Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Synthesis of 3D models by Petri net*

Mo-fei SONG†, Zheng-xing SUN†‡, Yan ZHANG, Fei-qian ZHANG
(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210046, China*)
†E-mail: murphysong@gmail.com; szx@nju.edu.cn

Received Dec. 29, 2012; Revision accepted May 21, 2013; Crosschecked June 6, 2013

Abstract: This paper presents a synthesis method for 3D models using Petri net. Feature structure units from the example model are extracted, along with their constraints, through structure analysis, to create a new model using an inference method based on Petri net. Our method has two main advantages: first, 3D model pieces are delineated as the feature structure units and Petri net is used to record their shape features and their constraints in order to outline the model, including extending and deforming operations; second, a construction space generating algorithm is presented to convert the curve drawn by the user into local shape controlling parameters, and the free form deformation (FFD) algorithm is used in the inference process to deform the feature structure units. Experimental results showed that the proposed method can create large-scale complex scenes or models and allow users to effectively control the model result.

Key words: Petri net, Inference, Interactive modeling, Model by example
doi:10.1631/jzus.CIDE1305 **Document code:** A **CLC number:** TP301.6

1 Introduction

As a new generation of digital media types, digital geometric models are becoming widely used in industrial manufacturing, digital entertainment, biomedical engineering and digital cultural heritage protection. As a result, how to reuse model resources becomes an important issue. Existing solutions focus on 3D model retrieval (Funkhouser *et al*., 2003; Tangelder and Veltkamp, 2004), which can obtain models only from high volumes of existing resources. To take full advantage of existing models, Funkhouser *et al*. (2004) proposed the model by example system, and Catalano *et al*. (2011) adopted this idea when tackling the challenging issues of semantic digital geometry processing.

Consequently, numerous example-based modeling techniques have been presented (Funkhouser *et al*., 2004; Chaudhuri and Koltun, 2010; Chaudhuri *et al*., 2011; Kalogerakis *et al*., 2012; Xu *et al*., 2012). Such algorithms can create new shapes by recombining parts of different models. However, structural relationships are limited in such a way that these methods cannot alter the decomposition of existing models. Recently, a number of more promising approaches adopt procedural modeling ideas for topology variations by extending and replicating the adjacent or repeated patterns (Bokeloh *et al*., 2010; Merrell and Manocha, 2011). However, these technologies cannot provide a free-form deformation editing interface. Bokeloh *et al*. (2011; 2012) presented a pattern-aware shape deformation method based on an elastic or algebraic model, which realizes free form deformation through pattern stretching, but users can edit the shape only by stretching the controlling points, which can be an inconvenient method. In addition, structure variation is limited to adding or removing elements. The inference-based modeling method (Biggers and Keyser, 2011) introduces Petri net to describe the modeling process, which provides

‡ Corresponding author
* Project supported by the National Natural Science Foundation of China (Nos. 61272219, 61100110, and 61021062), the National High-Tech R&D Program of China (No. 2007AA01Z334), the Program for New Century Excellent Talents in University (No. NCET04-04605), the Science and Technology Program of Jiangsu Province (Nos. BE2010072, BE2011058, and BY2012190), and the Graduate Training Innovative Projects Foundation of Jiangsu Province (No. CXLX12_0054), China

a more intuitive sketch-based interface for controlling the skeleton of the result and improves the global deformability by articulation, repetition, and interchanging of parts, but this surface-based representation method often produces redundant surfaces giving the example model a complex internal mesh structure. Besides this, this method does not provide local surface deformation. Therefore, a more appropriate representation and deformation mechanism is required to improve the process representation efficiency and deformation capacity.

In this paper, a synthesis method of 3D models using Petri net is presented. The proposed method has two main advantages. First, 3D model partitions are defined as feature structure units. By structural analysis, their shape features and adjacent constraints are computed, and then Petri net is used to describe and control the modeling process, including two inference processes, structure extending and piece deformation. Second, a construction space is introduced to convert the user-drawn curve into the shape controlling parameters, which are used to manipulate the shape and size of the modeling result, and then the free form deformation (FFD) algorithm (Sederberg and Parry, 1986) is used to deform feature structure units, in order to support local shape and global structure variations, as determined by simultaneous user control.

2 Overview of our method

The 'analyze-and-edit' framework (Gal *et al*., 2009) is applied to realize a modeling process based on model analysis, using a mesh model as the input. When used in accordance with user-provided size parameters and templates, our approach can generate a new model that maintains the shape and structure features of the example model. Fig. 1 shows the process flow, divided into four steps: model analysis, construction space creation, Petri net initialization, and inference-based modeling.

Model analysis: The goal of this step is to create a graph structure to describe the construction of the example model. First, an annotation-based segmentation method is used to extract the model partitions from the example model. The model partitions can then be classified into several categories in order to create the graph structure.

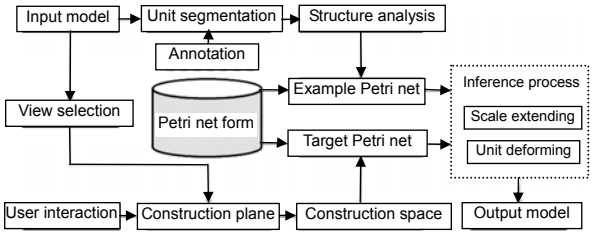


Fig. 1 The proposed process flow

Construction space creation: The method proposed in this paper provides a sketch-based interface to control the global shape of the modeling result. Users can draw a curve to set the extending direction of the model. The drawn curve is converted to a construction plane, which is then swept along one axis to generate a construction space whose extending direction is consistent with the curve.

Petri net initialization: Two Petri nets need to be initialized, the example net and the target net. The example net is created according to the graph structure of the example model. A local Petri net is created which records the features in each partition and connects adjacent nets to construct the example Petri net. The target net is created according to the construction space, where a local Petri net containing shape information records from every subspace connects adjacent nets to construct the target Petri net.

Inference-based modeling: According to the Petri net structure, the states and parameters of the nodes in the target Petri net are updated by an inference-based method. First the adjacent constraints are used to extend the model and then shape controlling parameters are used to deform the model partitions using the FFD algorithm.

2.1 Model analysis

Model analysis is the basis of the modeling process. The model partitions are extracted from the example model, and the partitions are categorized into several classes according to their shape features. The whole process requires two steps, piece segmentation and structure analysis.

2.1.1 Piece segmentation

The aim of this step is to create the model partitions. To extend the example model, it must be ensured that the partition has as many self-similar pieces as possible. Because the majority of models handled

have regular structures, a lattice is used to split the example model, each unit of the lattice being occupied by one segmented model piece. Creating the required lattice for maximizing the number of self-similar pieces is not a simple task, and the key is to define the scale of one lattice unit. To solve this, the user marks one single self-similar segment on the orthographic view of the example model (Fig. 2), by drawing a rectangle to determine the scale of one lattice unit (Fig. 2b). With that input, the required lattice structure can then be created by replicating the drawn rectangle along the X- and Y-axis of the view. The model partition $\{e_{i,j,k}\}$ is then created by dividing the example model with the lattice, where the subscripts i, j, k are the parameterized coordinates, indicating the position of a unit in the lattice (Fig. 2c).

Fig. 2 Decomposition process
(a) Input model; (b) Users' annotation (a rectangle is drawn to determine the scale of one lattice unit); (c) Decomposition result obtained using the proposed algorithm

2.1.2 Structure analysis

Using the example model and the model partition $\{e_{i,j,k}\}$, the structure of the model can be analyzed. The surface area is first computed, with the size of the convex hull as the feature of each piece. Then, the self-similar model piece is extracted and the same label is used to indicate similar model pieces. Each piece $e_{i,j,k}$ is a triangle mesh data, which can be expressed as $e_{i,j,k}=\{P, F\}$, where P is the vertex set and F is the facet set. The geometric feature is expressed as

$$F_e=\{S, X_0, Y_0, Z_0, X_1, Y_1, Z_1, f_{000}, f_{001}, f_{010}, f_{011}, f_{100}, f_{101}, f_{110}, f_{111}\}. \quad (1)$$

The meaning of each attribute is shown in Table 1. According to the feature vector, we classify the model pieces and use a unique label L to indicate each category of model partition.

After structural analysis, a graph structure is created by setting each model piece $e_{i,j,k}$ as the graph node. These nodes are connected with an edge in the graph where their parameterized coordinates i, j, k and i', j', k' satisfy $\|[i, j, k]-[i', j', k']\|=1$.

Table 1 Meaning of each attribute of the feature vector

Attribute	Meaning
S	Sum of surface area on triangle faces
$\{X_0, Y_0, Z_0, X_1, Y_1, Z_1\}$	The maximum and the minimum coordinates of the vertices
f_{ijk}, $i, j, k \in \{0, 1\}$	If $[X^i, Y^j, Z^k] \in P$, $f_{ijk}=1$, and vice versa

2.2 Construction space creation

After obtaining the example graph structure, the model pieces can be used to fill a subdivision space, or the construction space, and each subspace of the space is occupied by one model piece. When all subspaces are filled with a model piece, the result can be created by combining the filling model pieces. The construction space determines the arrangement of the model pieces and the size of the result, while users can control the shape by interactively defining the construction space. The approach proposed in this study provides a sketch-based interface to define the construction space, wherein users can select a suitable perspective to project the example model, and then draw a curve on the projection plane to represent the extending direction of the required construction space. Based on this user input, this method creates the construction space, and the process is described as follows.

Because the user draws the curve onto the projection plane, the curve reflects only the 2D projection information of the construction space. Therefore, the construction space creation is divided into two steps: first, the construction plane is created which is the projection of the construction space, and then the construction space can be achieved by sweeping the plane along the projecting direction. The construction space must be consistent with the drawn curve, and the grid points cannot be self-occluded. The process flow is as shown in Fig. 3, and the algorithm is as shown in Algorithm 1. The size of the example model's rectangular convex hull is (l_X, l_Y, l_Z), the numbers of units along each axis are N_X, N_Y, N_Z, and the user-drawn curve is along the X-axis on the XY projection plane, with the set sizes of the Y- and Z-axis being S_Y and S_Z respectively. The drawn curve

is expressed as $C=\{\boldsymbol{P}_i\}$ (Fig. 3a), where $\boldsymbol{P}_i$ is the segmentation point.

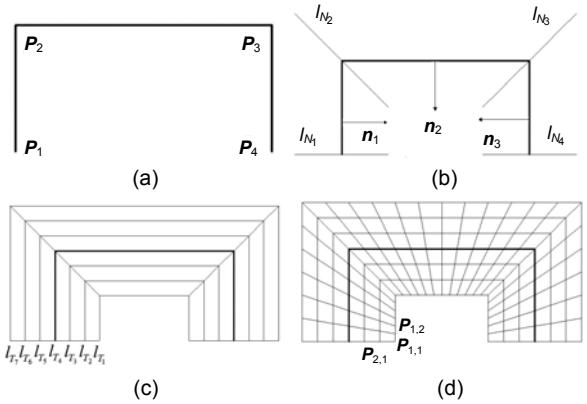


Fig. 3 Construction plane generation

(a) User provided lines; (b) Computing extending lines at the articulation point; (c) Sampling and connecting the sample points on the extending lines; (d) Sampling and connecting the sample points on the tangential lines

Algorithm 1 Construction plane generation

1 Compute the length of the curve $l=\sum\|\boldsymbol{P}_i-\boldsymbol{P}_{i+1}\|$
2 Compute the number of units along X-axis, $S_X=l/l_X \cdot N_X$
3 Compute the normal direction of each curve segment: $\{\boldsymbol{n}_i|\boldsymbol{n}_i\cdot(\boldsymbol{P}_i-\boldsymbol{P}_{i+1})=0\}$ (Fig. 3b)
4 for $k\leftarrow 1$ to $|\boldsymbol{P}|$ do
5 Compute the normal extending lines l_{N_i} and direction $\boldsymbol{d}_i$ on each point $\boldsymbol{P}_i$ (Fig. 3b)
6 if $i=|\boldsymbol{P}|\vee i=1$ then
7 $\boldsymbol{d}_i=\boldsymbol{n}_{i=1?1:|\boldsymbol{P}-1|}$
8 else
9 $\boldsymbol{d}_i=\boldsymbol{n}_{i-1}+\boldsymbol{n}_i$
10 end if
11 end for
12 Compute S_Y sample points: $\boldsymbol{P}_{N_{i,j}}=\boldsymbol{P}_i\pm(l_Y/N_Y \cdot j)\boldsymbol{d}_i$
13 Generate the tangential extending lines $\{l_{T_j}|1\le j\le S_Y\}$ by connecting the sample points (Fig. 3c)
14 Compute the sampling length of each tangential extending line: $S_{T_j}=\sum\|\boldsymbol{P}_{N_{i,j}}-\boldsymbol{P}_{N_{i+1,j}}\|/S_X$
15 Compute the sample points $\{\boldsymbol{p}_{i,j}\}$ on each tangential extending lines by length S_{T_j} (Fig. 3d)
16 Connect the corresponding points by normal and tangential lines to create the construction plane

By Algorithm 1, the required construction plane is created, expressed as $P_{\text{grid}}=\{\boldsymbol{p}_{i,j}|1\le i\le S_X,\ 1\le j\le S_Y\}$ (Fig. 3d), and the extending direction of the plane is consistent with the stroke drawn by users (thick lines in Fig. 3).

The construction space is then computed by translating the grid points along the Z-axis:

$$\boldsymbol{p}_{i,j,k}=[p_{i,j}.X,\ p_{i,j}.Y,\ k\cdot l_Z]. \tag{2}$$

All the points are merged to create the construction space, which is a grid space of size $N_X \cdot N_Y \cdot N_Z$. The space is expressed as a 3D grid point set $P_{\text{grid}}=\{\boldsymbol{p}_{i,j,k}|1\le i\le S_X,\ 1\le j\le S_Y,\ 1\le k\le S_Z\}$, which is used by the FFD method as the shape controlling space of model pieces during inference processes.

2.3 Petri net initialization

The proposed modeling process builds on the idea of model synthesis (Merrell and Manocha, 2011), while previous model syntheses use orthogonal lattice as their construction space. The construction space proposed in this work maintains the topology but every subspace can be an arbitrary hexahedron. To manage the necessary controlling parameters, a representation mechanism is required. Biggers and Keyser (2011) used a Petri net to describe the incremental surface fitting process. Inspired by this method, the proposed approach uses Petri net as the representation mechanism and 3D model pieces as the feature structure units. Compared with the surface fitting process, the complexity is significantly increased because the extending dimension changes from 1D to 3D. To solve this, a new colored Petri net is designed, which adds label and attribute definitions to the nodes of the Petri net. Using this method, the constraint relation can be represented along different directions to realize multi-dimensional extending.

Petri net is a bipartite graph in which the nodes represent transition and place (Peterson, 1977). The place node describes the system state, and the transition node describes the events between different place nodes. Because it is a powerful tool to describe the asynchronous concurrent system, it is suitable for representing model synthesis. In this study, the place corresponds to every subspace state in the filling process, and in every transition node, the corresponding operation which computes the parameters of the output place according to the input parameters is defined. The whole modeling process is divided into two stages: structure extending and piece deformation. Accordingly, three types of places and three types of transitions are defined in every feature structure unit space (Fig. 4).

Along the flow direction, the nodes are sequentially placed: updating transition, candidate place,

selecting transition, filled place, deforming transition, and deformed place. The colored Petri net structure records the piece feature of the example and shape information of the construction space in a 'token' structure:

$$T_{\text{token}}=\{s, s_{\text{u}}, \boldsymbol{C}, S_{\text{c}}, l, H\}. \quad (3)$$

Herein $s\in$ {'selectable', 'fillable', 'filled', 'deformed'} is the state flag, which represents the triggering condition of each transition; the updating state flags $s_{\text{u}}=\{f_{\text{u}}, f_{\text{d}}, f_{\text{l}}, f_{\text{r}}, f_{\text{f}}, f_{\text{b}}\}$ represent whether the six adjacent subspaces have already updated their states (if $s_{\text{u}}.f=1$, the node has not updated its state yet, vice versa); $\boldsymbol{C}=\{x, y, z\}$ is the parameterized coordinate; S_{c} is the candidate label set which represents the pieces that can be filled in the space; l is the result filled label; $H=\{\boldsymbol{p}_i|i=1, 2, ..., 8\}$ is the hexahedral shape controlling space with eight controlling points. The example and target Petri nets can then be created, respectively.

Example Petri net: The example Petri net is created in line with the graph structure of the example model. A local Petri net is created in each graph node (Fig. 4) and the corresponding token parameters are set. These parameters include the parameterized coordinate $\boldsymbol{C}$ and the label l. The place and transition between the nodes when the nodes are adjacent in the graph are then connected (Fig. 4). The Petri net implicitly defines the adjacent constraint of the example model (Merrell and Manocha, 2011), which is expressed by the label pair $<L_1, L_2>$. The direction between L_1 and L_2 is shown as v, and the adjacent constraint can be expressed as

$$((T_1.C-T_2.C=v)\wedge T_1.l=L_1\wedge T_2.l=L_2) \leftrightarrow <L_1, L_2>=1, \quad (4)$$

where T_1 and T_2 are two tokens in the Petri net. The Petri net structure not only describes the model synthesis process but also contains the constraints that the new model needs to maintain.

Target Petri net: According to the parameters of the construction space, the corresponding target Petri net is created, so that each subspace corresponds to a local Petri net structure (Fig. 4), which has the same structure as the example Petri net. The places and transitions are also connected between the adjacent subspaces. The parameters of the target Petri net are then initialized, which includes setting the state flag as 'selectable', and setting the hexahedral shape controlling space and parameterized coordinates of each subspace according to the construction space setup. The initial candidate label set should include all the labels from the example model pieces. After parameter initialization, the target Petri net can be created, determining features' size and shape of the required model.

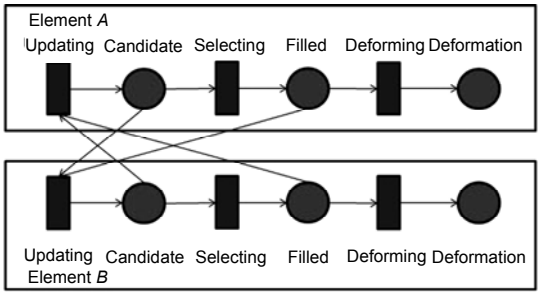


Fig. 4 Structure of the proposed Petri net
Places are illustrated as circles and transitions as rectangles. The lines show flow relation

2.4 Inference-based modeling

After creating the target Petri net, an inference-based modeling method is used to fill the model pieces within the construction space. The process includes two steps: structure extending and piece deformation. In the first step, the updating transition and selecting transition nodes select a suitable piece to fill the 3D construction space, and its main task is to compute the candidate label set of each subspace according to adjacent constraints, which are defined in the example Petri net. In the second step, the deforming transition uses the FFD algorithm to deform the pieces according to the corresponding shape controlling space. The inference process is driven by a message propagation method (Algorithm 2).

As shown in Fig. 5a, the example model has already been segmented and the same model segments are labeled correspondingly by the same number. The whole inference process generates a new model that maintains the features of the example model. Fig. 5b shows the starting state of the target Petri net. The inference process based on the Petri net triggers any viable transition to update the state of the net system iteratively until the modeling process is over. The proposed approach sets the subspace whose parameterized coordinates are (0, 0, 0) as the starting node. We use the piece in the subspace whose parameterized coordinates are (0, 0, 0) of the example model

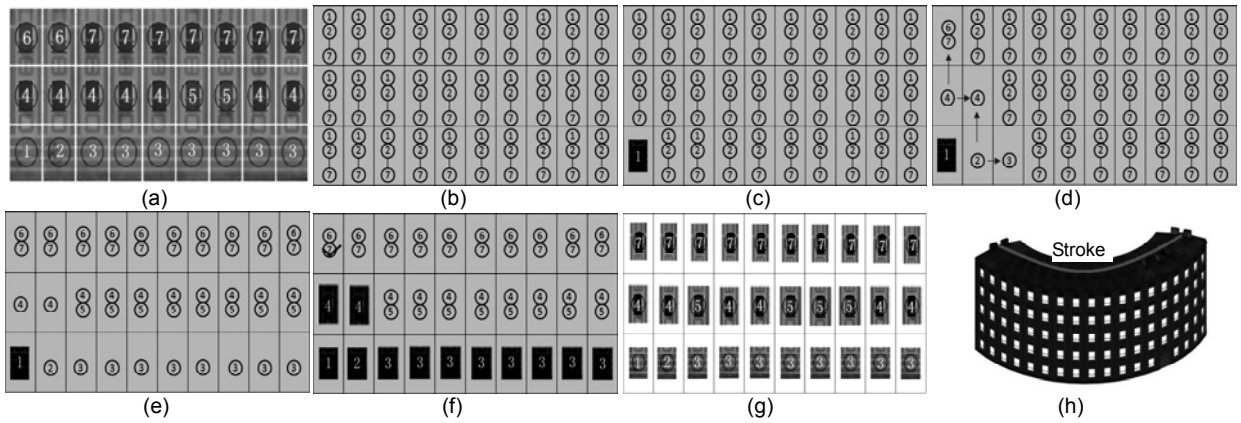


Fig. 5 Inference-based modeling process

(a) An already segmented input model with corresponding segments labeled accordingly; (b) Petri net initialization; (c) Filling the origin element; (d) Iteratively running transition updates; (e) Stable state after updating; (f) Enforcing selecting transition; (g) Filled elements; (h) Running deforming transition

Algorithm 2 Inference-based modeling

1 Set the origin subspace as the starting node
2 Set the token parameters of the starting node
3 Search for the place node whose state flag is 'selectable'. If there exists an updating state flag, trigger the updating transition of the corresponding subspace
4 Search for the place node whose state flag is 'fillable', and trigger the selecting transition
5 Search for the place node whose state flag is 'filled', and trigger the deforming transition
6 If there is not any valid transition, search for the place node whose state flag is 'selectable', enforce the selecting transition, and go to step 3
7 Trigger the valid transition iteratively until all the state flags of the nodes are 'deformed'

to fill this subspace and then modify the result label as the filled label and the state flag as 'filled' in the token structure (Fig. 5c). Meanwhile, the starting node sends messages to adjacent nodes and triggers their updating transition (Fig. 5c). Then, the inference process triggers any viable transition to update the state of the system iteratively. Three types of transitions are involved: updating, selecting, and deforming. The details of each transition are shown as follows:

Updating transition: This transition updates the candidate label set according to the candidate label set of adjacent nodes, and the updating result must satisfy these adjacent constraints, defined as Eq. (4), in the example Petri net. If the candidate label set has one label that cannot be adjacent to any label in the candidate label sets of adjacent nodes, the adjacent constraints are unsatisfied and the label is removed from the candidate label set. If the candidate label set has only one element in place after transition, the state flag is set to 'fillable'. If S_c has changed, all the updating state flags in set s_u of place node P_O are set to 1 so as to trigger the updating transition of the adjacent nodes. Fig. 5d shows the results of executing the transition twice, demonstrating that the elements in the label set all satisfy the adjacent constraints after updating.

Selecting transition: The aim of this transition is to select a label from the candidate set and use the corresponding piece to fill the subspace. When the state flag of one node is 'fillable', the transition is triggered and the result label is set as the unique validating label. Then the state flag is modified to 'filled' to trigger the deforming transition. When no viable transition exists (Fig. 5e), the state of the system becomes stable, which means all the labels in the candidate label set satisfy the adjacent constraints. At this time, if the state flag of one node is 'selectable', the selecting transition is forced to be executed. The nodes are searched for 'selectable' state flags, and the node with the highest priority is selected. The smaller the size of the candidate label set, the higher the priority of the space. The selecting transition of the corresponding place node is triggered. One label is selected from the candidate set randomly as the result label and the other labels are removed from the candidate label set. The corresponding piece is then used to fill the space and modify the state flag as 'filled'. Because the candidate label set has changed, we set all the updating state flags of this node as 1 and continue the inference process. Fig. 5f shows the result of

enforcing selecting transitions.

Deforming transition: When the state flag of one node is 'filled', this transition can be triggered. Taking the filling piece in the subspace of [x, y, z], the initial shape controlling space is a cuboid, which has eight controlling points. We first use the hexahedral shape controlling space parameters in the target Petri net as the deforming controlling points P_G. Deforming transition then uses the FFD algorithm to deform the filled piece according to the shape controlling points P_G.

When all the state flags are 'deformed', we create a valid result state of Petri net, which means that all the subspaces are filled with deformed pieces (Fig. 5g). The deformed pieces can then be assembled directly to achieve the result model (Fig. 5h).

3 Results and evaluation

Certain example processes have been implemented to show the efficacy of the proposed method. The experimental environment was Intel Core i5-2400, 3.10 GHz CPU, with 8 GB of memory. Two applications were tested with the methods outlined above, i.e., single object editing and large scene synthesis.

3.1 Single object editing

Fig. 6 shows the results of our proposed method for editing models. The input models include an office building, a residential building, a bridge, and a road. Compared with the inference-based method (Biggers and Keyser, 2011), our method can create complex scenes from closed curves and curve networks with intersecting curves (Fig. 6d). The results show that the proposed method is able to create variations of shape and structure simultaneously, while preserving the adjacent constraints in the example. Other models can also be created to have the same overall shape as the real-world data (Fig. 7), where the global shape of the building is the same as that of the Pentagon.

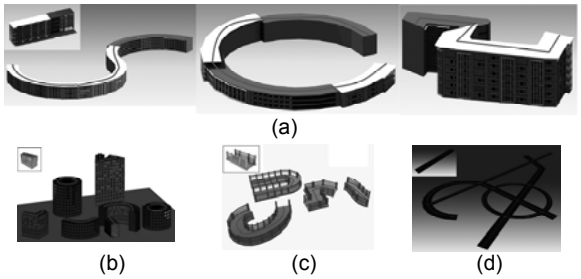


Fig. 6 Single object editing

(a) Office building; (b) Residential building; (c) Bridge; (d) Road. The thick line in each subfigure is the curve drawn by the user to control the extending size and direction

Fig. 7 Our modeling results (a) compared to the real image showing the Pentagon (b)

The proposed method can be compared with structure-aware shape editing methods (Bokeloh *et al*., 2011; 2012). Fig. 8 shows the number of editing operations, and the time cost by extending and by deforming. Fig. 8a shows that our interactive approach is very facile for users to effectively control the result, in particular for curved shapes. As shown on the left in Fig. 6a, users need only to draw one stroke using the proposed method, while Bokeloh's method requires more than 20 editing operations, which introduces more difficulty of interaction. Besides this, the FFD method used by the proposed approach is faster than the model-based deformation used by Bokeloh's method (Fig. 8c). Fig. 8b shows that the time cost of the extending process in the proposed method is, however, larger than that in Bokeloh's method. This is

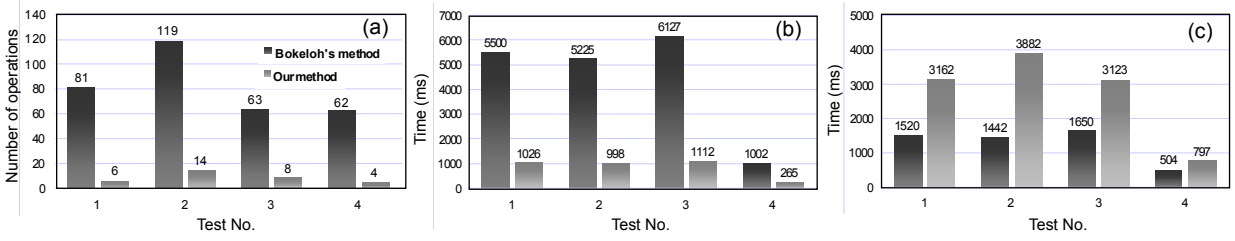


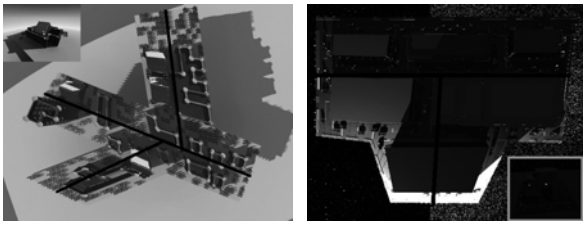
Fig. 8 Comparison between our method and Bokeloh's method

(a) Operation number; (b) Time cost by deforming; (c) Time cost by extending

because the proposed extending process uses context-sensitive construction rules, which needs an online constraint analysis process, while Bokeloh's method uses context-free construction rules.

3.2 Large scene synthesis

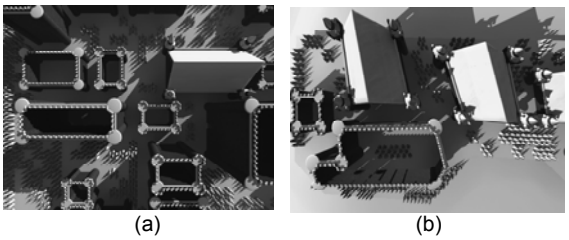
Fig. 9 presents the results of scene synthesis using the proposed method. The input models are two simple scenes, a city and a castle. Users draw strokes to create a corresponding construction space for scene synthesis (the thick line in Fig. 9). The skeleton of the global space is consistent with the stroke, so that users can control the global size and shape of scene.



(a) (b)

Fig. 9 Large scene synthesis: (a) castle; (b) city

In this respect, the proposed method can be compared with Merrell and Manocha (2011)'s synthesis method. As shown in Fig. 10, Merrell and Manocha (2011)'s synthesis method is limited on its parallel grid structure, while our proposed method can create models with a non-parallel structure to achieve more innovative results because of the shape parameters defined in the Petri net. To verify this, a normal histogram can measure the degree of difference between the example and the results. The normal is a basic and important geometric feature, and has been widely used in model analysis, model retrieval, and geometric modeling. The normal of each face of the mesh is first computed, and the X and Y components selected. Then the normal histogram is computed by dividing the 360° into 12 equal parts according to the direction of the normal.



(a) (b)

Fig. 10 Comparison between Merrell and Manocha (2011)'s method (a) and our proposed method (b)

As shown in Fig. 11, the distribution in the normal histogram depicting the results obtained using our proposed method is different from that in the example histogram, while that obtained using Merrell and Manocha (2011)'s method is quite similar to that of the example one. The discriminations between the histograms can be measured by

$$D(H_1,H_2)=\sum_{i=1}^{12}\left(\frac{H_{1i}}{\sum_{j=1}^{12}H_{1j}}-\frac{H_{2i}}{\sum_{j=1}^{12}H_{2j}}\right)^2, \qquad (5)$$

where H_1 and H_2 are the two histograms, and H_{1i} and H_{2i} are the ith components of H_1 and H_2, respectively. As shown in Fig. 12, the discrimination between the histograms showing the results of our proposed method is much larger than that of Merrell and Manocha (2011)'s method. This indicates that our method can create more surprising and inspiring results, improving the diversity of the modeling results.

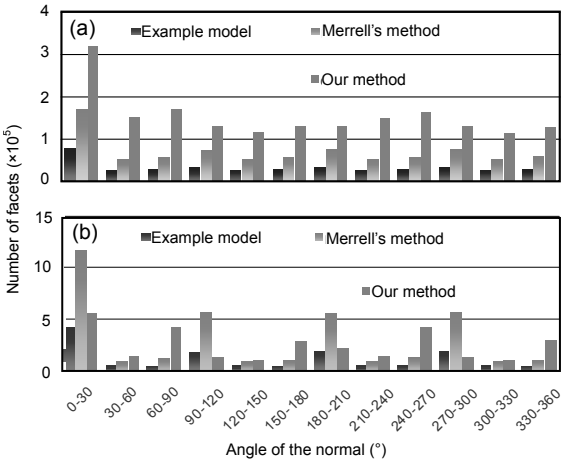


Fig. 11 Normal histogram of the castle model (a) and city model (b)

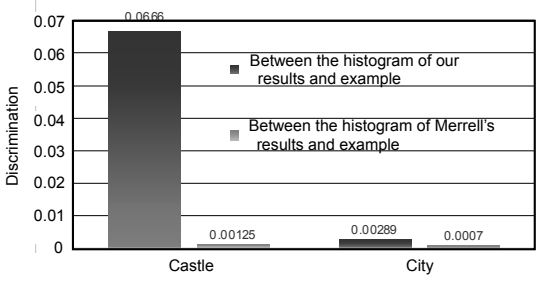


Fig. 12 Comparison between our proposed method and Merrell's method

The proposed method handles large-scale man-made objects well, especially considering that all the models used in the above experiments have more than 200 000 vertices. The experiment results show the adaptability of our 3D modeling method to the scale and type of objects.

4 Conclusions

This paper presents a synthesis method for 3D models using Petri net. A new Petri net structure is designed to describe the modeling process of a 3D model in order to extend the types of feature structure units and the constraint relations. At the same time, a construction space generating algorithm is presented to create mapping between the global structure and local shape, so as to control the global shape. In addition, the FFD algorithm is introduced into the inference process and the model is extended according to adjacent constraints to realize shape and structure variations simultaneously. Experiments showed that the Petri net creates a unified description framework of user interaction constraints and model structure feature constraints, and that our approach can generate large-scale complex models reflecting user intention and maintaining the characteristics of the example model.

Although it is possible for the proposed method to create models with an irregular grid structure, this model analysis method is best suited for models with a simple grid structure, and can handle only self-similarity if transformed through translation. Future work should focus on detecting other kinds of self-similar units, such as invariance under rotation or scaling transformation (Pauly *et al*., 2008). Also, if user annotation is less than accurate, the analysis method may fail to find self-similar units, leading to meaningless synthesis results, and requiring manual adjustments so as to refine the analysis results. If an automatic adjustment strategy can be introduced, the results will be more effective.

References

Biggers, K., Keyser, J., 2011. Inference-based procedural modeling of solids. *Comput.-Aid. Des.*, **43**(11):1391-1401. [doi:10.1016/j.cad.2011.09.003]

Bokeloh, M., Wand, M., Seidel, H., 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph.*, **29**(4), Article 104, p.1-10. [doi:10.1145/1778765.1778841]

Bokeloh, M., Wand, M., Koltun, V., Seidel, H., 2011. Pattern-aware shape deformation using sliding dockers. *ACM Trans. Graph.*, **30**(6), Article 123, p.1-10. [doi:10.1145/2070781.2024157]

Bokeloh, M., Wand, M., Seidel, H., Koltun, V., 2012. An algebraic model for parameterized shape editing. *ACM Trans. Graph.*, **31**(4), Article 78, p.1-10. [doi:10.1145/2185520.2185574]

Catalano, C.E., Mortara, M., Spagnuolo, M., Falcidieno, B., 2011. Semantics and 3D media: current issues and perspectives. *Comput. Graph.*, **35**(4):869-877. [doi:10.1016/j.cag.2011.03.040]

Chaudhuri, S., Koltun, V., 2010. Data-driven suggestions for creativity support in 3D modeling. *ACM Trans. Graph.*, **29**(6), Article 183, p.1-10. [doi:10.1145/1882261.1866205]

Chaudhuri, S., Kalogerakis, E., Guibas, L., Koltun, V., 2011. Probabilistic reasoning for assembly-based 3D modeling. *ACM Trans. Graph.*, **30**(4), Article 35, p.1-10. [doi:10.1145/2010324.1964930]

Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D., Jacobs, D., 2003. A search engine for 3D models. *ACM Trans. Graph.*, **22**(1):83-105. [doi:10.1145/588272.588279]

Funkhouser, T., Kazhdan, M., Shilane, P., Min, P., Kiefer, W., Tal, A., Rusinkiewicz, S., Dobkin, D., 2004. Modeling by example. *ACM Trans. Graph.*, **23**(3):652-663. [doi:10.1145/1015706.1015775]

Gal, R., Sorkine, O., Mitra, N.J., Cohen-Or, D., 2009. iWIRES: an analyze-and-edit approach to shape manipulation. *ACM Trans. Graph.*, **28**(3), Article 33, p.1-10. [doi:10.1145/1531326.1531339]

Kalogerakis, E., Chaudhuri, S., Koller, D., Koltun, V., 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph.*, **31**(4), Article 55, p.1-11. [doi:10.1145/2185520.2185551]

Merrell, P., Manocha, D., 2011. Model synthesis: a general procedural modeling algorithm. *IEEE Trans. Visual. Comput. Graph.*, **17**(6):715-728. [doi:10.1109/TVCG.2010.112]

Pauly, M., Mitra, N.J., Wallner, J., Pottmann, H., Guibas, L.J., 2008. Discovering structural regularity in 3D geometry. *ACM Trans. Graph.*, **27**(3), Article 43, p.1-11. [doi:10.1145/1360612.1360642]

Peterson, J.L., 1977. Petri nets. *ACM Comput. Surv.*, **9**(3):223-252. [doi:10.1145/356698.356702]

Sederberg, T.W., Parry, S.R., 1986. Free-form deformation of solid geometric models. *ACM SIGRAPH Comput. Graph.*, **20**(4):151-160. [doi:10.1145/15886.15903]

Tangelder, J.W.H., Veltkamp, R.C., 2004. A Survey of Content Based 3D Shape Retrieval Methods. Proc. Shape Modeling Applications, p.145-156. [doi:10.1109/SMI.2004.1314502]

Xu, K., Zhang, H., Cohen-Or, D., Chen, B., 2012. Fit and diverse: set evolution for inspiring 3D shape galleries. *ACM Trans. Graph.*, **31**(4), Article 57, p.1-10. [doi:10.1145/2185520.2185553]