Procedural Arrangement of Furniture for Real-Time Walkthroughs

T. Germer† and M. Schwarz‡
Otto-von-Guericke University of Magdeburg
Department of Simulation and Graphics
Universitätsplatz 2, D-39106 Magdeburg, Germany

Abstract
This paper presents a procedural approach to generate furniture arrangements for large virtual indoor scenes. The interiors of buildings in 3D city scenes are often omitted. Our solution creates rich furniture arrangements for all rooms of complex buildings and even for entire cities. The key idea is to only furnish the rooms in the vicinity of the viewer while the user explores a building in real time. In order to compute the object layout we introduce an agent-based solution and demonstrate the flexibility and effectiveness of the agent approach. Furthermore, we describe advanced features of the system, like procedural furniture geometry, persistent room layouts, and styles for high-level control.

Categories and Subject Descriptors (according to ACM CCS): I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: procedural modeling, interior generation, multi-agent system

1. Introduction
Procedural modeling research has led to a number of impressive systems capable of generating complex city models including detailed buildings. At the same time, rapidly increasing power of commodity hardware resulted in applications such as 3D games with huge virtual environments which comprise several complex cities with thousands of buildings. However, these applications allow the viewer to enter and explore only selected buildings. Typically, only the interiors of interest are modeled and thus most doors to the remaining rooms stay closed.

Reasons for not modeling every single room’s interior include the difficulties to store, manage and render such vast scenes. Modeling all indoor scenes of entire cities manually would be impractical and the viewer will probably enter only a limited number of buildings anyway.

Our work aims to break this limitation and to complement manually modeled interiors. To overcome these restrictions, this paper presents a procedural solution that only generates the furniture needed in order to create the illusion of entire interiors. Initially, the rooms of the buildings are empty. The interiors of the rooms are created automatically at run-time just before they become visible to the viewer. This way, only a few rooms are actually furnished at any instant, allowing for complex interiors. Nevertheless, we retain the ability to create the illusion of completely modeled interiors, even for whole cities.

To meet these requirements, our system has to generate furniture arrangements rapidly while the user explores the scene. The arrangements must be persistent, such that they appear unaltered if the user re-enters a room he or she visited before. The interiors should also be plausible and interesting, i.e., exhibit some variance. In general, the problem of interior modeling can be divided into geometry modeling and object layout. In this paper we concentrate on the latter and focus on the arrangement of the furnishing. The major contributions of this paper are as follows:

† e-mail: germer@isg.cs.uni-magdeburg.de
‡ e-mail: martin.schwarz@bitmanagement.com

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.
Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.
We present a fast and flexible system for procedural furniture arrangements. To our knowledge, it is the first approach to create room interiors suitable for complex interactive environments including whole buildings and even cities.

We exploit the multi-agent system paradigm to create object layouts automatically. This offers an effective solution based on simple, local representations, which has the flexibility needed to generate complex indoor scenes.

With real-time walkthroughs of large interiors, we demonstrate a useful application of procedural modeling on demand.

After summarizing related work in section 2, we introduce our agent-based furniture model in section 3. We then use these agents to develop our interior generation system in section 4.

2. Related Work

Procedural modeling has a long history in computer graphics [EMP*02]. In the past, the focus has been mainly on natural phenomena, such as landscapes [Mus93], clouds [Per85], materials [RTB*92], or plants [PL96]. These methods often are based on fractals or grammars like L-systems.

Recent research also focuses on architecture ranging from whole cities to facades and floor layouts. The first sophisticated system for the automatic generation of virtual cities called “CityEngine” was introduced by Parish and Müller [PM01]. It employs an extended form of L-systems to create convincing cityscapes including street layouts and simple buildings of various types. Greuter et al. present a procedural city-modeling system with the emphasis on real time generation, allowing the user to explore “pseudo infinite” cities [GPSL03]. They build on the idea of only generating parts of the city that are necessary to fill the view frustum. Interactive modeling of street layouts has been presented by Chen et al. [CEW*08] as well as Kelly and McCabe [KM06, KM07]. Aliaga et al. present a flexible editor for high-level modifications of existing urban layouts that also employs GIS-data [ABVA08]. Wonka et al. focused on complex building geometries and used split grammars to create detailed building shapes [WWSR03]. More recently, Müller et al. integrated the split grammar technique into the CityEngine and extended it to support even more complex and consistent buildings, resulting in a very powerful system for various cityscapes [MWH*06]. An approach to generate detailed facades based on photographs of real buildings was presented by Müller et al. [MZWG07].

Although modeling the detailed outer shape of buildings, none of these systems considers interiors. An approach to model floor plans for residential houses was presented by Martin [Mat06]. Here, modeling starts by creating a graph representing the room topology, which is converted into room geometry by exerting certain pressures between neighboring rooms. This can be regarded as a first step to integrate interiors into procedural city modeling. However, it does not create or arrange furnishing inside the rooms. Another approach to generate floor plans was presented by Hahn et al. [HBW06]. To generate building interiors they split the floor at random positions and connect the resulting regions by portals. They use a lazy generation scheme in conjunction with a persistent change manager to allow interactive walkthroughs of large buildings. While being very useful for generating floor plans, this system again does not consider the furnishing inside the rooms.

The problem of arranging furnishing in predefined rooms can be described as creating object layouts. The Wordseye system infers 3D scenes from natural language roughly describing that scene [CS01]. Seversky and Yin use spatial relationships described in natural language for the automatic arrangement of simple scenes [SY06]. However, both systems are constructed only for single individual scenes and...
are not suitable as a general tool for object layouts because of the known ambiguities in natural language.

A related area of research is “declarative modeling” [Gaï07, BP99, RGc04]. Here, the user has to specify a high-level description of the scene including relationships between the objects. The system then generates object layouts that satisfy these constraints using various methods such as evolutionary algorithms [SRLG03] or meta-heuristics [LRG04]. However, these systems still rely on user interaction and manual descriptions of the scene. The CAPS system supports the automatic creation of complex object layouts [XSF02]. Using a combination of placement constraints, pseudo-physics and a semantic database, it creates complex and plausible interiors. However, being intended as a user-oriented modeling tool with layout times of several minutes, it is not suitable for the automatic generation of interiors in real time.

The main inspiration for our object layout system is the work of Akazawa et al. proposing a framework for the automatic generation of interiors [AON05]. Here, bounding boxes as simplified representations are used to arrange objects next to each other through contact constraints. Objects are connected in hierarchies representing functional dependencies. In conjunction with collision tests and simple semantic descriptions, the system provides a simple solution for object arrangements. However, it only supports very simple interiors for single rooms and does not consider inter-active walkthroughs of furnished buildings. Therefore, concepts like room style or persistence are missing. In addition, the system still requires several seconds for the generation of moderate scenes. This may result from the need to solve several contact constraints for each object simultaneously at run time. The objects are also placed in a fixed order according to their type. The order of object types is determined by the hierarchy. This results in very rigid layouts, because all objects of the same type are arranged at the same time. Other objects can only be placed afterwards, which restricts the possible arrangements. In contrast, we create flexible objects in the form of agents that employ local rules and simple heuristics instead of contact constraints. They arrange themselves dynamically without a fixed order. This makes our system fast and flexible enough to generate a variety of rooms to be used for interactive walkthroughs of virtual buildings.

Another approach to automatically arrange furniture is to recursively divide the space of a room into functional subspaces (e.g., “working area”) until single objects can be placed [Kjø00]. A set of predefined templates is used to determine the splitting for every level of the hierarchy. If conflicts arise (overlapping subspaces), a set of modiﬁcations can be applied to the template to accomplish a valid layout. However, the templates still must be modeled manually, which restricts the possible furniture layout. In addition, the decomposition is restricted to rectangular footprints in the floor plan, so the system can only handle rectangular rooms and templates.

In contrast to this top-down method of recursive decomposition, our object layout solution follows a bottom-up strategy and exploits the paradigm of multi agent systems. The concept of “agents” originates from the ﬁeld of Artiﬁcial Intelligence. An agent can be considered as an autonomous entity existing in an artiﬁcial world exhibiting a certain behavior [Wei99]. Unfortunately, there is no single deﬁnition of agents and researchers often disagree about what exactly deﬁnes an agent [FG97]. We will not propose yet another deﬁnition, but rather conceive the agent paradigm as the local perspective under which the solution for a problem is developed. Thus, our understanding of multi agent systems in this work is that:

- the problem is distributed to multiple autonomous entities, situated and interacting in a common environment,
- each entity is self-contained and exhibits a certain behavior,
- a solution is built by local actions of the entities,
- data and computation are decentralized.

There are several applications of multi agent systems to computer graphics, ranging from animation [Rey87] to NPR [SGS05]. Agent systems were even applied to the simulation of growing cities for the generation of street layouts and land usage [LWW07]. Here, the decision to use agent systems was motivated by the simple, local description of building behavior and the ability of agents to deal with special cases. Lechner et al. observe that encoding such behavior in L-systems would have required a big number of rules and parameters.

3. Agent-based Interiors

The agent paradigm as introduced in the previous section provides a means for us to develop a ﬂexible, robust and extensible solution for procedurally modeling interiors. We regard single pieces of furniture as autonomous agents that are able to move around inside a predefined room in order to arrange themselves properly. Each agent is augmented with a semantic description, which encodes the local design aspects of the interiors. The agent behavior and semantic descriptions can be easily combined with other 3D models. This allows us to quickly create similar agents.

3.1. Box-based Model

A good approximation for many furniture objects are oriented bounding boxes. We arrange objects next to each other based on the faces of their bounding boxes. Thus, although agents represent actual 3D objects, their functional shape can be thought of as the corresponding bounding box. Object relations can then be deﬁned as “above”, “below”, etc. (see ﬁgure 2, left). We found this to be sufﬁcient for most layout tasks.
For each furniture object, there is usually a single other object that it depends on, such as the floor, a wall, or a desk. For example, if the desk is moved, the depending objects like a chair in front of it, or a lamp on top of it, should be moved accordingly. To capture such asymmetric dependencies we adopt the notion of parent-child relationships similarly to Akazawa et al. [AON05], with the root represented by the room. Refer to figure 2 (right) for an example.

3.2. Agent Behavior

The main goal for each agent is to find a suitable parent object and to place and orient itself properly. In order to reach this goal, the semantic description of each agent contains:

- its own size, i.e., its bounding box,
- the class of possible parent objects and for each class the sides where the agent can place itself,
- the occupation of its own sides, describing for each face of the bounding box how much space and how many places are left for children.

These agent semantics allow us to encode functional relationships. Such dependencies are necessary to create plausible object layouts. For example, chairs may either stand next to a table or may be solitary and, thus, have several possible parents. Functional relationships are highly subjective and require user input. We allow the designer to encode such relationships with custom semantic descriptions to define sets of objects to be grouped together.

Another goal for each agent is to respect physical constraints in order to create plausible interiors. An agent must not collide with other objects, except for its parent. We use a simple collision detection scheme based on the agent’s bounding boxes and the separating axis theorem [GLM96] to test for this condition. Usually, an agent should not be floating in the air. Because we do not compute real physics, each agent is responsible for itself to compute an appropriate position by aligning itself properly with its parent.

We describe the behavior of our agents using three states (see figure 3) corresponding to the following actions:

**Search**: Initially, each agent is in search state. An agent searches for possible parents by examining other agents in its environment (the surrounding room). If it finds a possible parent, the agent examines the semantics of the parent, looking for a suitable side with enough space and places left. If it succeeds, the agent changes its state to “arrange”. Otherwise, it tries to find another parent. If no parent could be found, the room is considered to be full and the agent is deleted.

**Arrange**: In arrange-state, an agent attempts to place itself next to the chosen side of the parent candidate. First, it aligns itself with the parent such that the side of the parent touches its opposite side (e.g., top to bottom). If the top or bottom side of the parent was chosen, the agent uses a random 2D position sideways. Otherwise, the agent aligns its bottom side with the bottom of the parent, and uses a random 1D position along the touching side. This imitates the effect of gravity to ensure physical plausibility. Afterwards, the agent adjusts its orientation. In most cases, it simply aligns parallel to the parent. Depending on its semantics, it can also diverge from that. For example, a random offset can be applied or the agent can be oriented towards the center of the parent. Next, the agent tests for collisions with other agents. If there are no collisions, the arrangement was successful and the agent changes its state to “rest”. Otherwise, it tries to arrange itself at another random position at the same parent. If this does not succeed after a few trials, it returns to the state “search” to find another parent. Note that the collision test also takes care of fragmented free space at the sides of the parent and prevents agents from overlapping with siblings.

**Rest**: Once the placement of an agent is finished, other agents can become its children. If its parent moves, the agent moves accordingly. However, if this results in collisions, the parent is lost and the agent returns to the state “search”. Agents in rest-state usually do not move anymore. We experimented with agents pushing resting ones aside (e.g. tables in a restaurant). However, this increases computational costs and we found the layout of most scenes to work well without this feature.

In performing these actions, each agent modifies its own position and its environment (the room). It influences the
behavior of other agents by occupying certain space in the room and at its parent. In practice, we prioritize agents according to their expected depth in the hierarchy, so that the arrangement of tables is attempted before chairs search for parents. This speeds up the search process.

3.3. Regular Interval Layouts

In some situations random positions are not appropriate and a regular object layout is required. We support this by extending the semantic description of agents to include the specification of grid layouts. Instead of arranging randomly at the side of their parent, agents with grid layout arrange in grid patterns with intervals specified in their semantics. They do this by looking for neighboring agents that are of the same type and are already placed at the considered side of the parent. Figure 4 shows an example of a restaurant that was procedurally arranged. The lamps on the ceiling were arranged with the grid layout feature.

3.4. Geometry

Each agent carries the geometry of a single 3D object, typically modeled manually. In addition, procedural 3D models offer a convenient alternative for our system. Because we only generate a few rooms at any instant, our approach supports relatively complex interiors. Procedural models provide a means to generate such complexity. We demonstrate this idea with simple procedural models of bookshelves and couches, consisting of differently sized primitives (see figure 5). We instantiate these models with different parameterizations, increasing both the complexity and the variation of our interiors.

4. Interior Generation System

The agents described in the previous section need a common environment to perform their actions. In our system the environment for each agent is defined as its surrounding room, because furniture objects mostly belong to a single room.

We assume that the room geometry is given and has an arbitrary polygonal floor plan. We also assume that a room knows about its characteristics and how it should be furnished. Therefore, rooms are responsible for creating agents. They manage existing agents in lists according to their type, allowing searching agents to quickly find parents of particular classes. The different parts of the rooms (floor, walls, windows, door) are the root parents for the agents when arranging themselves. Agents recognize the room parts by generic names listed in their semantic description. The room designer is responsible for naming them appropriately (see figure 6). Given the agent behavior described in section 3.2, our object layout algorithm is straightforward:

1. Initialize all rooms.
2. Create agents corresponding to the room characteristics.
3. Set all agents to the state “search”.
4. Perform actions for each agent according to its current state until all agents are in the state “rest” or deleted.

This way, the furniture objects are incrementally arranged in the room until there are no further objects or the room is full. Rooms with such furnishing capabilities can be easily combined into floors, which in tum can be combined into whole buildings.
To create the illusion of a fully furnished building for real time walkthroughs, we employ a simple room generation scheme. The interior of a room is created once the viewer comes closer than a certain threshold and can possibly take a look inside that room. On the other hand, rooms far away from the viewer are discarded to free resources. This basic scheme works well but does not guarantee that all rooms are readily furnished when they are visible to the viewer. More sophisticated schemes are possible, depending on the application and shape of the rooms. For example, it would be beneficial to generate rooms in a background thread and to create a cache of furnished rooms based on probable movements of the viewer. An implementation of portal rendering [LG95] could determine the rooms to be generated based on their actual visibility. This would implement the idea of generating the rooms on demand more precisely (as shown in the context of city generation by Greuter et al. [GPSL03]). Generating the interiors just before opening the door to a room would be another option.

4.1. Persistence

Suppose the viewer departs from a room and the room layout is discarded. If the viewer re-enters the room later, the interior should be the same to create the illusion of persistent interiors. In order to guarantee that all objects are arranged exactly the same every time the room is filled, we employ a deterministic random number generator for all variations of interiors. Unique seeds are computed independently for each room based on its position and number.

In addition, we allow users to move furniture objects at run time. The room saves such displacements as “offsets” for each of its moved agents. When the room is re-created, the offsets are added to the corresponding agents as soon as the procedural arrangement is finished. Because the object layout is deterministic, this fully restores the interior.

At present, our system does not provide a means to interactively add or remove objects at run time. However, it would be straightforward to extend our persistence concept to these operations in order to store the user generated effect as an “offset” at the room level.

4.2. Styles

To give the designer a form of high-level control of the scene, our system supports different styles. A style determines the appearance and layout of the interior. It describes one or more attributes of the room. Examples are:

Number and type of furniture objects describe the purpose of the room, e.g., office, kitchen, lounge or hotel.

Geometry for 3D objects describes the style of furniture, e.g., old, modern, plain or abundant. Different model sets can be assembled to depict certain cultures or epochs.

Position and orientation can also be altered by styles. For example, untidiness can be modeled with random offsets for the agent rotation and distance from its parent. This can be used for abandoned buildings.

Styles allow the designer to quickly create floors and buildings with great diversity. Figure 7 presents several different styles.

At the current version of our system, a style represents a fixed choice of objects that designers can use to furnish a room. It does not include room semantics describing room function or purpose explicitly. Therefore, it is possible to define styles that are contrary to common sense. In addition, the number of furniture objects to be placed in a room should typically depend on its size, which is not possible with a fixed choice of objects. To solve these problems, we plan to incorporate more flexible high-level styles that include semantic descriptions and can be parameterized to reflect various attributes of the rooms (see section 6.2).

4.3. User Input

This section gives a short summary of the input required by the room designer. Apart from the (possibly procedurally generated) building including floor plans, our system needs:

- for each type of room the tags for the room faces,
- for each room the style (which includes the number and types of objects to be created),
- the semantic description for each type of object. This includes (1) the possible parent classes and the corresponding sides, (2) the free places and space for each side, (3) an option for grid layouts and the according intervals, (4) an option on how to orientate towards the parent.

Note that we do not specify any hierarchy trees explicitly, but rather give each object a list of possible parents and let the system work out the hierarchy as part of the layout process.

In conjunction with a procedural floor generation system, the room-specific input (tags and styles) could be created procedurally from high-level floor and building styles. In fact, our current system implements a rudimentary procedural floor generation, where the designer only has to parameterize each prototypical room once with tags and style. This template is then replicated to create the floors.

The object-specific input (semantic description) only is required once for each object, which then can be used for various rooms. In addition, objects with similar behavior can share the same semantic description. Therefore, designers only must provide new semantic descriptions when they wish to use new kinds of objects.

In practice, our system needs much input when starting from scratch. Once the database of semantic descriptions and room styles is larger, new floors and buildings can be created rapidly.
5. Results

We implemented our system with a VRML/Javascript-based prototype. Although it is not optimized for speed, it supports walkthroughs of buildings containing moderately furnished rooms with interactive frame rates. The computation cost needed to generate a room mainly depends on its interior complexity, i.e., the number of objects to be arranged. An average room of 20 to 30 objects can be generated in less than a second on a 3 GHz PC. Very complex rooms such as the restaurant in figure 4 with nearly 300 objects can take up to 4 seconds to finish arrangement. This includes loading times of model files. Because the interior generation is performed in a background process, only few lags are noticeable.

Figure 1 shows a hotel floor with procedural interiors. Note that only rooms in the vicinity of the viewer (in the center) are furnished. Among other examples, figure 7 shows an interior view of the hotel scene. Figure 8 shows another example of an office with a concave floor plan. Figure 9 shows the great complexity that can be achieved by combining procedural object layouts with procedural geometry. The accompanying video shows our system in action.

6. Discussion

We have introduced a procedural solution to arrange furniture for virtual environments. It is the first system that facilitates the modeling of interiors for whole buildings and even cities. We introduced the application of agent systems for laying out objects creating plausible furniture arrangements. In our system, global criteria for the room layout are incorporated in three ways:
Parent-child hierarchies provide asymmetric dependencies, e.g., a lamp on the desk, a chair next to a table. Styles provide high level control and general layout-guidelines to be used by a set of agents. Localized layout criteria encode symmetric dependencies like regular interval layouts, e.g., lamps on the ceiling.

By employing user defined semantics, agents are self-contained and can be designed independently from agents which are not relevant. This makes our system flexible and easily extensible. New agent types can be created by providing a semantic description along with furniture geometry.

To validate the plausibility of the furniture arrangements, we performed a subjective evaluation with 30 different rooms of the hotel scene (see figure 1). We examined every room and determined inconsistencies in the layout. Nine rooms (30%) had very realistic layouts without failures. Five rooms (17%) were not convincing and had deficiencies that would not appear in real hotel rooms. The remaining rooms (53%) were still realistic and had acceptable layouts with minor inconsistencies. The failure points (also shown in figure 10) were:

- The TV has an unusual position, does not point towards the couch or the bed, or stands in front of the window. Note that the latter problem could be solved by giving the wall with the window an appropriate label. (14 rooms)
- Some objects are blocked or are not accessible. (11 rooms)
- The furniture distribution is uneven with one half of the room very crowded. (4 rooms)
- The bathtub stands directly in front of the toilet bowl. (4 rooms)

Note that this is just a rough subjective evaluation of the object layout. A formalized user study that compares procedurally furnished rooms with human-modeled interiors is necessary to measure the layout quality more precisely.

6.1. Limitations

Our interior generation system allows us to interactively explore complex buildings containing a large number of furnished rooms in real time. Our solution gains efficiency through the mainly local and context-free layout of objects. A drawback of this approach is that it does not support complex functional dependencies. For example, there is no way to tell a TV to be placed at the wall opposite to the couch. The system also does not prevent objects from blocking each other. In practice, we found that this issue has little impact and that the interiors are mostly plausible.

A consequence of the generation-on-demand is that our system has a certain maximum walkthrough speed. If the viewer rushes through a building too fast, the system will not be able to generate as many rooms as needed. Thus, our approach is suitable for applications with moderate travel speeds. Another limitation results from the box-based agent model. When the furniture geometry is not sufficiently approximated by a bounding box, the arrangement of neighboring objects produces artifacts like rectangular free space around a round table as shown in figure 10.

As described in section 4.2, the room styles of our system determine the furnishing at a low level. The designer must take care to not produce inconsistent styles that break common conventions or natural constraints (e.g., an office containing a double bed), because the styles do not include high-level semantics like room usage or ergonomics. High-level styles and style hierarchies could encode and abstract such aspects, which could then be used to compose consistent room styles.
From an optimization point of view, our system implements a greedy approach. On the one hand, this results in rapid room layout which is necessary for our primary goal of interactive walkthroughs. On the other hand, the approach has problems finding the optimal layout for very dense scenes. Once an object is arranged, it does not move anymore to make room for other objects. To alleviate this problem, we experimented with agents pushing each other while arranging themselves, producing dense layouts. This surpasses the pure greedy approach, but introduces computational complexity, which contradicts our goal of interactive walkthroughs. However, our system still supports a wide range of scenes as shown in the examples. Moreover, densely packed rooms could be facilitated by treating neighboring objects as parents (e.g., a cabinet standing next to another cabinet).

### 6.2. Future Work

Our approach offers a lot of directions for future work:

**High-level control:** Most importantly, we need a comfortable high-level tool for designers to quickly construct new interiors and model new agents. Our style concept can be used as the basis for a sophisticated style framework that manages whole style libraries and supports the easy creation and manipulation of different styles. Modular, parameterized styles describing various attributes of interiors could be combined into high-level, more abstract styles that also describe high-level attributes like room purpose. Such powerful styles would enhance the flexibility, variation and plausibility of the interiors.

**More flexible layout:** The integration of functional dependencies as well as the generalization of the box-based model to different (e.g., round) objects would enhance plausibility.

**Sophisticated room generation schemes:** Implementing more sophisticated room generation schemes (e.g., based on portal rendering) would allow for more complex rooms and reduce possible artifacts.

**Procedural geometry:** The incorporation of sophisticated procedural geometry would enhance the richness of our scenes, e.g., to add subtle details like pipes, plinth panels or worn-off edges. We are currently working on a system suitable for generating complex furnishings from predefined 3D models.

**Integration:** A very interesting development would be the integration of our approach with procedural floor [HBW06] and city modeling systems [MWH06]. This way, complex cities with furnished buildings could be created. In combination with floor generation systems, the objects to be placed in each room could be chosen randomly in order to increase variation. In conjunction with high-level styles, even the room styles could be generated procedurally to incorporate design aspects from the floor and building level.

### 6.3. Conclusion

In conclusion, we believe that our system is of great use for many entertainment and education applications dealing with buildings and cities. It could be used to complement traditional content with generic indoor scenes in order to enhance the illusion of virtual buildings. Future games could unlock closed doors to give players more freedom and to create the appeal of inhabited urban scenes. Another application that would profit from the enrichment of interiors is architectural visualization, where our system could replace heavily mirrored or blind windows with detailed interiors.

### References


