SEMANTIC REPRESENTATION OF COMPLEX BUILDING STRUCTURES

Dieter Finkenzeller and Jan Bender
University of Karlsruhe
Am Fasanengarten 5
76128 Karlsruhe
Germany

ABSTRACT

In this paper we present an abstract semantic representation that is suitable for complex buildings. Facades with high-level detail are required in several domains, e.g. visualization of architectural settings and archaeological sites as well as computer animations. In order to support the user's modeling task, besides geometrical data structural information like spatial relations is required. This supplementary information represents the semantics of the model. Therefore the model description must incorporate the geometry and the semantics. Such a description allows a partial automation of the modeling process, e.g. adjacent and nested elements are adjusted automatically. An abstract model representation with integrated semantics is presented in this paper and it is shown that it facilitates the modeling task significantly.

KEYWORDS

Semantic procedural modeling, virtual reality, architecture.

1. INTRODUCTION

The conceptual design of complex structures is a major topic in computer science. Especially in the field of entertainment—large virtual environments, computer animation and computer games—high-performance hardware enables complex and high quality scenes. Therefore it is important to support the modeling process through powerful software.

Modeling detailed scenes, e. g. cities and buildings, becomes a more important topic and requires a great deal of time. To tackle this problem, we focus on transforming the modeling task to an abstract, high-level process. Hence we use typed graphs to obtain a semantic representation of the entire building—including its coarse outline and style. For the building outline the graph consists of nodes representing building parts, like side wings, oriels and wall projections. Edges in the graph indicate the types of connections between these parts. The style nodes and edges present the layout of walls, windows and cornices and so on. All nodes and edges contain a minimum of geometric information such as width and height. This solves several problems. With this approach we achieve a clear separation of the modeling task in which both, the designer and the computer have their distinguished duties. The semantic representation of the building and its style can be exported at any time to a geometry engine that parses the information and automatically creates the mesh. Due to the representation in the typed graph spatial dependencies can be easily derived. They are used for the automatic adaptation of adjacent architectural structures, e.g. for windows that fit into their corresponding walls. For the designer the modeling is performed on a semantic level. As a consequence he produces more complex facades in less time. He can try different styles on the same building outline and produces higher quality facades and therefore save human resources and money.

2. RELATED WORK

The procedural modeling of buildings and entire cities is an evolving topic in computer graphics with many contributions. In this section we introduce some of the most relevant ones concerning their semantic representation.

Parish et al. (2001) describe cities and buildings using grammars and L-systems. The facades are modeled by using procedurally generated textures. Detailed structures such as cornices and arbitrary window frames are not supported. The buildings are generated by a set of production rules. There is later on no data structure representing the building including different architectonical parts in a semantic manner.

Wonka et al. (2003) introduce split grammars for creating buildings. A facade is represented as a non-terminal shape. Such a shape can be split further into smaller non-terminal shapes. In the last step of splitting this leads to terminal shapes like windows and wall elements.

The work from Müller et al. (2006) incorporates the two works mentioned above. Coarse building outlines are created through the combination of boxes and cylinders, which are called shapes. These shapes may overlap, and therefore occlusion tests have to be performed to determine visible faces. The individual building parts do not possess any information regarding their purpose in the building. There is no clear semantic representation of an entire building but the application of the production rules.

Greuter et al. (2003) present an engine that creates large cities at runtime. The basic structure for the city is a regular grid, which also acts as the road map. A building is placed at each junction. Via a pseudo random number a convex polygon for the building's first floor is chosen. To obtain different building outlines, the polygon is extruded and combined with a new randomly chosen polygon. This process is repeated several times. Afterwards the different building sections are textured with predefined tiles. A semantic model of the building structure is missing.

Birch and colleagues (2001) describe techniques for the interactive modeling of buildings and architectural structures. Their main idea is to reduce the number of parameters to a manageable size. Details like window frames are generated procedurally but have less variety then the ones of Müller. Some semantics are included in the interactive modeling process. An explicit semantic model of the entire building including the style is absent.

The method in this paper shows how to describe different building parts with their purpose in the entire building—in a single story and a cross adjacent stories. It also includes definitions for different styles including cornices and window frames and so on.

3. TYPED GRAPHS

The geometry and the semantics of a building must be described in a common representation structure. A typed graph is a well-suited representation structure for this application. Such graph consists of two parts. The first part is an abstract type description of a graph (which we refer to as the *concept model*) that contains class definitions for nodes and edges. The class of a node defines its type and its attributes. An edge class also specifies a type and attributes. In addition to that the source and target node class of an edge is defined. This means that an edge can only connect nodes of a predefined class. The attributes of a node class describe the properties of the objects represented by this node type. The properties of a relation between two nodes are described by the attributes of an edge class. The formal description is as follows:

The directed graph consists of a finite set N of nodes, a finite set E of edges, and a finite set A of attributes. Every edge $e \in E$ connects exactly one source node $k \in N$ to one target node $l \in N$. An edge is assigned its source and target with the two functions $\alpha : E \to N$ and $\omega : E \to N$. $\alpha(e)$ represents the source and $\omega(e)$ the target node. Attributes are assigned to source and target nodes with the function $\tau : (N \cup E) \to P(A)$, where $P(A)$ is the power set of A.

The second part is an instance of the previously defined graph type (the *world model*). In such a graph instance each node is an instance of a specific node class. The nodes that can be connected by an instance of an edge type are well defined by the corresponding edge class. In each node and edge instance information can be stored by setting the values of the attributes.

4. BUILDING REPRESENTATION

In this section we describe the method how we applied the typed graphs for entire buildings. This includes the representation for the coarse building outline and the style. For both we supply independent graph structures. In each case we provide a *concept model* and a *world model*. This enables us to separate the style from the building outline. We also support a consistency check so that the following conditions hold:

- Every node and every edge is unique.
- Every edge represents exactly one link between a source and a target node.
- Attributes have to be unique only in the scope of its node or edge respectively.

4.1 Building outline

In this section we present the nodes and edges that define the coarse building outline. We will only present a subset of the nodes and edges due to space limitations. An overview of all nodes and edges is given in Figure 1. We start with empty sets for the nodes, edges, and attributes: $N, E, A = \varnothing$. From one node definition to another we expand the sets with the new nodes and edges respectively.

- **Node definition:** *cornerstone*
  The coarse building outline starts with the *cornerstone*. Starting at this point a story and next level stories will be created. It has the two attributes *height* and *width*.
  $N \cup cornerstone$,
  $A \cup \{height, width\}$,
  $\tau(cornerstone) = \{height, width\}$

- **Node definition:** *floor_plan_module*
  This node defines building structure types like side wings, oriels, and wall projections. It has two attributes *type* and *width*.
  $N \cup floor\_plan\_module$,
  $A \cup \{type\}$,
  $\tau(floor\_plan\_module) = \{type, width\}$

As mentioned above this is only a subset of the nodes. The next definitions describe the relations between the nodes. Here we also present only a subset.

- **Edge definition:** *initial_floor_plan_module*
  With this relation the *cornerstone* receives its initial *floor_plan_module* (*fpm*). It has no attributes.
  $E \cup initial\_floor\_plan\_module$,
  $\alpha(initial\_floor\_plan\_module) = cornerstone$,
  $\omega(initial\_floor\_plan\_module) = floor\_plan\_module$

- **Edge definition:** *next_story*
  This relation sets for the current *cornerstone* the *cornerstone* for the story on the next level. It has no attributes.
  $E \cup next\_story$,
  $\alpha(next\_story) = cornerstone$,
  $\omega(next\_story) = cornerstone$
  This relation defines a classical cycle but it will not show up in the *world model* as we check for illegal cycles during the geometry generation.

- **Edge definition:** *intra_connection*
  This powerful relation connects different building parts (*floor_plan_modules*) on the same level to build a floor plan for the current story. Its attributes describe the parameters of the connection between two *floor_plan_modules*. Two *floor_plan_modules* are connected via their edges. We use *edge_i* for the first and *edge_j* for the second *fpm*. The edges are also parameterized that one edge resides in an interval on the other edge. Then one *fpm* is transformed (scaled, translated, and rotated) so the two edges match.
  $E \cup intra\_connection$,
  $A \cup \{type, edge\_k, edge\_i, edge\_j, a\_edge\_i, b\_edge\_j\}$,
  $\tau(intra\_connection) = \{type, edge\_k, edge\_i, edge\_j, a\_edge\_i, b\_edge\_j\}$
  $\alpha(intra\_connection) = floor\_plan\_module$,
  $\omega(intra\_connection) = floor\_plan\_module$

The entire semantic model with all nodes and edges is given in Figure 1.

4.2 Facade style

According to the definitions for the nodes and edges of the coarse building outline we present the nodes and edges which describe the style. The style includes the subdivision of the walls into sub walls, the partitioning of walls, cornice and window descriptions, texture descriptions, and materials. Again we will only present a small subset of the definitions. Figure 2 shows the entire graph for the style definition. For visual convenience we omitted the attributes.

- **Node definition:** *style*
  This node represents the style for an entire building or coarse structures like *floor_plan_modules*.
  $N \cup style$
  As we see it has no attributes. Its semantics arises when it is connected to the other nodes.

- **Node definition:** *suvdivison*
  This node supports the subdivision of walls in smaller walls, corners, and elements between smaller walls. It has the attributes *elem*, *com_subdiv*, *h_elem*, and *h_corner*. *elem* defines small elements between walls, *com_subdiv* includes on how to subdivide walls into smaller walls, *h_elem* and

h_corner determine if small wall elements and corners are to be subdivided.
$N \cup suvdivison$,
$A \cup \{elem, com\_subdiv, h\_elem, h\_corner\}$,
$\tau(suvdivison) = \{elem, com\_subdiv, h\_elem, h\_corner\}$

- **Node definition:** *wallpartition*
  This node controls the vertical subdivision of walls. Such a partition can contain a cornice or a window or a door and so on. It has the attributes *v_start* and *v_stop*, which determines the horizontal interval for the partition.
  $N \cup wallpartition$,
  $A \cup \{v\_start, v\_stop\}$,
  $\tau(wallpartition) = \{v\_start, v\_stop\}$

- **Node definition:** *windoor*
  This node describes the appearance of windows and doors. Each door or window is defined via a coarse rectangular hole in a wall. Now every edge can be refined with a line strip to achieve arbitrary windows and doors. Each line strip is later assigned to one of four different frame types. At last the inner window or door frame including the window pane is defined. All these refinements are individual nodes. Their correlation is given in Figure 2. The *windoor* node's attributes define its position in a wall.
  $N \cup windoor$,
  $A \cup \{v\_start, v\_stop, h\_start, h\_stop, id\}$,
  $\tau(wallpartition) = \{v\_start, v\_stop, h\_start, h\_stop, id\}$

- **Node definition:** *cornice*
  With this node the data for a cornice is defined. The cornice profile is created via a logo like language and is given in a textual form. We will see in the next nodes that the profile of a cornice can be used for several purposes. *cornice_description* includes the textual profile description based on lines and arcs. The arcs are approximated by line strips. With the attribute *subdivs* the granularity of the line strip is determined.
  $N \cup cornice$,
  $A \cup \{cornice\_description, subdivs\}$,
  $\tau(cornice) = \{cornice\_description, subdivs\}$

- **Node definition:** *windoor_edge_refinement*
  This node enables the refinement for one of the four edges of a door or window. The refinement is a line strip which is given as a cornice profile. The *windoor_edge_refinement* node has two attributes *cornice_description* and *subdivs*.
  $N \cup windoor\_edge\_refinement$,
  $A \cup \{cornice\_description, subdivs\}$,
  $\tau(windoor\_edge\_refinement) = \{cornice\_description, subdivs\}$

- **Node definition:** *windoor_frame_doublecornice*
  This node defines a special frame type for a window or a door. The previous node just refined an edge. With this node it receives one of four different frame types. Currently the types simple, cornice, double cornice, and bricks are supported. The frame's dimensions are specified with the three attributes *depth*, *width*, and *projection*.
  $N \cup windoor\_frame\_doublecornice$,
  $A \cup \{depth, width, projection\}$,
  $\tau(windoor\_frame\_doublecornice) = \{depth, width, projection\}$

Now as we defined the most necessary nodes the most important relations between the nodes will be defined.

- **Edge definition:** *has_windoor*
  This relation assigns an entire definition for a door or window to a style. It has no attributes.
  $E \cup has\_subdivisions$,
  $\alpha(has\_subdivisions) = style$,
  $\omega(has\_subdivisions) = windoor$

- **Edge definition:** *has_refinement_left*
  With this relation the left edge of a *windoor* node is refined with a line strip. There are three additional relations, which refine the bottom, right, and top edge. It has no attributes.
  $E \cup has\_refinement\_left$,

α (*has_refinement_left*) = *windoor,*
ω (*has_refinement_left*) = *windoor_edge_refinement*

- **Edge definition:** *has_frame_doublecornice*
  Refined edges can be assigned to one of four frame types. In this case the edge receives a double cornice. The windows in the bottom and middle stories in Figure 4 have double cornice refinements.
  $E \cup$ *has_frame_doublecornice,*
  α (*has_frame_doublecornice*) = *windoor_edge_refinement,*
  ω (*has_frame_doublecornice*) = *windoor_frame_doublecornice*

Now we defined the nodes and relations for the coarse building outline's *concept model* and for the style's *concept model*. Instances of both *concept models* define a particular building outline or style.

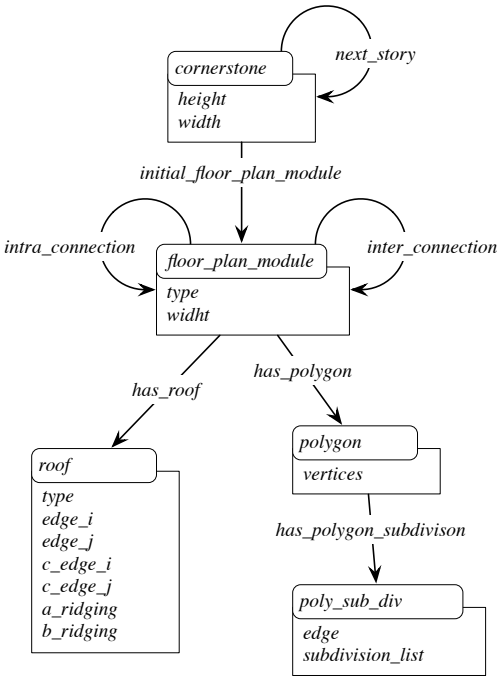

Figure 1: Graph representing the coarse building outline.

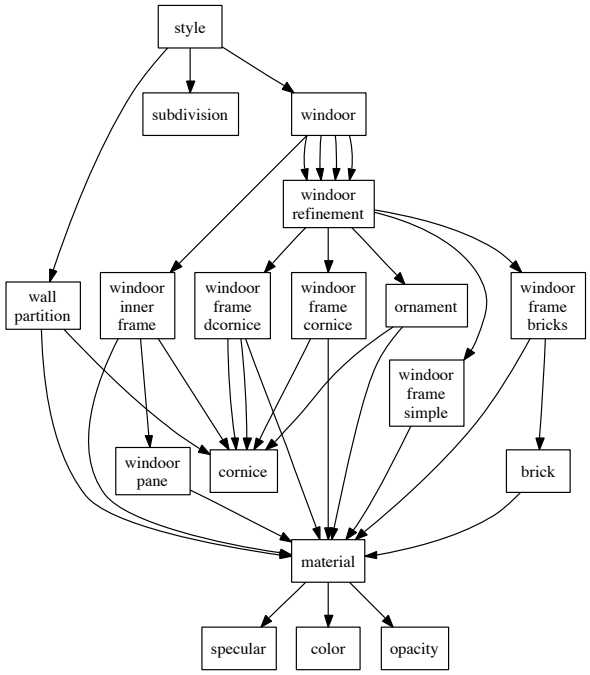

Figure 2: All nodes and relations defining the style.

5. GEOMETRY ENGINE AND RESULTS

For any building outline any style can be applied. After that a new semantic model (*intermediate model*) is generated which combines the building and the style. At this point the user can still modify the building outline or the style for the entire building, which will result in the recreation of the *intermediate model*. But the user has also the possibility to interact on a semantic level with the *intermediate model*. If he is content with the building the geometry engine parses the *intermediate model* and creates exact fitting geometry and textures. The workflow of the modeling process is given in Figure 3. There we have the three different models. For model to model the complexity and the expressiveness grows. But as one can easily see, the memory consumption for the *intermediate model* is very low. Compared to the full model (the output) the semantic representation only needs a fraction of memory. With our prototype we generated the results in Figure 4 and Figure 5.

6. CONCLUSION

We presented a method for the semantic modeling of buildings. Our method is based on typed graphs, which contain the semantic representation of the coarse building outline independent of the style. With this approach the user can easily change the building and its style on a semantic level. Finally the semantic information is given to a geometry engine, which creates the mesh for the building. Due to the graph representation adjacent structures are derived easily and adapted automatically. This enables the designer to create more complex

buildings multiple times faster than with usual modeling tools. The disadvantage of this approach is that it can only handle standard build types—organic or curved structures even inclined walls are not possible yet.

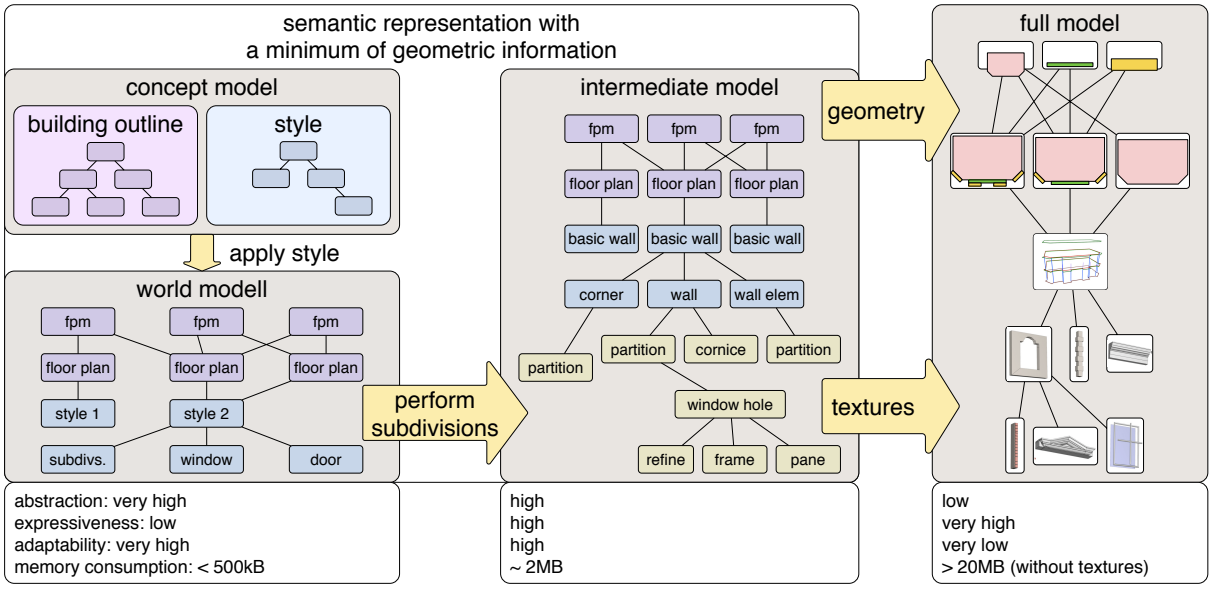

Figure 3: Modeling work flow. From the abstract concept model to the geometry (full model).

REFERENCES

Parish, Y. and Müller, P., 2001. Procedural modeling of cities. *Proceedings of 28th annual conference on Computer graphics and interactive techniques (Siggraph).* Los Angeles, California USA, pp. 301-308.

Wonka, P. and Wimmer, M. and Sillion, F. and Ribarsky, W., 2003. Instant architecture. *Proceedings of 30th annual conference on Computer graphics and interactive techniques (Siggraph).* San Diego, California USA, pp. 669-677.

Müller, P. and Wonka, P. and Haegler, S. and Ulmer, A. and Van Gool, L., 2006. Procedural modeling of building. *Proceedings of 33th annual conference on Computer graphics and interactive techniques (Siggraph).* Boston, Massachusetts USA, pp. 614-623.

Greuter, S. and Parker, J. and Stewart, N. and Leach, G., 2003. Real-time procedural generation of 'pseude inifinite'cities. *Proceedings of 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia.* Melbourne, Australia, pp. 87-94.

Birch, P. and Jennings, V. and Day, A. and Arnold, D, 2001. Procedural modelling of vernacular housing for virtual heritage environments. *Proceedings of 19th Eurographics UK Conference.* University College, London.

Figure 4: Classical mansion with cornices and ornaments.

Figure 5: Row houses with brickwork pattern.