AN INTERACTIVE BUILDING MODELING SYSTEM BASED ON THE LEGO CONCEPT

Sheng-Yi Chen, Cong-Kai Lin, and Wen-Kai Tai

Department of Computer Science and Information Engineering,
National Dong Hwa University
Taiwan, R.O.C
E-mail: wktai@mail.ndhu.edu.tw

ABSTRACT

In this paper, we proposed an interactive GUI (Graphical User Interface) system to model buildings with an editable script. Our system also provides probabilistic finite-state machine (PFSM) to define the relationships of sub-models with transformation matrices and transition probabilities for constructing new novel building models automatically. User can not only get various building models by PFSM but also adjust the probabilities of sub-models from PFSM to get desired building models. As shown in the results, the various and vivid building models can be constructed easily and quickly for non-expert users. Besides, user can also edit the script file which is provided by our system to modify the properties directly.

Keywords: modeling, Lego concept, interactive modeling, probabilistic FSM

1. INTRODUCTION

Nowadays, there are many methods to model buildings but there are some problems when using these methods such as domain knowledge, complex operations, in need of photos, non-controllable results, and artistry, etc. In order to provide a more convenient user interface, we proposed an interactive GUI system and a novel procedural method based on probabilistic finite-state machine to model various buildings. Besides, user can also edit script files obtained from our system to modify the result of buildings directly.

Our proposed system is inspired from Lego concept. The structure of many buildings is almost regular which we can subdivide the building model into some sub-model as Lego brick. Similarly, we also can subdivide the sub-model continuously until the basic smallest component, terminal. Therefore, we can use these terminals to construct new novel buildings. In our proposed method, we subdivide a building model into five levels including sub-model, shape, face, column, and terminal.

We provide a friendly interactive GUI system. User can edit a script file obtained from our system to modify the properties including relationships of sub-models with transformation matrices and transition probabilities directly. Furthermore, we proposed a probabilistic finite-state machine to construct new novel buildings automatically. Our system will construct various building models with different styles by given some different terminals. User can also construct the desired buildings by adjusting the probabilities of sub-models of PFSM.

The rest of this paper is organized as follows. In Section 2, related work is described. The framework of our proposed method is presented in Section 3. Experimental results are shown in Section 4. Conclusions and future work are described in Section 5.

2. RELATED WORK

Nowadays, there are many researches focused on the modeling and rendering of 3D models and architectural models by using procedural modeling, photo assisted modeling, example-based modeling, and sketch-based modeling. The procedural modeling techniques are powerful in generating various models such as buildings, plants, streets, etc. The common idea of these procedural modeling techniques is to subdivide the buildings into terminals (basic components of buildings) and combine the redundant terminals to generate a new model. There are many different kinds of combination methods such as attributed grammars [5], Lindenmayer system (L-system) [2], shape grammars [7] [8], Semi-Thue processes [14], Chomsky grammars [15], and graph grammars [10]. Moreover, some researchers proposed methods to generate buildings and cities automatically such as Prusinkiewicz *et al.* [20], Parish *et al.* [22], Müller *et al.* [17] and Wonka *et al.* [19].

L-system [2] is controllable and flexible in generating branching objects like stem, streets, plants, etc. Although it can be used for generating buildings with symmetric and branch-like structure, buildings in general have stricter spatial constraints and are not designed by growth process. Therefore, L-system cannot be adapted easily to model buildings. Shape grammars [7] method is suitable to design and model architecture. It uses shape as the basis and defines rules for the transformation and specification of 2D and 3D shapes.

Müller *et al.* [17] proposed a method to generate massive models with unprecedented level of detail using shape grammar. This method provides user to control parameters and grammar, and employs a pre-defined grammar to generate buildings and architectures from a database of given rules and attributes. This method with controllable grammars that correspond to architectural principles is

useful for a wide range of architectural styles with consistent outputs guaranteed and is the first focuses on the aspect of volumetric mass modeling of buildings including the design of roofs. Wonka *et al.* [19] proposed an automatic grammar derivation approach for architecture modeling based on an organized database of grammar rules. Legakis *et al.* [13] proposed the cellular patterns to construct details for architectural models. It is shown that these grammars are useful in the analysis and construction of many architectural styles by extracting data from actual real-world cities or buildings. However, it is not suitable for automatic or semi-automatic modeling because the derivation of grammars needs user interventions. Besides, Müller *et al.* [16] proposed a method to construct facade details of buildings based on the idea of split rule.

Aliaga *et al.* [3] proposed a method to subdivide buildings and construct a grammar from photograph for quick sketching of new architectural structures in the style of the original. Build-by-Number [4] is a method for quickly designing, visualizing realistic architectural structures, and capturing the real-world image data. It can retain the features of models and provide users to design the new structures which are rendered as same as the original images. Example-based modeling method [18] synthesizes many different large and complex buildings which are difficult to generate manually or produce by existing procedural methods. However, it has a considerable limitation that the example models need to be decomposed into pieces of model manually. Additionally, the example models should be constructed carefully to prevent from synthesizing models the same with examples because of the tight constraints.

Shlyakhter *et al.* [11] proposed a sketch-based modeling method to generate models of trees by fitting a coarse branching from a set of photographs. Zeleznik *et al.* [21], and Shesh and Chen [1] combine synthetic models with a pen-based interface to construct models including simple architecture. The method of Oh *et al.* [12] and Google Sketchup [9] are developed as utilities for sketching buildings. However, there are several problems need to be addressed such as geometric interpretation, concise notation, control of the derivation, and the design of actual models. We proposed an interactive GUI system based on the procedural modeling to provide an easy to use modeling utility to construct building models as the concept of playing Lego toys. We consider all of problems by providing an interactive GUI, an editable script and a probabilistic finite-state machine.

3. PROPOSED METHOD

The overview of our proposed interactive modeling system is illustrated in Figure 1. User can manipulate the terminals with properties (including transformation, probability, and relationship) and rules by our interactive GUI system to construct a new model. Our system records all the properties and rules of model in the script. The results of the new model can be displayed immediately for each user interaction. We also provide a probabilistic finite-state machine to allow user to construct new building models easier and quicker.

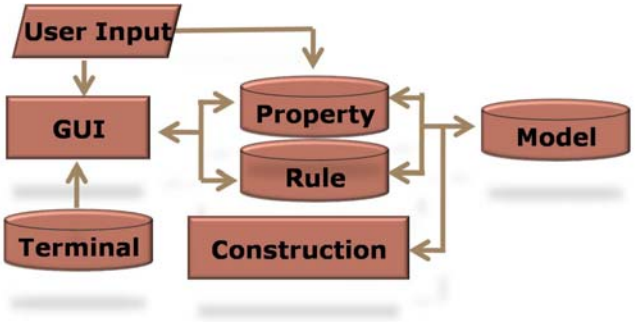


Fig. 1: The overview of our proposed interactive modeling system.

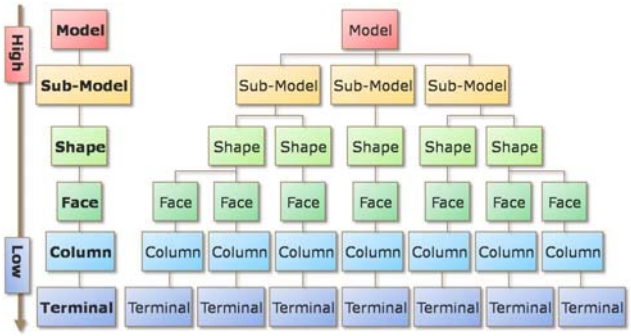


Fig. 2: The hierarchy of model.

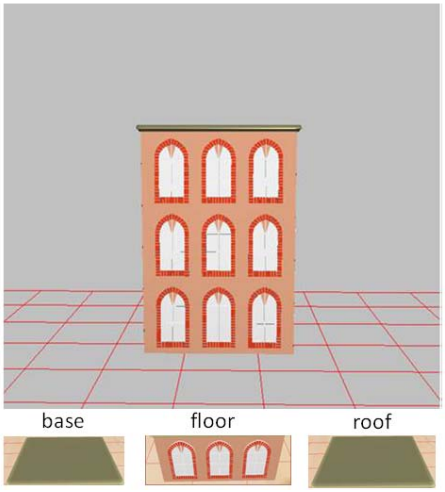


Fig. 3: An example of building and its three groups including roof, floor, and base.

3.1 The Hierarchy of Model

A typical building model usually contains a regular structure that can be subdivided into some terminals. We can use these terminals to construct a new building model similar with Lego concept. We decompose a building model into five levels including sub-model, shape, face, column, and terminal. The hierarchy of model is shown in Figure 2. For example, considering a square building model (Figure 3) with three floors each has three windows, we can regard one floor as a sub-model. Each sub-model consists of three groups including roof, floor, and base with square shape. In this square shape, we have four faces that

each face can be regarded as a wall with or without windows. And, each face has three columns, each has one window surrounded with trim and wall material. Therefore, the window, trim, and wall material (such as brick, stone, and so forth) are the lowest level terminals. We formalize the model with the following general forms and definitions respectively:

Model **M** → $\{S_0, S_1, \cdots, S_{n-1}\}$
Sub-model **S** → $(\{U_0, U_1, \cdots, U_{n-1}\}_{roof})$
$(\{U_0, U_1, \cdots, U_{n-1}\}_{floor})$
$(\{U_0, U_1, \cdots, U_{n-1}\}_{base})$
Shape **U** → $\{F_0, F_1, \cdots, F_{n-1}\}$
Face **F** → $\{C_0, C_1, \cdots, C_{n-1}\}$
Column **C** → $\{T_0, T_1, \cdots, T_{n-1}\}$
Terminal **T**

Definition 3.1: The terminal **T** is the smallest 3D component of building.
User subdivides the model into many terminals such as doors, windows, walls, roofs, corners, stairways, and pillars which form the columns for our system.

Definition 3.2: The column **C** is a set of terminals.
User determines which terminals could be grouped together to form a column. For example, a wall, and a window, can be grouped together to form a wall with a window. The formed column is meaningful in itself.

Definition 3.3: The face **F** is a set of columns.
A face F of a building can be represented by a set of columns with a production rule defined by user in procedural modeling. In Figure 4, this face contains four columns, and the production rule is represented as the string $F = C_1C_2C_3C_4$, where C_j is the j^{th} column.

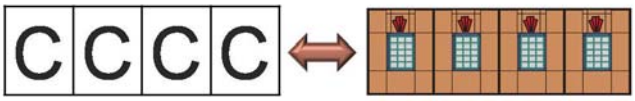


Fig. 4: An example of a face with four columns.

Definition 3.4: The shape **U** could be arranged by a set of faces.
User connects faces side by side to form various shapes as the basis shape of a building. Figure 5 shows that a user connects a set of faces to form different shapes from the top view.

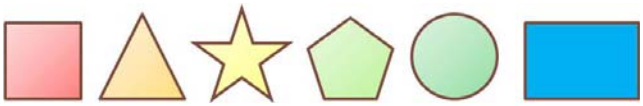


Fig. 5: Examples of different shapes.

Definition 3.5: The sub-model **S** is a non-uniform size block that contains three groups of shapes.
Like Lego brick concept, these bricks can be stacked upon another and can also be regarded as a basic unit and a component. Based on this concept, the sub-model is regarded as a kind of brick which contains three groups of shapes including roof, floor, and base in our definition. Each of roof, floor, or base group is represented as the top, middle, and bottom of brick respectively and has its own features. The sub-model consists of three groups after choosing by user. Figure 6 shows an example of sub-model.

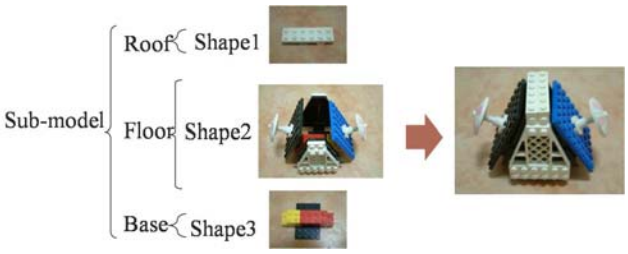
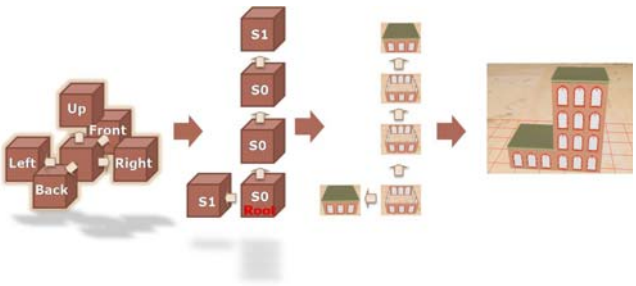


Fig. 6: A real Lego brick example of sub-model.

Definition 3.6: The model **M** constructs from many sub-models.
We can pile lots of bricks to construct a model with plenty of different combinations. Therefore, we can construct a lot of building models from our sub-models. In our definition, each sub-model has five directional positions which can be piled by another sub-model. When piling the sub-model, we need to check the relationships of sub-models to see if the position is legal. After piling several times, we can get a building model according to user's need. Figure 7 shows these five directional positions and an example of a building model.



(a) (b) (c) (d)
Fig. 7: (a) Five directional positions of a sub-model can be piled. (b) A relationship between sub-models. (c) A relationship between real sub-models. (d) An example of building model is constructed with five sub-models.

3.2 Model Construction

With our GUI system user manipulates each level of components by creating, deleting, modifying, and doing transformation. After manipulating, our system records all the properties and rules in accordance with each terminal by using script file. User can use not only the GUI system but also script file to modify the properties and rules of each level of components. In our script file, we define a basic structure to record what the properties and what the rules are. The script file is a simple text file to let user understand and use easily. The following description is the format we defined: **Parameters**, **Sections**, and **Comments**. **Parameter** is the basic element which contains the parameter name and its value, delimited by an equal sign (=) as Name = Value. **Section** is used for grouping some parameters by given a section name, in square brackets ([and]) as [Name]. In sections, it is not allowed for nested

declaration and appears only once on a line in the text. **Comment** is essential for users to understand the script. We use the semicolon (;) to denote that the text behind the semicolon is the comment which is ignored by our system while parsing the script. The comment is as ;comment text. Figure 8 shows an example of our script file, and our GUI system is shown in Figure 9.

```
[Mesh]
Mesh0=.\Terminal\.\\TERMINAL\\BLDNG086-WALL-3.X
Mesh1=.\Terminal\.\\TERMINAL\\BLDNG086-WINDOW-1.X
Mesh2=.\Terminal\.\\TERMINAL\\BLDNG086-ROOF.X
Mesh3=.\Terminal\.\\TERMINAL\\BLDNG081-WALL-1.X
Mesh4=.\Terminal\.\\TERMINAL\\BLDNG081-WINDOW.X
Mesh5=.\Terminal\.\\TERMINAL\\BLDNG081-ROOF-2.X
[TerminalScript]
Terminal0=I0,T0,0,0,S1,1,1,R0,0,0,
TerminalName0=Wall1
Terminal1=I0,T0,0,0,S1,1,1,R0,0,0,
TerminalName1=Window1
Terminal2=I0,T0,0,0,S1,1,1,R0,0,0,
TerminalName2=Roof1
Terminal3=I0,T0,0,0,S1,1,1,R0,0,0,
TerminalName3=Wall2
Terminal4=I0,T0,0,0,S1,1,1,R0,0,0,
TerminalName4=Window2
Terminal5=I0,T0,0,0,S1,1,1,R0,0,0,
TerminalName5=Roof2
```

Fig. 8: An example of script file.

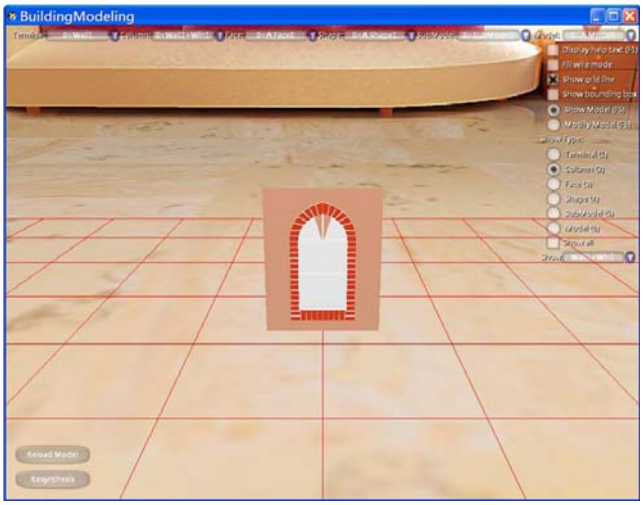


Fig. 9: A screenshot of our interactive GUI system.

3.3 Probabilistic Finite-State Machine (PFSM)

Our idea is inspired by Vidal *et al.* [6] like hidden Markov models. User could create a virtual building model by given the relationships of sub-models with transformation matrices and transition probabilities. Our system could generate mass of various building models by traversing a probabilistic finite-state machine (PFSM) we modified. Although PFSMs are used in a variety of areas, it is impracticable to be applied to our system directly. We found that there are two problems in traditional finite-state machine when applied to our modeling system. The first problem is that the result of building model is not controllable. Second, we need some information such as translation, scaling, rotation, and sub-model ID while constructing. We propose the PFSM to solve the above problems by adjusting the probability of the sub-model. If we want to increase the appearance times of some sub-model, we just increase the probability of this sub-model. Besides, we need to modify the output function of PFSM to provide additional information (transformation matrices and sub-model ID).

We define our PFSM as follows: **M = (Q, I, F, E, T, ω, Γ)** where **Q** is a finite set of states which is mapping to a sub-model, $q \in Q$, **I** is a set of initial states which belongs to Q, **F** is a set of final states, and $q_f \in Q$ is a special (final) state, ω is the output function (ω_j: $(Q-\{q_f\})_j \times p_j \rightarrow \Gamma$, j: a direction, $j \in J$, J = {front, back, left, right, up}, and p_j is the probability for some direction), **E** is a state-based symbol emission probability function (E_j: $(Q-\{q_f\})_j \times \mathrm{Adj}(Q-\{q_f\})_j \rightarrow P$), **T** is a transition set from a state to another state in one direction (T_j: $(Q-\{q_f\})_j \times p_j \rightarrow Q$, subject to the following normalization conditions: $\sum_{q' \in Adj(q)} E_j(q,q') = 1$, $\forall q \in (Q - \{q_f\})$, and Γ is the output parameters which contain sub-model ID and transformation matrices including translation, scaling, and rotation.

We start from an arbitrary initial state as a root in the PFSM, and the probabilistic transition between states is determined by the emission probability function E. In PFSM, state transition T_j is to determine which state is the next state using the j direction of current state. When we get the final state, a new building model is constructed by traversing the PFSM.

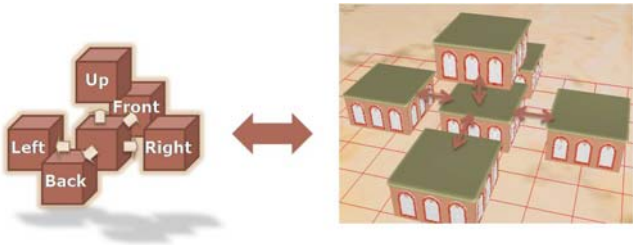


Fig. 10: A simple example of a virtual model.

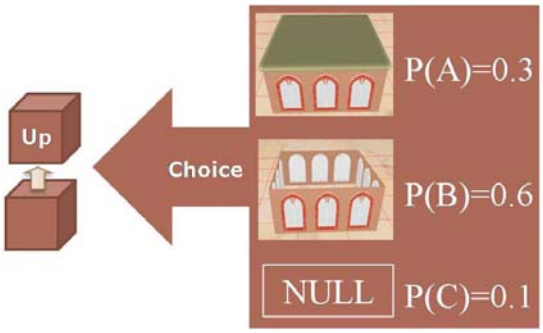


Fig. 11: There are three different probabilities for three different sub-models respectively.

3.3.1 PFSM Construction

We provide an easy way to use the GUI system to construct the PFSM for each sub-model. In Figure 10, we show a simple example of a virtual model. Take a look at the center sub-model of the virtual model. There are five directional neighbors including front, back, left, right, and up to let user to determine some information about transformation matrices, probability, and sub-model ID for each neighbor by using the GUI system. For example, in

Figure 11, consider the up direction which we give three different probabilities for three different sub-models. While traversing the PFSM, we can get different building models. If we want to control the results of building model, we just adjust the probabilities of sub-model. Beware that the summation of all probabilities would be one. In addition, we could also modify the script file directly according to the data structure we defined in Figure 12. Figure 13 shows a script example of Figure 10.

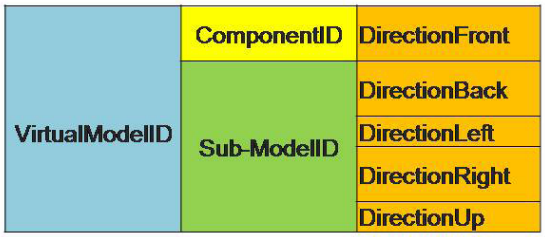


Fig. 12: The data structure of virtual model.

```
//VirtualModelNameID=VirtualModelName
//VirtualModelID-ComponentID=Sub-modelID,
//VirtualModelID-ComponentID-Direction-Counter=ComponentID,Translate,Scaling,Rotation,Probability
VirtualModelName2=A VirtualModel3
VirtualModel2-0=I1,
VirtualModel2-1=I1,
VirtualModel2-2=I1,
VirtualModel2-3=I1,
VirtualModel2-4=I1,
VirtualModel2-0-0-0=I1,T0,0,1.6,S1,1,1,R0,0,0,P1.0,
VirtualModel2-0-1-0=I2,T0,0,-1.6,S1,1,1,R0,0,0,P1.0,
VirtualModel2-0-2-0=I3,T-1.6,0,0,S1,1,1,R0,0,0,P1.0,
VirtualModel2-0-3-0=I4,T1.6,0,0,S1,1,1,R0,0,0,P1.0,
VirtualModel2-0-4-0=I0,T0,0.8,0,S1,1,1,R0,0,0,P0.7,
VirtualModel2-0-4-1=I-1,T0,0.8,0,S1,1,1,R0,0,0,P0.3,
```

Fig. 13: A script example of a virtual model in Figure 10.

3.3.1 Model Construction Algorithm Base on PFSM

Base on PFSM, we proposed an algorithm to construct a building model. The pseudo code is shown in Figure 14. The result of virtual model is M = (Q, I, F, E, T, ω, Γ). First, we initiate a queue and check the five directional neighbors to see whether the root can connect a sub-model or not by using T_j for each root as initial state q_i of I. Additionally, we also check each directional neighbor to see whether it satisfies the probability defined by E_j. Our system gets the parameters for each satisfied check and constructs a new building model based on Γ. Therefore, we can run the process several times and get several different building models for each time. If the result is not quite well, we can adjust the transformation matrices or probabilities of sub-model by using GUI system or modifying script file directly. Figure 15 shows two different examples of virtual building model.

4. EXPERIMENTAL RESULTS

The snapshot of our interactive GUI system of building modeling is shown in Figure 9. We can model buildings easily through our GUI system with some operations. User can not only select the basic component of five levels (including terminal, column, face, shape, sub-model, and model) which user wants to manipulate from the dropdown menus but also choose the desired operations to create, delete, modify, or transform the basic components. In addition, user can modify the properties such as transformation, probability, and relationship, and our GUI system will display each modified result immediately.

```
init a queue Q
A model M= ∅
for each root q of a virtual model
{
    Q.push(q)
    while(Q is not null and Iteration Number<MaxIteration)
    {
        q = Q.pop()
        for all direction j
        {
            synthesize new q' by probability
            M=M U {q'|q ∈ Adj(q), if Ej(q,q')is satisfied}
            Q.push(q')
        }
    }
    Output a model M and M.clear()
}
```

Fig. 14: The pseudo code of our proposed algorithm.

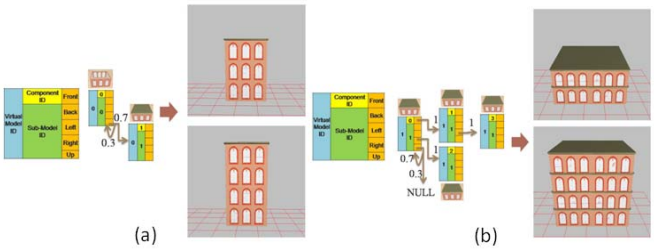


Fig. 15: (a) and (b) are two simple examples of virtual model.

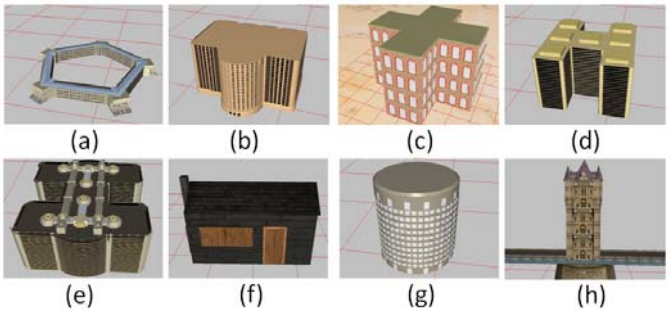


Fig. 16: Some example buildings are constructed by our system.

Table 1: The number of terminals is used for each building model in Figure 16.

Figure	Num. of Terminals
16(a)	4
16(b)	6
16(c)	3
16(d)	6
16(e)	5
16(f)	5
16(g)	4
16(h)	5

We show some examples in Figure 16, and the number of terminals we used is shown in Table 1 for each demonstrated building example. Our first demonstrated building is called “Pentagon” (Figure 16(a)), and the terminals we used and parts of script are shown in Figure 17 and 18 respectively. Second example is called “Skyscraper” (Figure 14(b)), and its terminals are shown in Figure 19. Third, the simplest building is also shown in Figure 16(c). Finally, the more complex building is shown in Figure 20. Therefore, these results show that our interactive GUI system can generate buildings range from

simple to complex. In Figure 21, we demonstrate that our system can create an existing famous building and a new novel building which is composed of the terminals of existing one.

Given the sub-models and the PFSM as shown in Figure 22, our interactive GUI system will produce these examples shown in Figure 23 automatically by assigning different probabilities for each sub-models. Using our interactive GUI system, we only take from half to one hour to create a new building model with PFSM averagely. All we need to do is to subdivide a model into some terminals for input model and to adjust properties (transformation matrices and transition probabilities) of each level of component (such as column, face, shape, and sub-model). Finally, we use the PFSM to produce various new building models and display in our system immediately. Besides, we can also modify the script file directly, and the new building model is also presented immediately. Our proposed interactive GUI system is a flexible and efficient tool for people without domain knowledge.

Table 2 summarizes the characteristics of eight different algorithms with the comparisons of using parameter, user's manipulation, interactive control, requiring photo, domain knowledge, decomposed models, and artistry. In shape grammars [7] and L-system [2], their methods require domain knowledge to control the system well, but users who use our proposed system can produce building models easily even without domain knowledge or for non-expert users. In short, although our system does not work well for all kinds of buildings, it provides a powerful, yet simple to use utility.

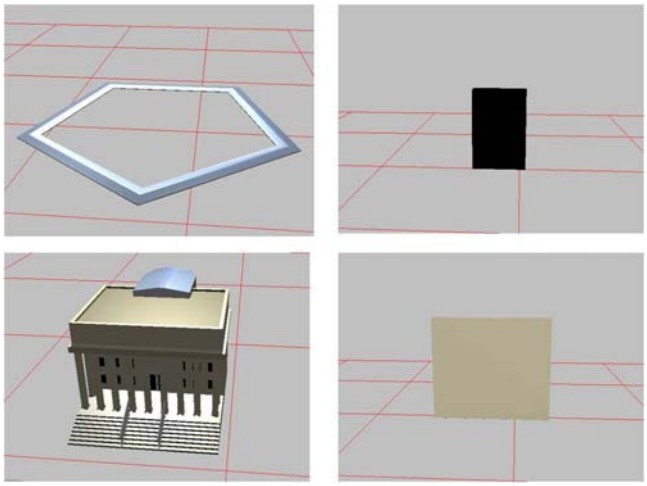


Fig. 17: All terminals are used in the Pentagon.

5. CONCLUSIONS AND FUTURE WORK

We presented an interactive GUI system to model a building with an editable script. We also proposed a probabilistic finite-state machine to construct new novel and various building models easily and rapidly. The proposed GUI system is well appropriate not only for advanced users but also for non-expert users. Our contributions are summarized as follows:

- An interactive GUI system for user to model new buildings.
- Increasing multiplicity of models using PFSM.
- Reducing the quantity of transmission on network; it means that when we transmit the building models, we can just only transmit the properties and terminals to reduce the quantity of transmission on network and to save the transmitting time.

```
[Mesh]
Mesh37=.\\Terminal\\042-五角建築.x
Mesh38=.\\Terminal\\042-屋頂.x
Mesh39=.\\Terminal\\042-wall.x
Mesh40=.\\Terminal\\042-窗戶.x

[TerminalScript]
TerminalName37=042-五角建築
Terminal37=I0,T0,0,0,S0.1,0.1,0.1,R.0,0,0,
TerminalName38=042-屋頂
Terminal38=I0,T0,0,0,S0.1,0.1,0.1,R.0,0,0,
TerminalName39=042-wall
Terminal39=I0,T0,0,0,S0.02,0.02,0.02,R.0,0,0,
TerminalName40=042-窗戶
Terminal40=I0,T0,0,0,S0.1,0.1,0.1,R.0,0,0,

[ColumnScript]
ColumnName32=042-五角建築
Column32-0=I37,T0,0,0,S1,1,1,R0,0,0,
ColumnName33=042-屋頂
Column33-0=I38,T0,0,0,S1,1,1,R0,0,0,
ColumnName34=042-Window+Wall
```

Fig. 18: Parts of script of Pentagon.

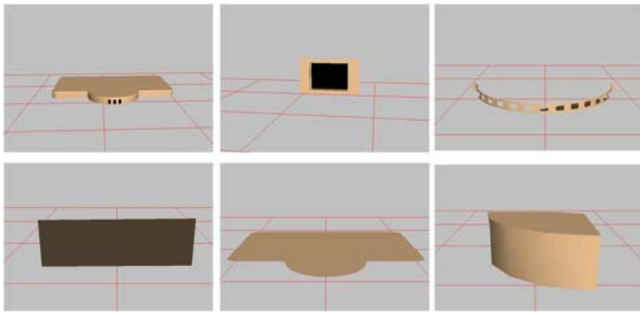


Fig. 19: All terminals are used in the Skyscraper.

Some future work is worthy to be addressed. First, we are exploring to combine our system with an automatic city modeling system. Second, we are also searching for extensions for our GUI system to construct all the buildings of an existing city with some specific style. Moreover, we look forward to improving in model construction and visualization in computer graphics.

REFERENCES

[1] Amit Shesh and Baoquan Chen, et al. "Smartpaper: An Interactive and User Friendly Sketching System," Computer Graphics Forum Vol. 23., No. 3., pp. 301–310, 2004.

[2] Aristid Lindenmayer. "Mathematical Models for Cellular Interaction in Development," In Journal of Theoretical Biology, Vol. 18. pp. 280–315, 1968.

[3] Daniel G. Aliaga, Paul A. Rosen, and Daniel R. Bekins, et al. "Style Grammars for Interactive Visualization of Architecture," IEEE Transactions on Visualization and Computer Graphics Vol. 13. No.4. pp. 786–797, 2007.

[4] Daniel R. Bekins, and Daniel G. Aliaga, et al. "Build-by-number: Rearranging the Real World to Visualize Novel Architectural Spaces," In IEEE Visualization, Vol. 19., 2005.

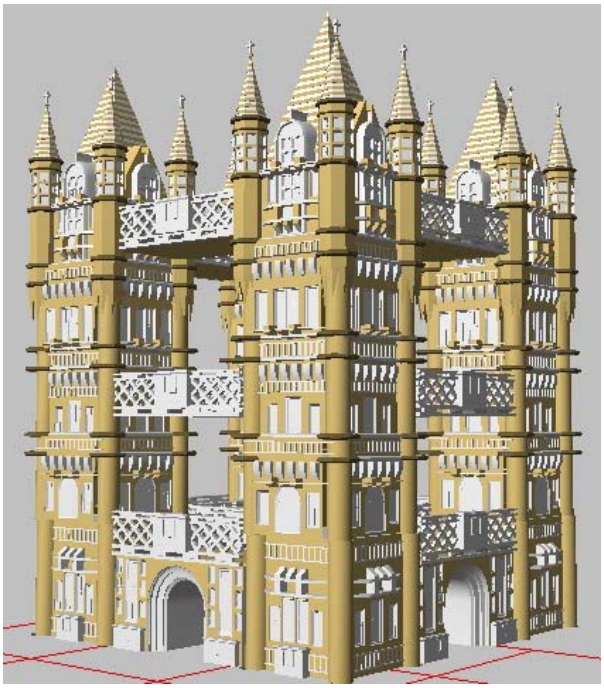


Fig. 20: The more complex building, Cathedral, is constructed by our system.

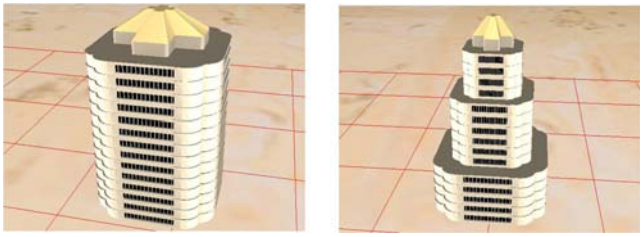


Fig. 21: Two different kinds of skyscraper are constructed by the same terminals.

[5] Donald E. Knuth, et al. "Semantics of context-free languages," Theory of Computing Systems Vol. 2 No. 2. pp. 127–145, 1968.

[6] Enrique Vidal, Frank Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco, et al. "Probabilistic Finite-state Machines–PartII," IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 27., No. 7., pp. 1026–1039, 2005.

[7] George Stiny. "Pictorial and Formal Aspects of Shape and Shape Grammars and Aesthetic Systems," Birkhauser Verlag, Switzerland, 1975.

[8] George Stiny and William J. Mitchell, et al. "The Palladian Grammar," Environment and Planning B: Planning & Design Vol. 5., pp. 5–18, 1978.

[9] GOOGLE, et al. Google Sketchup. Google, www.sketchup.com, 2006.

[10] H. Ehrig, G. Engels, H.-J Kreowski, and Grzegorz Rozenberg, et al. "Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools, " World Scientific Publishing Company, Singapore, 1999.

[11] Ilya Shlyakhter, Max Rozenoer, Julie Dorsey, and Seth J. Teller, et al. "Reconstructing 3D Tree Models from Instrumented Photographs," IEEE Computer Graphics and Applications Vol. 21., No. 3., pp. 53–61, 2001.

[12] Ji-Young Oh, Wolfgang Stuerzlinger, and John Danahy, et al. "Comparing SESAME and Sketching on Paper for Conceptual 3D Design," EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling Vol. 10., No. 2., pp. 81–88, 2005.

[13] Justin Legakis, Julie Dorsey, and Steven J. Gortler, et al. "Feature-based Cellular Texturing for Architectural Models," In SIGGRAPH 2001, Computer Graphics Proceedings, ACM Press / ACM SIGGRAPH, New York, NY, USA, pp. 309–316, 2001.

[14] Martin D. Davis, Ron Sigal, and Elaine J. Weyuker, et al. "Computability, Complexity, and Languages, Second Edition: Fundamentals of Theoretical Computer Science," 2nd ed. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[15] Michael Sipser. "Introduction to the Theory of Computation," 2nd ed. Course Technology, Boston, 1996.

[16] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool, et al. "Image-based Procedural Modeling of Facades," ACM Transactions on Graphics Vol. 26. No. 3. pp. 335–342, 2007.

[17] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer and Luc Van Gool, et al. "Proocedural Modeling of Buildings," ACM Transactions on Graphics Vol. 25. No. 3. pp. 614–623, 2006.

[18] Paul Merrell. "Example-based Model Synthesis," In I3D '07: Proceedings of the 2007 symposium on Interactive 3D Graphics and Games, ACM Press, New York, NY, USA, pp. 105–112, 2007.

[19] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky, et al. "Instant Architecture," ACM Transactions on Graphics Vol. 22., No. 4., pp. 669–677, 2003.

[20] Przemyslaw Prusinkiewicz and Aristid Lindenmayer, et al. "The Algorithmic Beauty of Plants," 1st ed. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[21] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes, et al. "Sketch: An Interface for Sketching 3D Scenes," In SIGGRAPH '07: ACM SIGGRAPH 2007 courses, pp. 19, 2007.

[22] Yoav I. H. Parish and Pascal Müller, et al. "Procedural Modeling of Cities," In SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and Interactive techniques, pp. 301–308, 2001.

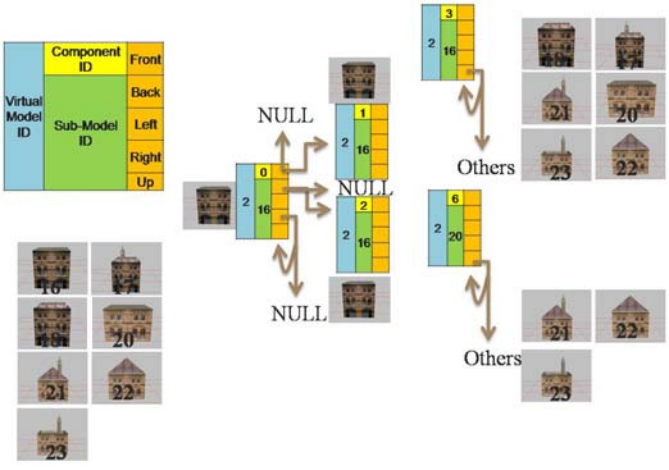


Fig. 22: The PFSM with sub-models is used for constructing different buildings in Figure 23.

Table 2: Comparison between eight different methods.

Method	Parameter	Manipulation	Interactive	Photo	Knowledge	Decomposed model	Artistry
Shape Grammars [7]	Y	N	N	N	Y	Y	N
L-system [2]	Y	N	N	N	Y	Y	N
Procedural [17]	Y	N	N	N	Y	N	N
Sketch-based[21]	N	Y	Y	N	N	N	Y
Example-based [18]	Y	Y	N	N	Y	Y	N
Build-by-Number [4]	Y	Y	Y	Y	Y	N	N
Style Grammars [3]	Y	Y	Y	Y	Y	N	N
Our method	Y	Y	Y	N	N	Y	N

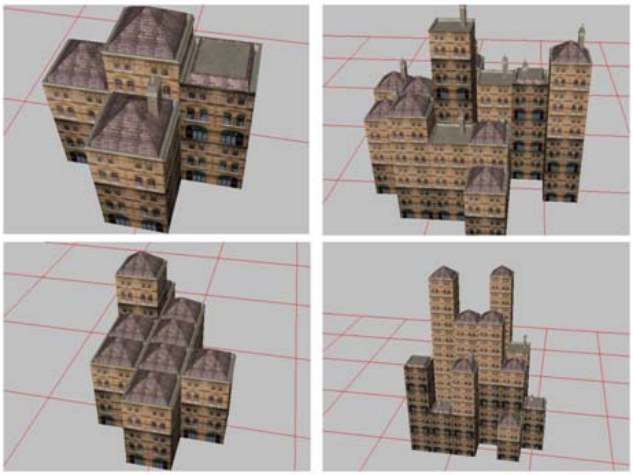


Fig. 23: Different example buildings are constructed by the PFSM in Figure 22.