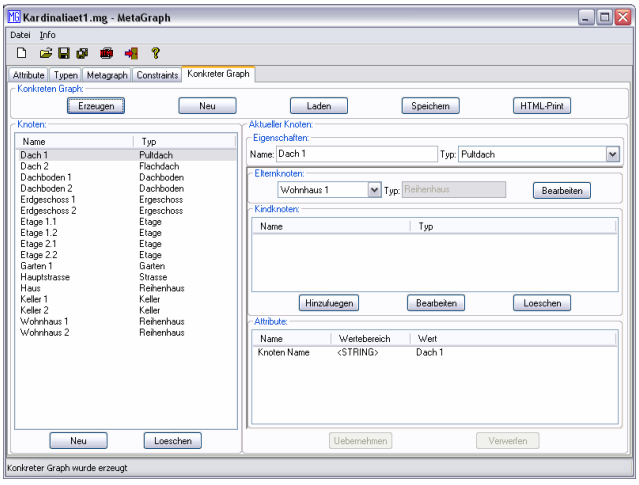

Figure 13 : Graphical user interface

On a second branch we are working on a graphical user interface which will replace and extend the command line tool. The underlying structure of the user interface is based on a graph structure in which we will map our decomposition. This is still work in progress and future work. At the moment we are already able to fill the graph with dummy data via the user interface. The user interface enables the user to traverse the graph interactively and manipulate the data at any level. In figure 13 we see a screenshot of the user interface. On the left side titled “Knoten” (node) all nodes of the graph are listed. Each of them can be selected and its data is displayed on the right side “Aktueller Knoten” (active node). On the right side the node’s data can be manipulated, e.g. change its attributes values and even attributes can be added or removed.

8- Conclusion

In this paper we presented a method for creating highly detailed building façades at a low cost of human resources. In section 3 we described floor plans consisting of single convex polygons, the floor plan modules. Hence we are able to distinguish coarse façade features, e.g. projections, oriels, bay windows etc., in an early phase of façade generation. Then we presented a hierarchical decomposition of walls, corners, doors and windows with the possibility of refinement in section 4. In section 5 we discussed how the floor plan modules can be used for roof generation. In section 6 we presented our geometry factory. It takes the information from the floor plan generation, the façade representation and the roof generation to produce geometry for the different architectural structures. The main aspect is that the geometry of adjacent structures is automatically adapted at a very high level of detail, relieving the burden of a tedious modeling task from the designer. At last we discussed in section 7 the implementation of our prototype.

9- Future work

In the near future we will extend our work with detailed doors and windows, filling the door and window holes in the walls. We are going to include more façade elements like different types of balconies. Additional roof types will be added as well. At the moment we only generate optimized geometry. In the future we will use this geometrical information to create textures exactly fitting the geometry. Then we will automatically generate shaders for patterns of brick walls, even with bump mapping and displacement shading. We want to extend the output formats by OpenGL, so that our method can be used for real time applications.
As mention in section 7 we are currently working on a graphical user interface relying on a graph structure. Now we have to connect the user interface with our Python modules. This enables an easy manipulation of the data and should also help us in the automatically generation of buildings, resulting in a breakthrough in façade modeling. Different building façades in different styles could emerge in some seconds. Highly detailed virtual cities of different ages with different styles could be modeled in a few seconds.

10- References

[1] World of WarCraft. http://www.worldofwarcraft.com © 2005 Blizzard Entertainment (seen march 2005)

[2] Maya. http://www.alias.com/maya (seen march 2005)

[3] 3D Studio Max. http://www.discreet.com/3dsmax (seen march 2005)

[4] Parish Y., Müller P. Procedural Modeling of Cities. International Conference on Computer Graphics and Interactive Techniques, 2001.

[5] Greuter S., Parker J., Stewart N., Leach G. Undiscovered Worlds – Towards a Framework for Real-Time Procedural World Generation. Fifth International Digital Arts and Culture Conference, Melbourne, Australia, 2003.

[6] Greuter S., Parker J., Stewart N., Leach G. Real-time Procedural Generation of `Pseudo Infinite' Cities. International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, 2003.

[7] Arnold D. Economic reconstructions of populated environments – progress with the CHARISMATIC project. In VAST Conference Proceedings, 2000.

[8] Birch P., Jennings V., Day A., Arnold D. Procedural Modelling of Vernacular Housing for Virtual Heritage Environments. EGUK, 2001.

[9] Birch P., Browne S., Jennings V., Day A., Arnold D. Rapid procedural-modelling of architectural structures. Conference on Virtual reality, archeology, and cultural heritage, Glyfada, Greece, 2001.

[10] Legakis J., Dorsey J., Gortler S. Feature-based cellular texturing for architectural models. Conference on Computer graphics and interactive techniques, 2001.

[11] Havemann S., Fellner D. A versatile 3D model representation for cultural reconstruction. Conference on Virtual reality, archeology, and cultural heritage, Glyfada, Greece, 2001.

[12] Mäntylä M. An introduction to solid modeling. Computer Science Press, 1988

[13] Corney J., Lim T. 3D Modeling with ACIS. Saxe-Coburg Publications, 2001.

[14] Baumgart B. Geometric Modeling for Computer Vision. Stanford Artificial Intelligence Laboratory, Memo no. AIM-249, Stanford University, October 1974. (http://www.baumgart.org/winged-edge/winged-edge.html)

[15] Brausewetter A. Das Bauformenbuch – Erster Teil. Verlag von E. A. Seemann, Leipzig, 1895.

[16] Felkel P., Obdrzalek S. Straight Skeleton Implementation. 14th Spring Conference on Computer Graphics, April 23 -25, Budmerice, Slovakia, 1998.

[17] Boost. http://www.boost.org/libs/python/doc/index.html (seen may 2005)

[18] Python. http://www.python.org (seen may 2005)

[19] wxPython. http://www.wxpython.org (seen may 2005)

[20] wxWidgets. http://www.wxwidgets.org (seen may 2005)

[21] 3Delight. http://www.3delight.com (seen march 2005)

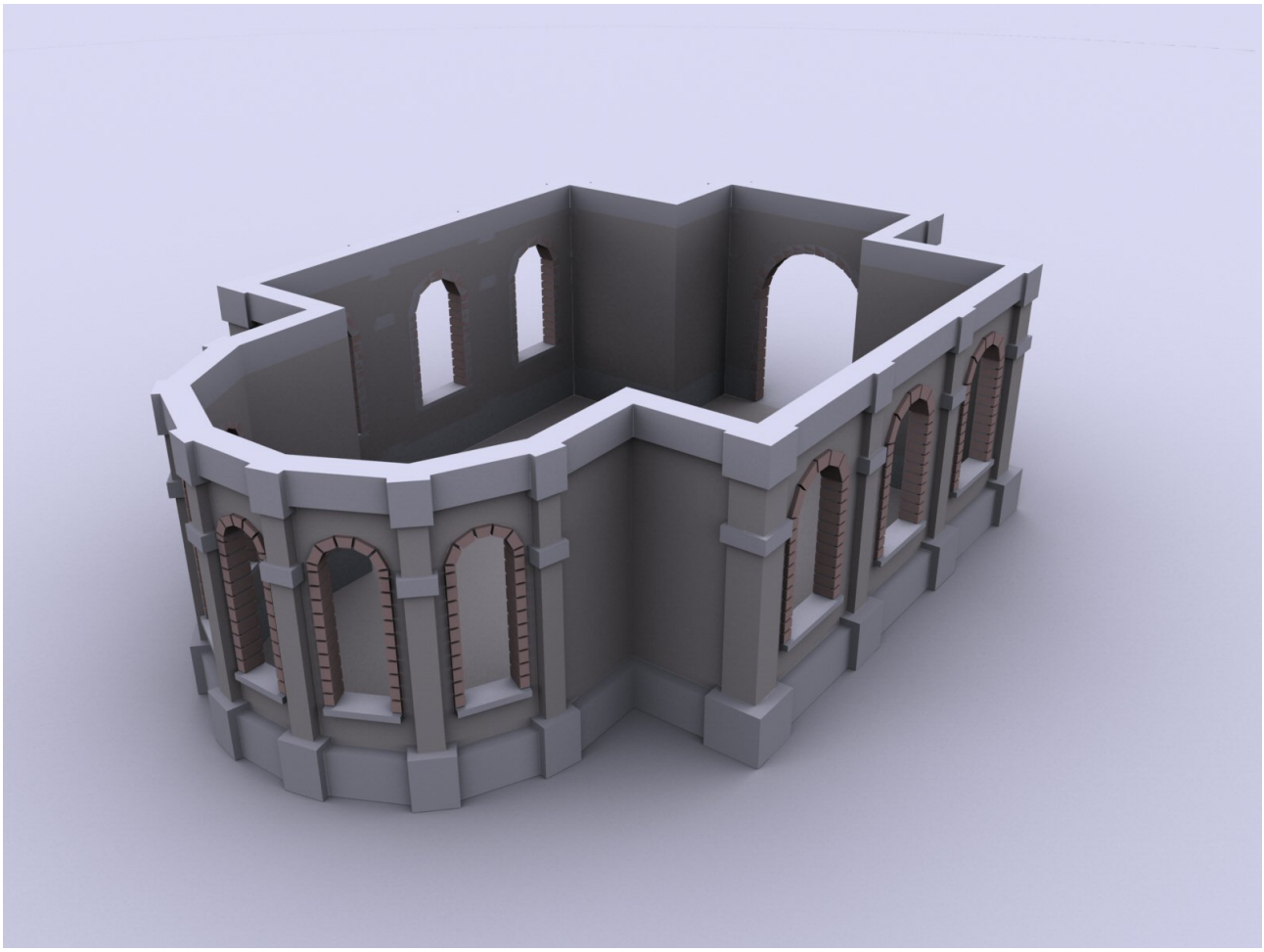

Figure 14 : Façade with detailed window frames.

