# Automatic Generation Method for Processing Plants

Kentaro Nomura and Kazunori Miyata*

*Japan Advanced Institute of Science and Technology, Japan*
**E-mail address: miyata@jaist.ac.jp*

(Received November 4, 2010; Accepted July 25, 2011)

A processing plant consists of massive parts including tanks, pipelines, processing columns, frames, and so on. This paper reports a method for automatically generating the landscape of a processing plant from a 2D sketch input and some control parameters. This is difficult to implement with conventional procedural methods. The results show that the landscapes of a processing plant are satisfactorily represented, while some detailed parts, such as valves, steps, and branching pipelines, are not generated. The generated 3D geometric data are useful for constructing background scenes in movies and video games, and are also applicable for pre-visualizing a landscape to construct a processing plant.

**Key words:** Procedural Modeling, Processing Plant, Geometry Model

## 1. Introduction

This paper proposes a method for automatically generating the landscape of a processing plant from a 2D sketch input, which is difficult to implement with conventional procedural methods. The generated 3D geometric data are useful for constructing background scenes in movies and video games, and are also applicable for pre-visualizing a landscape to construct a processing plant.

### 1.1 Background

Recently, massive geometric models, such as buildings and forests, are required to produce high-definition computer-generated imagery. The accuracy and the volume of the models are increasing, and consequently the workload in the modeling process becomes heavier. It is therefore important to generate contents procedurally or automatically, so as to produce such models efficiently.

The main target of this paper is processing plants. A processing plant consists of massive parts, and most of the parts are connected to other parts with pipelines. It is very hard to assemble such complicated geometric models manually. Our method can generate the landscape of a processing plant from a 2D sketch input which specifies the outline of an intended model. We do not pursue precision in the functionality of the processing plant, but consider visual satisfaction.

### 1.2 Related works

Grammar-based modeling covers many targets, from generation of natural shapes to architectural models. Procedural Inc. sells software, *CityEngine* (Procedural Inc., 2008), for generating a large-scale 3D urban environment procedurally, including street networks and 3D buildings. Frischer *et al.* have applied this software to rebuild highly-detailed ancient Rome at the peak of the Roman Empire (Frischer *et al.*, 2008). However, this software does not consider connectivity of buildings.

Parish and Müller proposed a system using a procedural approach based on L-system to model virtual cities (Parish and Müller, 2001). Müller *et al.* reported a novel shape grammar, CGA shape, for the procedural modeling of building shells to obtain large scale city models (Müller *et al.*, 2006). They also introduced algorithms to automatically derive 3D models of high visual quality from single facade images of arbitrary resolutions (Müller *et al.*, 2007). Shape Grammar (Stiny, 1980) and Split Grammars (Wonka *et al.*, 2003) are also used for shape, facade and detail modeling. There are automatic generation methods using a model input by a user, to generate a large complex model that resembles the user input (Merrell, 2007; Merrell and Manocha, 2009). These approaches are extensions of texture synthesis based on 3D geometry. There are also ways of combining procedural methods and interactive editing. In (Chen *et al.*, 2008), users can create a street network or modify an existing street network by editing an underlying tensor field.

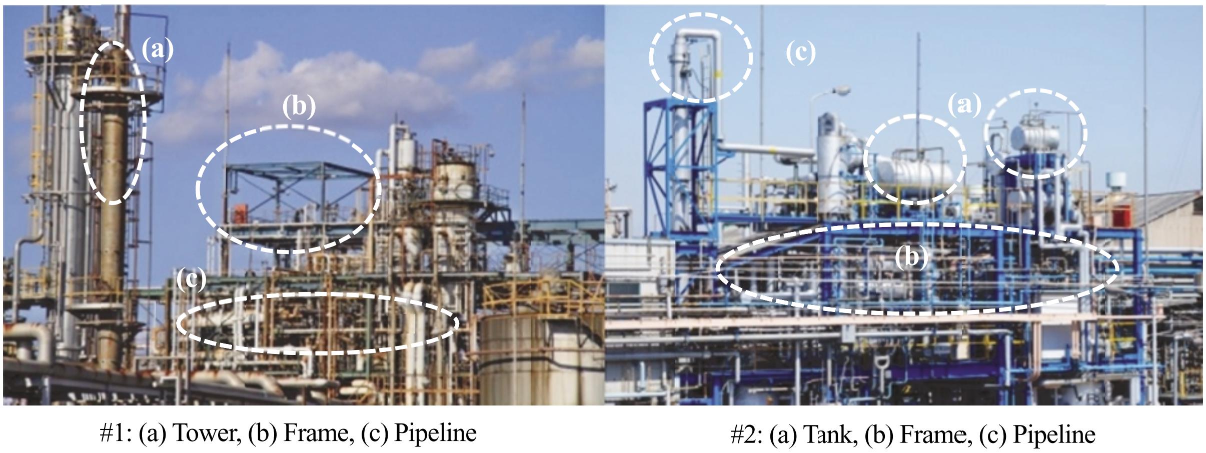
Most of these procedural methods generate virtual cites from two-dimensional images. In contrast, this paper proposes a method to generate a processing plant model which consists of tanks and pipes from two-dimensional sketch input.

Fujita *et al.* presented a constraint-directed approach for layout design of power plants by representing conditions as spatial constraints (Fujita *et al.*, 1992). In their method, the user has to specify the parts to be placed and constraint conditions of a power plant. However, it is not easy for a user who does not have any knowledge about power plants to input technical data. In our method, a user can design a process plant intuitively by drawing 2D sketch input.

## 2. Overview of a Processing Plant

A processing plant consists of massive and interlacing parts including tanks, pipelines, processing columns, frames, and so on as shown in Fig. 1 (see also Ohyama and Ishii, 2007). In the real world, a plant layout is de-

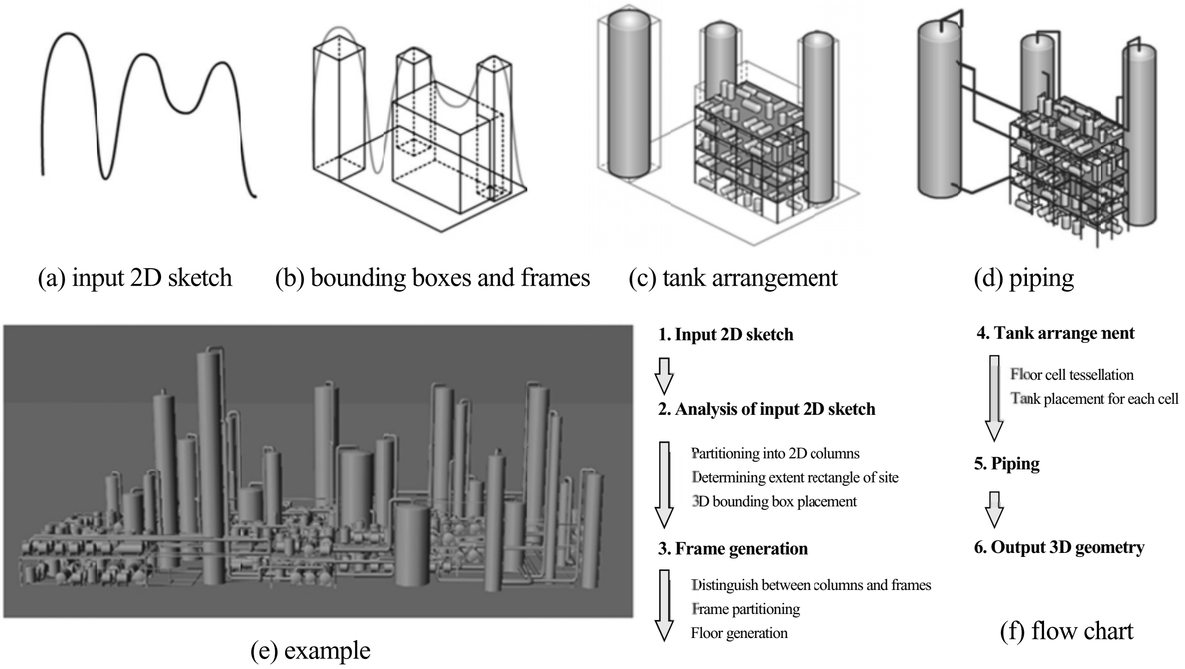
Copyright © Society for Science on Form, Japan.



#1: (a) Tower, (b) Frame, (c) Pipeline

#2: (a) Tank, (b) Frame, (c) Pipeline

Fig. 1. Example of processing plant (courtesy of http://www.photolibrary.jp/).


1. Input 2D sketch
2. Analysis of input 2D sketch
Partitioning into 2D columns
Determining extent rectangle of site
3D bounding box placement
3. Frame generation
Distinguish between columns and frames
Frame partitioning
Floor generation
4. Tank arrange nent
Floor cell tessellation
Tank placement for each cell
5. Piping
6. Output 3D geometry

(a) input 2D sketch (b) bounding boxes and frames (c) tank arrangement (d) piping

(e) example (f) flow chart

Fig. 2. Procedure overview.

signed considering the space, relations between parts, material handling, communications, utilities, and structural forms.

The main purpose of this paper is to generate the landscape of a processing plant from simple input data, so we only consider the space and structural forms of the plant, and do not care about the precise details of functionality. Therefore, our method generates only towers, pipelines, tanks, and frames. Other apparatus, such as valves, flarestacks, lights, and ladders are not considered, because these models are so small compared with towers and tanks.

## 3. Generation Method for Processing Plant

This section describes the overview of the algorithm, and then explains each process in detail.

### 3.1 Process overview

Our algorithm for generating a processing plant model consists of the following four steps, which are also illustrated in Fig. 2:

(1) Analysis of 2D sketch input;
(2) Frame generation from the bounding boxes;
(3) Tank arrangement;
(4) Piping.

The procedure first analyses 2D sketch input. Here, a 2D sketch specifies the profile of a processing plant. Then 3D bounding boxes are generated from the 2D sketch. Next, frames are generated from the bounding boxes, then tanks are arranged on each floor. Finally, each tank is connected to adjoining tanks by pipelines.

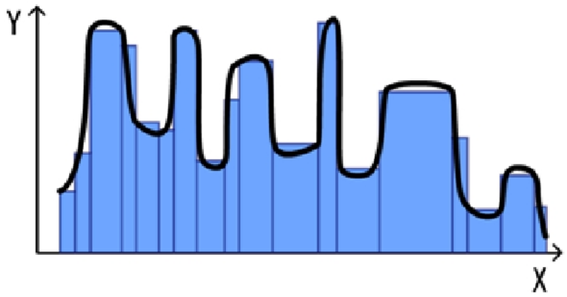
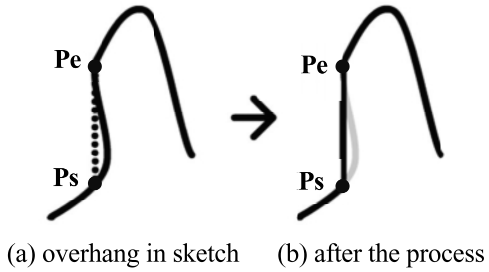
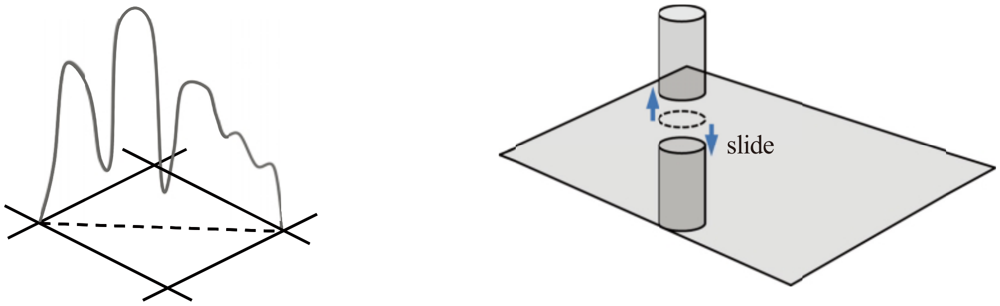


Fig. 3. Partitioning of 2D sketch.



(a) overhang in sketch (b) after the process

Fig. 4. Processing for overhang.



(a) determining the extent rectangle of the site (b) column placement

Fig. 5. Extent rectangle of the site and column placement.

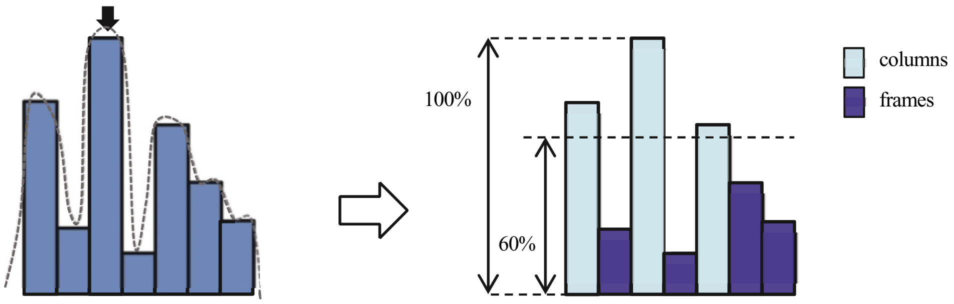


Fig. 6. Distinguishing between columns and frames.

### 3.2 Analysis of input 2D sketch

3D bounding boxes of the processing plant are generated from a 2D sketch which is drawn with a single stroke.

First, the input 2D sketch is partitioned into columns as shown in Fig. 3. The specified 2D sketch is scanned from left to right to calculate the gradient of the profile, dy/dx. A column is formed when the absolute value of the gradient is over the pre-defined threshold parameter. The steeper the gradient, the narrower the width. Here, the minimum column width is also pre-defined. We have set the threshold parameter to 0.9, and the minimum width to 15, respectively. If there is an overhang in the 2D sketch as shown in Fig. 4(a), the relevant curve is flattened as shown in Fig. 4(b) by connecting the start point, Ps, and the end point, Pe, with a straight line.

Then, the extent rectangle of the site for the processing plant is determined by placing the 2D sketch so as to fit the diagonal of the rectangle as shown in Fig. 5(a). Here, the three-dimensional viewing angle is specified by a user. Next, a bounding box for each column is placed randomly within the obtained extent rectangle of the site, by sliding its position according to the viewing direction, as shown in Fig. 5(b).

### 3.3 Frame generation

First, the highest rectangle is calculated from the partitioned 2D sketch. Then a threshold value, *Th*, is used to distinguish between columns (whose height is higher than the height of highest rectangle x *Th*) and frames (other rectangles), as shown in Fig. 6. Here, we set *Th* to 60%. A user can specify *Th*, from 30% to 60%, to change the characteristics of a processing plant.

Figure 7 illustrates the procedure for generating a frame from a bounding box. A bounding box, which is labeled as a frame (red box in Fig. 7(a)), is partitioned into lattices avoiding other boxes, as shown in Fig. 7(b). Then, each floor is generated repeatedly, as shown in Fig. 7(c). Here, the vertical, frontage and depth interval of each lattice can be changed with random numbers.

A bounding box which is labeled as a processing column is replaced with a column model in the following procedure.

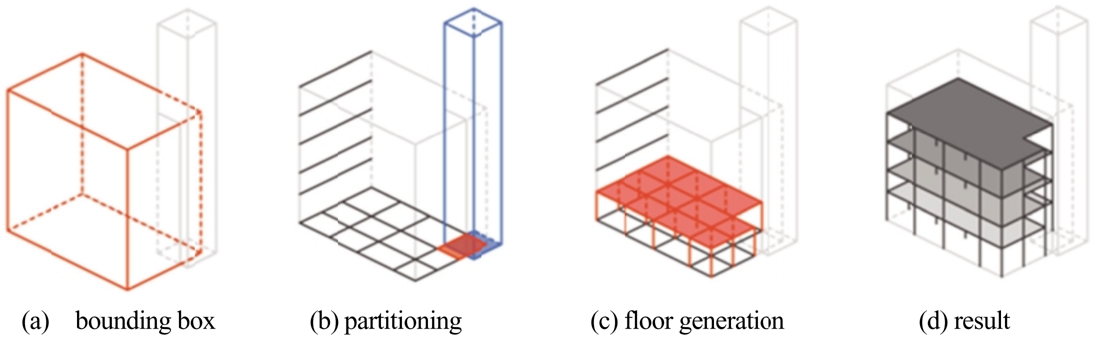
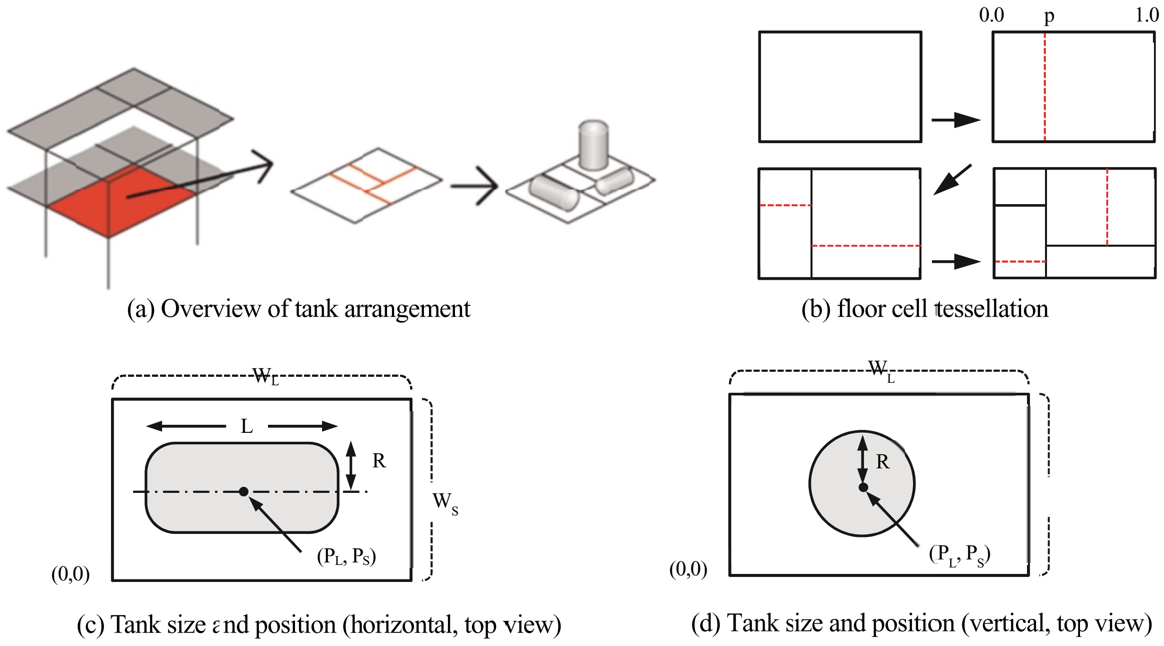

(a) bounding box (b) partitioning (c) floor generation (d) result

Fig. 7. Frame generation.



(a) Overview of tank arrangement (b) floor cell tessellation

(c) Tank size ɛnd position (horizontal, top view) (d) Tank size and position (vertical, top view)

Fig. 8. Tank arrangement.

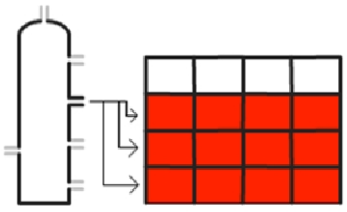


Fig. 9. Piping.

### 3.4 Tank arrangement

After frame generation, each floor is recursively tessellated into smaller rectangular cells until the size reaches a threshold value, and then each cell is filled with a tank, as shown in Fig. 8(a). Figure 8(b) shows a result of the cell tessellation process. A tessellation position, $p$, gives a position in a normalized space (0.0–1.0), and is determined by Eq. (1). Here, $Rnd$ is a uniform random number, and $C_{devrnd}$ is a parameter to control the randomness of the tessellation position.

$$p = 0.5 \pm Rnd \times C_{devrnd} \qquad (1)$$

A tessellation direction, horizontal or vertical, is also controlled by the parameter, $C_{devdir}$, which is in a range from 0.0 to 1.0. The tessellation direction is fixed as horizontal when $C_{devdir}$ is set to 0.0 or as vertical when $C_{devdir}$ is set to 1.0, or varied fifty-fifty when $C_{devdir}$ is set to 0.5.

After floor cell tessellation, a tank is placed into each cell by the following procedure.

(1) The algorithm determines stochastically whether to place a tank into a cell or not. A user can specify this probability to control density of tanks.

(2) Determine tank placement direction (lie down horizontally or stand vertically). This direction is also determined stochastically with a parameter to control the probability.

(3) Calculate the radius ($R$) and the length ($L$) of a tank. These sizes are calculated using the following equa-

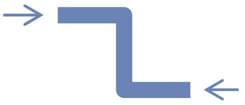
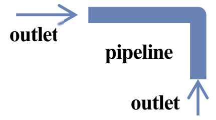
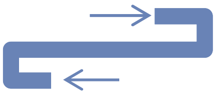
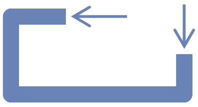
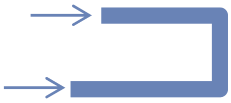
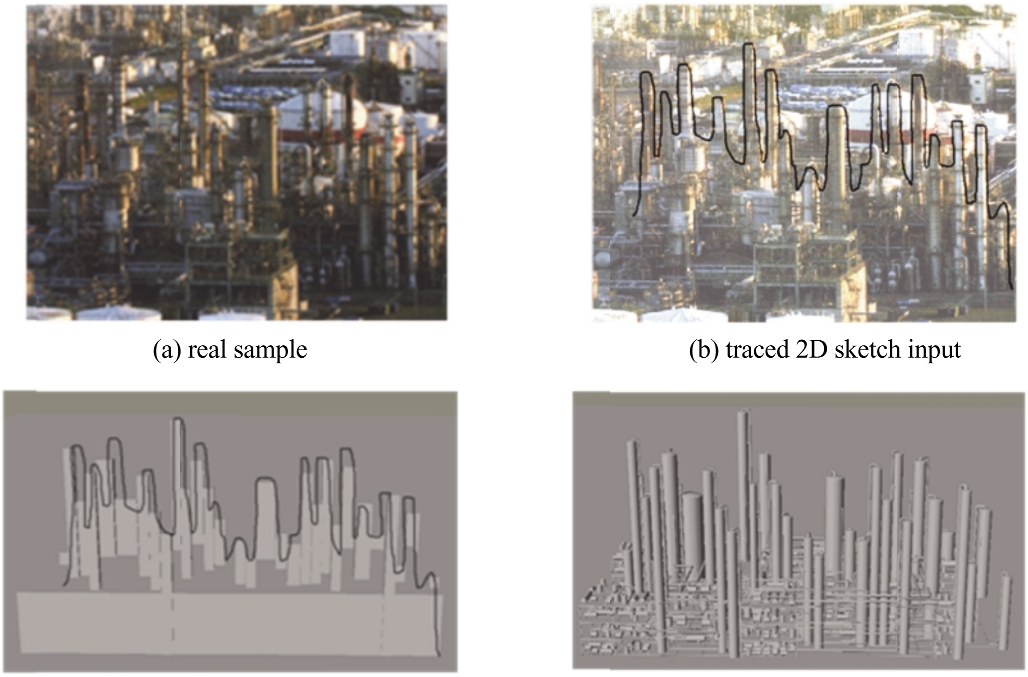
| | Opposite | Orthogonal | Same |
|---|---|---|---|
| Facing each other |  |  | |
| Other |  |  |  |

Fig. 10. Piping patterns.



(a) real sample

(b) traced 2D sketch input

(c) bounding box

(d) generated model

Fig. 11. Comparison of real sample and generated model.

tions, Eq. (2) and Eq. (3.1);

$$R = \frac{W_s}{2} \times T_{min\_radius} + (T_{max\_radius} - T_{min\_radius}) \times Rnd \qquad (2)$$

$$L = W_L \times T_{min\_length} + (T_{max\_length} - T_{min\_length}) \times Rnd \qquad (3.1)$$

Here, $W_s$ and $W_L$ are the lengths of major and minor axis, respectively. The range of radii is specified by the parameters, $T_{min\_radius}$ and $T_{max\_radius}$, and the range of length is specified by the parameters, $T_{min\_length}$ and $T_{max\_length}$. $Rnd$ is a uniform random number.
If tank placement direction is vertical, $L$ is calculated from $R$, not from $W_L$, as given by Eq. (3.2).

$$L = R \times T_{min\_rate} + (T_{max\_rate} - T_{min\_rate}) \times Rnd \qquad (3.2)$$

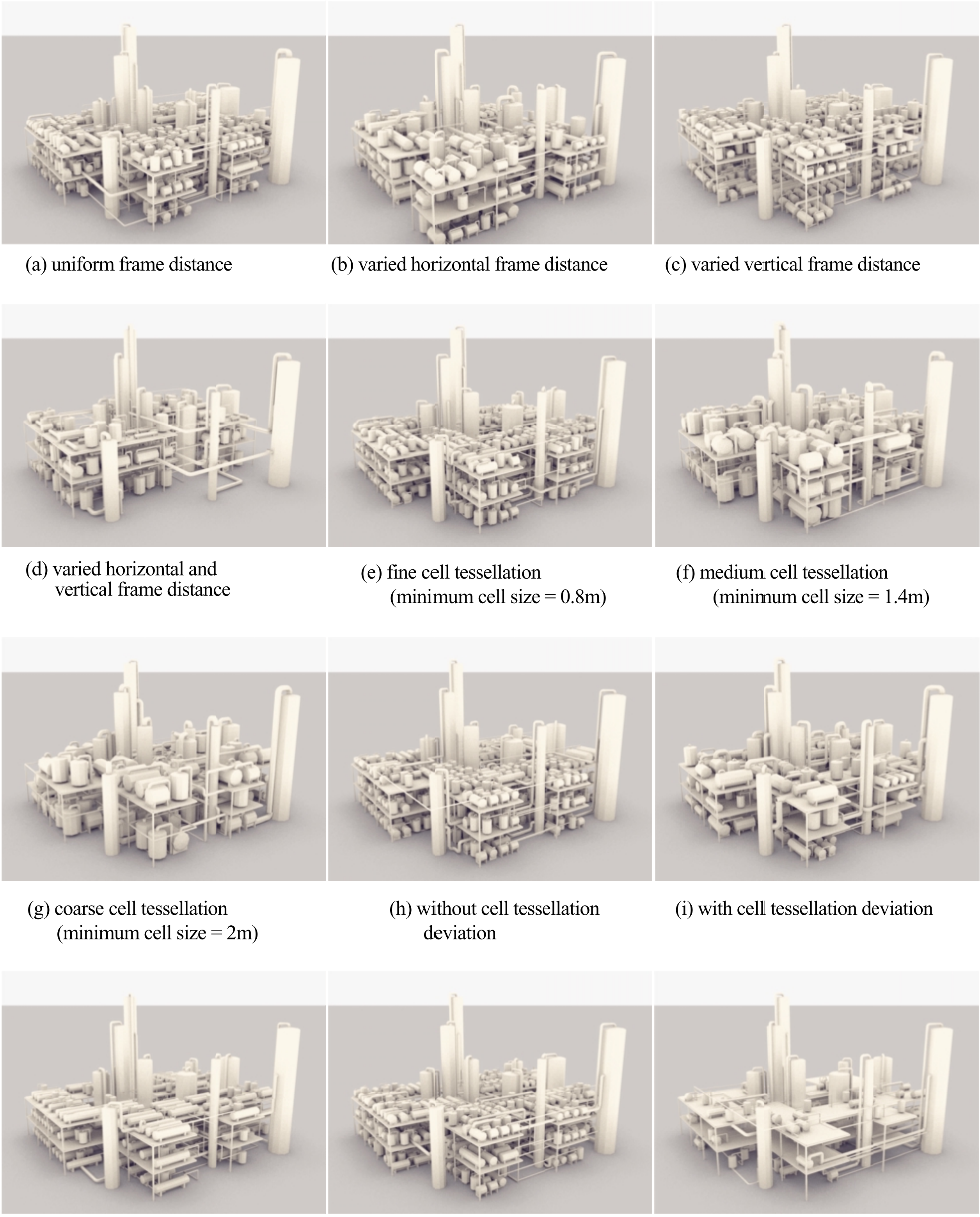
Here, $T_{min\_rate}$ and $T_{max\_rate}$ control the length of a tank. We set $T_{min\_rate} = 3.0$ and $T_{max\_rate} = 10.0$ respectively, after observing actual processing plants.

(4) The position of a tank, $(P_L, P_S)$, is determined stochastically to be contained in a cell. There is also a parameter to control the randomness.

### 3.5 Piping

Finally, each outlet of a tank is connected to one selected adjoining lower tank by a pipeline, as shown in Fig. 9. The selection probability is inversely proportional to the distance between each outlet. A piping pattern is determined by the outlet directions and their relationships, as shown in Fig. 10.

Ideally speaking, we should avoid collisions of a pipeline with other objects. However, it is a time consuming process to detect collisions, hence we omit this procedure in a practical way. Ikehira *et al.* presented an automatic design method for pipe arrangement using multi-objective genetic algorithms for optimizing the piping process (Ikehira *et al.*, 2005). However, it took about 100 iterations to arrange 15 pipes. A processing plant contains over 100 pipelines usually, so a more effective method should be developed.



(a) uniform frame distance

(b) varied horizontal frame distance

(c) varied vertical frame distance

(d) varied horizontal and vertical frame distance

(e) fine cell tessellation (minimum cell size = 0.8m)

(f) medium cell tessellation (minimum cell size = 1.4m)

(g) coarse cell tessellation (minimum cell size = 2m)

(h) without cell tessellation deviation

(i) with cell tessellation deviation

(j) tanks in same direction

(k) tanks in random directions

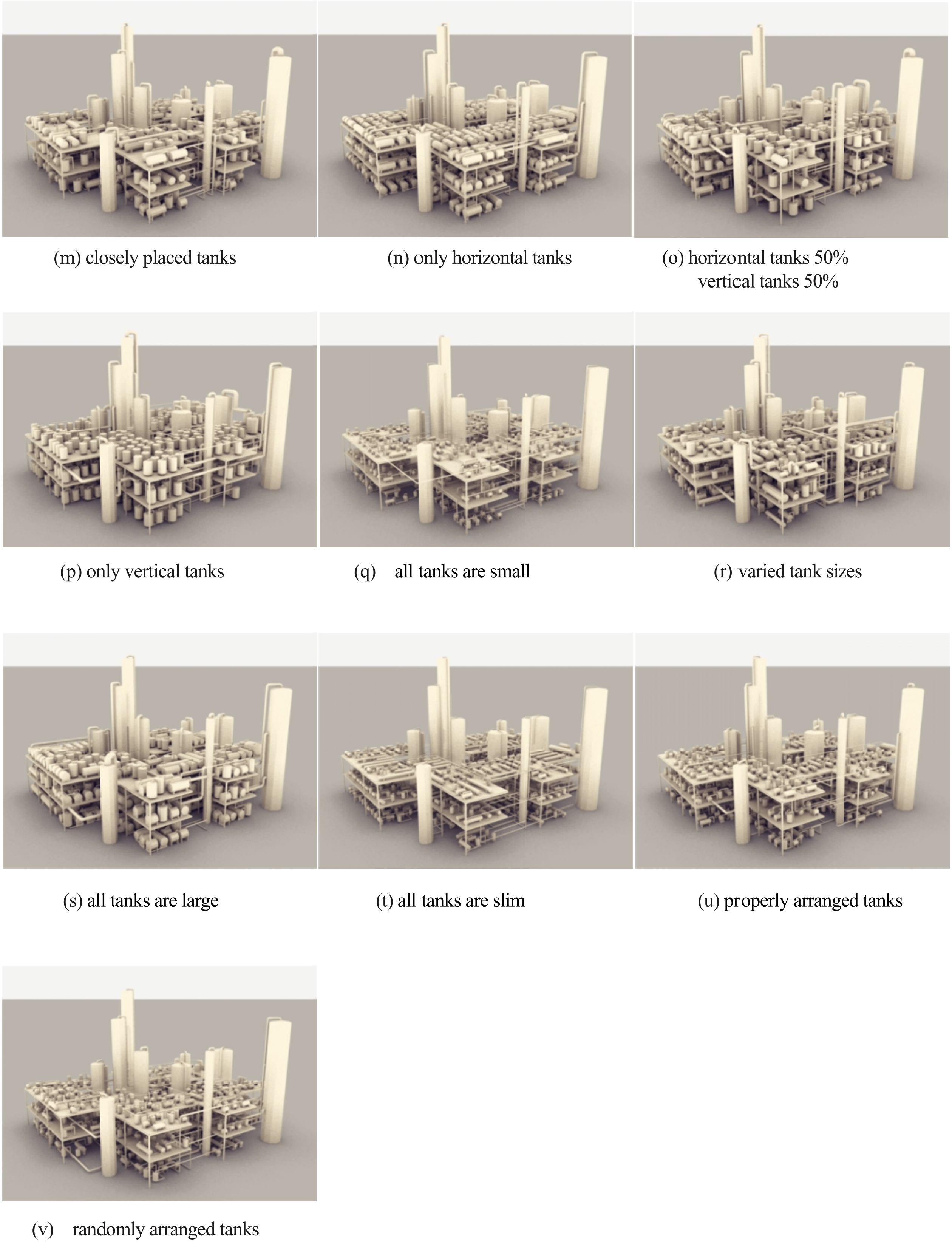
(l) few tanks

Fig. 12. Variations.

## 4. Results

Figure 11 illustrates the comparison of a real sample and a generated image. The black curve in Fig. 11(b) is the sketch input specified by tracing a real sample, Fig. 11(a). It is impossible to specify the precise depth data for each processing tower in our system, but we can observe similarity between the real sample and the generated image.

Figure 12 shows the variations controlled by some parameters, such as the number of tessellations, the tank sizes, and the proportions of tanks, from the same input sketch



(m) closely placed tanks

(n) only horizontal tanks

(o) horizontal tanks 50%
vertical tanks 50%

(p) only vertical tanks

(q) all tanks are small

(r) varied tank sizes

(s) all tanks are large

(t) all tanks are slim

(u) properly arranged tanks

(v) randomly arranged tanks

Fig. 12. (continued).

data. Each image is rendered using skylight illuminance.

Figure 13 shows the results generated by this method. Here, we show the effectiveness of parameter *Th* for generating frames, which is described in Subsec. 3.3, in Figs. 13(a-2) and (b-2). These two examples are obtained from the same sketch input, however we set *Th* to 30% for Fig. 13(a-1) and to 60% for Fig. 13(b-1), respectively. Therefore, we can change the appearance of the model dras-

Table 1. Parameter list for Fig. 12.

| Parameter \ ID | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_{rnd\ H}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.0 | 0. 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $P_{rnddist\ H}$ | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $D_{base\_H}$ | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5. 0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 |
| $D_{rnd\_V}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.0 | 0. 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $P_{rnddist\_V}$ | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $D_{base\_V}$ | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3. 0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 |
| $C_{min}$ | 1.0 | 1.0 | 1.0 | 1.0 | 0.8 | 1.4 | 2.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| $D_{devrnd}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| $C_{devdrec}$ | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 |
| $P_{tankput}$ | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1. 0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| $P_{tankvar}$ | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0. 3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |
| $T_{fillmin\_rad}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 | 0. 8 | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 |
| $T_{fillmax\_rad}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 | 0. 8 | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 |
| $T_{fillmin\_len}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 | 0. 8 | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 |
| $T_{fillmax\_len}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 | 0. 8 | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 |
| $T_{rnd\_len}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0. 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $T_{rnd\ rad}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0. 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

| Parameter \ ID | l | m | n | o | p | q | r | s | t | u | v |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_{rnd\ H}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0. 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $P_{rnddist\ H}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $D_{base\_H}$ | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5. 0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 |
| $D_{rnd\_V}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0. 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $P_{rnddist\_V}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $D_{base\_V}$ | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3. 0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 |
| $C_{min}$ | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| $D_{devrnd}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $C_{devdrec}$ | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| $P_{tankput}$ | 0.2 | 0.7 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| $P_{tankvar}$ | 0.3 | 0.3 | 0.0 | 0.5 | 1.0 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |
| $T_{fillmin\_rad}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 | 0.4 | 0.4 | 0.9 | 0.4 | 0.5 | 0.5 |
| $T_{fillmax\_rad}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 | 0.4 | 0.9 | 0.9 | 0.4 | 0.5 | 0.5 |
| $T_{fillmin\_len}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 | 0.4 | 0.4 | 0.9 | 0.7 | 0.5 | 0.5 |
| $T_{fillmax\_len}$ | 0.8 | 0.8 | 0.8 | 0.8 | 0.8 | 0.4 | 0.9 | 0.9 | 0.9 | 0.5 | 0.5 |
| $T_{rnd\_len}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| $T_{rnd\ rad}$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |

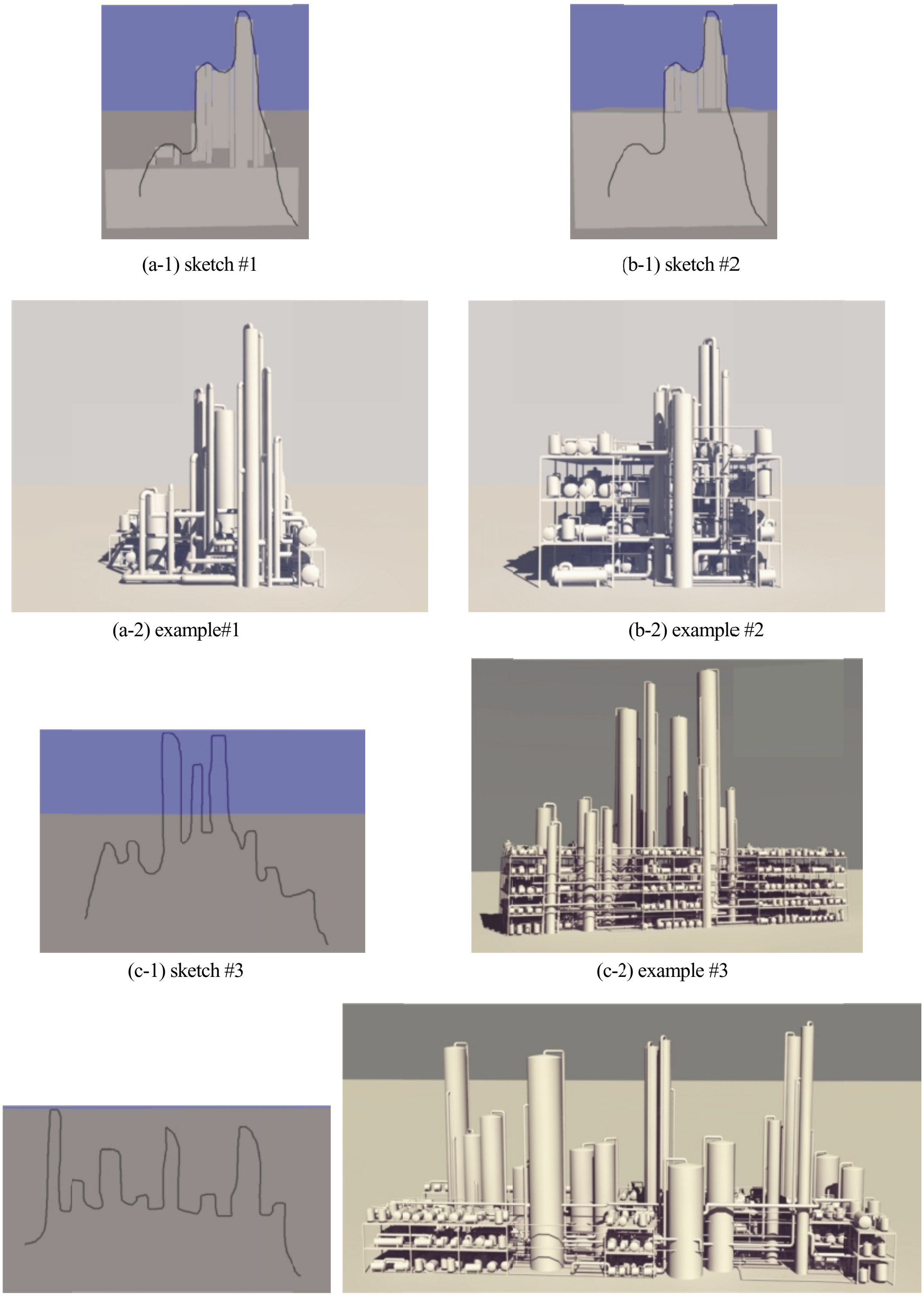
Table 2. Parameter list for Fig. 13.

| $D_{md\_H}$ | $P_{mddist\_H}$ | $D_{base\_H}$ | $D_{md\_V}$ | $P_{mddist\_V}$ | $D_{base\_V}$ |
|---|---|---|---|---|---|
| 0.0 | 0.0 | 5 | 0.0 | 0.0 | 3 |
| $C_{min}$ | $D_{devmd}$ | $C_{devdrec}$ | $P_{tankput}$ | $P_{tankvar}$ | |
| 1 | 0.0 | 0.5 | 1.0 | 0.3 | |
| $T_{fillmin\_rad}$ | $T_{fillmax\_rad}$ | $T_{fillmax\_rad}$ | $T_{fillmax\_len}$ | $T_{md\_len}$ | $T_{md\_rad}$ |
| 0.8 | 0.8 | 0.6 | 0.8 | 0.5 | 0.5 |

tically with parameter *Th*.

The processing time for generating tanks and piping depends linearly on the number of pipelines, as shown in Fig. 14. The time was measured on a PC with Intel Core2 Quad 2.83GHz, 4GB memory.

We conducted subjective evaluations for Fig. 13. Each respondent evaluated the figures using a 10-point scale, with a higher number representing higher satisfaction. Figure 15



(a-1) sketch #1

(b-1) sketch #2

(a-2) example#1

(b-2) example #2

(c-1) sketch #3

(c-2) example #3

(d-1) sketch #4

(d-2) example#4

Fig. 13. Examples.

shows the evaluation results from 22 respondents. The average scores are 6.9, 7.3, 6.6 and 7.0, respectively. This result shows that the generated examples are highly evaluated.

## 5. Conclusion

The examples show that the landscapes of processing plants are satisfactorily represented, while some detailed parts, such as valves, steps, and branching pipelines, are not generated.

The method does not consider collisions of pipelines and other objects, either. Therefore, some pipelines penetrate tanks and processing columns. In general, a collision-detection process is a time-consuming one; therefore we have to design an effective way to avoid collision in the piping process.

From a practical application standpoint, it is necessary

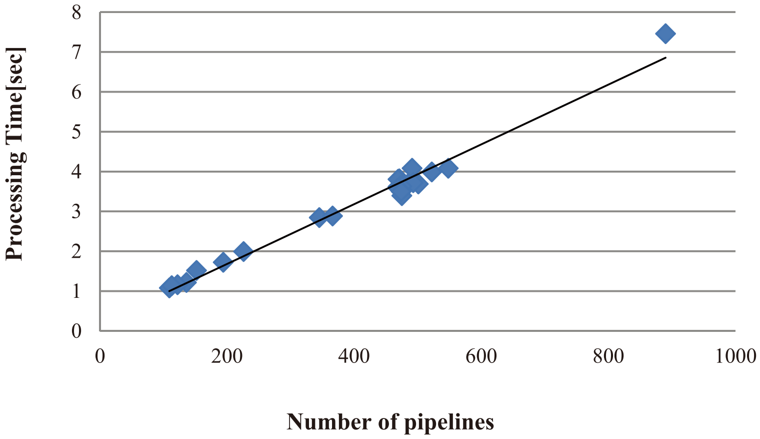


Fig. 14. Processing time.

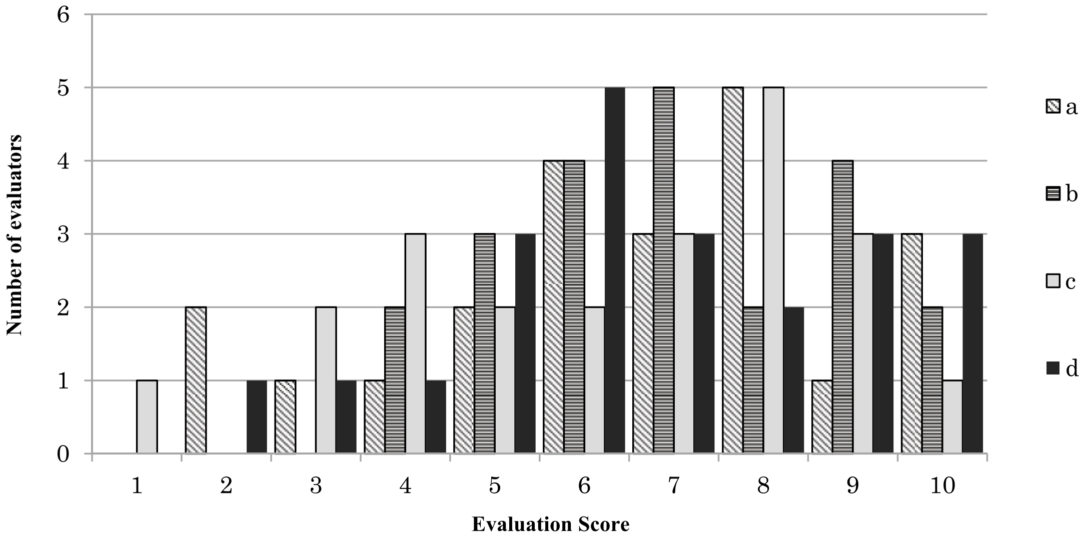


Fig. 15. Subjective evaluations for Fig. 13.

to edit a generated model interactively. For example, the operator will want to change the piping, tank layout, and tower placement. In our current implementation, we do not support interactive editing for a generated model. It is also necessary to texture massive models automatically, however this is outside our research scope.

We plan to address these issues in future research.

## References

Chen, G., Esch, G., Wonka, P., Müller, P. and Zhang, E. (2008) Interactive procedural street modeling, in *Proc. of ACM SIGGRAPH 2008*, **27**, 1–10.

Frischer, B. *et al.* (2008) Rome Reborn, *SIGGRAPH2008 New Tech Demos*.

Fujita, K., Akagi, S. and Doi, H. (1992) Automated acquisition of constraints in plant layout design problems, *J. Japan Soc. Mech. Eng.*, **58**, No. 555, 3449–3454 (in Japanese).

Ikehira, R., Kimura, G. and Kajiwara, H. (2005) Automatic design for pipe arrangement using multi-objective genetic algorithms, in *12th International Conference on Computer Applications in Shipbuilding (ICCAS)*, **2**, 97–110.

Merrell, P. (2007) Example-based model synthesis, in *I3D '07: Proc. of the 2007 Symposium on Interactive 3D Graphics and Games*, pp. 105–112.

Merrell, P. and Manocha, D. (2009) Constraint-based model synthesis, in *SPM '09: 2009 SIAM/ACM Joint Conf. on Geometric and Physical Modeling*, pp. 101–111.

Müller, P., Wonka, P., Haegler, S., Ulmer, A. and Van Gool, L. (2006) Procedural modeling of buildings, in *Proc. of ACM SIGGRAPH 2006/ACM TOG*, **25**, pp. 614–623.

Müller, P., Zeng, G., Wonka, P. and Van Gool, L. (2007) Image-based procedural modeling of facades, in *Proc. of ACM SIGGRAPH 2007*, **26**, #85.

Ohyama, K. and Ishii, T. (2007) *Kojo Moe*, Tokyo-Shoseki Co. Ltd. (in Japanese).

Parish, Y. I. H. and Müller, P. (2001) Procedural modeling of cities, in *Proc. of ACM SIGGRAPH 2001*, pp. 301–308.

Procedural Inc. (2008) CityEngine, http://www.procedural.com/

Stiny, G. (1980) Introduction to shape and shape grammars, *Environment and Planning B: Planning and Design*, **7**, 3 (May), 343–351.

Wonka, P., Wimmer, M., Sillion, F. and Ribarsky, W. (2003) Instant architecture, *ACM Transaction on Graphics*, **22**, 4 (July), 669–677.