Flower Factory: A Component-based Approach for Rapid Flower Modeling

Siyuan Wang*
State Key Lab of VR
Technology & Systems
Beihang University

✉ Junjun Pan†
State Key Lab of VR
Technology & Systems
Beihang University
Peng Cheng Lab

Junxuan Bai‡
State Key Lab of VR
Technology & Systems
Beihang University
Peng Cheng Lab

Jinglei Wang§
Bytedance, Inc.

Abstract

The rapid 3D objects modeling provides an effective way to enrich digital content, which is one of the essential tasks in VR/AR research. Flowers are frequently utilized in real-time applications, such as video games and VR/AR scenes. Technically, a realistic flower generation using the existing 3D modeling software is complicated and time-consuming for designers. Moreover, it is difficult to create imaginary and surreal flowers, which might be more interesting and attractive for the artists and game players. In this paper, we propose a component-based framework for rapid flower modeling, called Flower Factory. The flowers are assembled by different components, e.g., petals, stamens, receptacles and leaves. The shape of these components are created using simple primitives such as points and splines. After the shape of models are determined, the textures are synthesized automatically based on a predefine mask, according to a number of rules from real flowers. The whole modeling process can be controlled by several parameters, which describe the physical attributes of the flowers. Our technique is capable of producing a variety of flowers rapidly. Even novices without any modeling skills are able to control and model the 3D flowers. Furthermore, the developed system will be integrated in a lightweight application of smartphone due to its low computational cost.

Keywords: Procedural modeling, geometric modeling, 3D flower, component, texture synthesis.

Index Terms: Computing methodologies—Computer graphics—Shape modeling—Parametric curve and surface models; Computing methodologies—Computer graphics—Image manipulation—Texturing

1 Introduction

The rapid modeling of 3D geometric objects is very important for enriching digital content, which is an essential task in VR/AR research and development. Flowers are frequently utilized in CG applications, such as video games, film production and VR/AR scenes. On one hand, due to the complex structure of the flower, it is quite tactical and time-consuming for designers to generate a realistic flower using the 3D modeling software. In the traditional modeling pipeline, designers must depict the shapes and produce the textures for each component in a flower. It usually takes more than 20 minutes to model a 3D flower for a skilled designer. On the other hand, it is quite challenging to create imaginary and surreal flowers through traditional pipeline, which might be more interesting and attractive for artists and audience. Meanwhile, with the popularity of smartphones and the growing need of mobile entertainment, a lightweight modeling software for flowers is in great demand. Although there are some semi-automatic modeling methods for flowers [10, 33–35], they require users to provide real photos of the flower or complex structural information of the flower. Due to the high computational cost and complicated interactions, these methods can hardly be integrated in lightweight applications, such as XR in smartphones.

In this paper, we propose a component-based approach for rapid flower modeling that is named as Flower Factory. It is able to overcome the above limitations and generate realistic 3D flowers at the mobile platform. In our framework, the flowers are assembled by different components, e.g., petals, stamens, receptacles and leaves. The shapes of the components are created using geometric primitives such as points, splines. After the shapes are determined, the textures are synthesized automatically based on a predefine mask. The whole modeling process can be controlled by 84 parameters (Appendix), which describe the physical attributes of the flowers. Designers can produce a variety of flowers rapidly using our method. Even novices without any modeling experience are able to create 3D flowers. Finally, due to its simple interactions and low computational cost, our technique can be integrated in lightweight applications for smartphones.

To sum up, the technical contributions of our approach can be summarized as follows:

- We propose a comprehensive component-based framework for rapid flower modeling, which can be controlled by a set of parameters. The flower components are all generated by simple geometric primitives, including petals, stamens, receptacles and leaves. It greatly reduces the computational cost which ensures the framework can be integrated into mobile platform. With an intuitive and easy-to-use interface, novices can quickly create 3D flowers with the style on their own.
- We assemble the components in different ways to create a complete flower model. The geometric information and the topological structure of the model are processed separately. Several straightforward but effective strategies are presented to handle the collisions of petals in floral components during assembling process and blooming animation.
- Considering the natural characteristics of the textures of real petals, we design a number of rules and provide a predefine mask to adjust the color distribution on the petal surface. The textures of petals can be controlled and synthesized with different attributes automatically.

The overview of our system is shown in Fig. 1. The input of the system is the parameters that are provide by the user or system. The geometry and texture of the components are generated according to the input, and then the entire flower is assembled using these components.

The remainder of the paper is organized as follows: After briefly discussion about the related work in Section 2, we will introduce the geometry generation of flowers including petals, stamens, receptacles and leaves in Section 3 and texture synthesis in Section 4. Section 5 documents the experimental results and comparisons. Finally, Section 6 concludes the paper and points out the future work.

*e-mail: siyuanwang121@buaa.edu.cn
†e-mail: pan_junjun@buaa.edu.cn (Corresponding author)
‡e-mail: baijx6@163.com
§e-mail: wangjinglei@bytedance.com

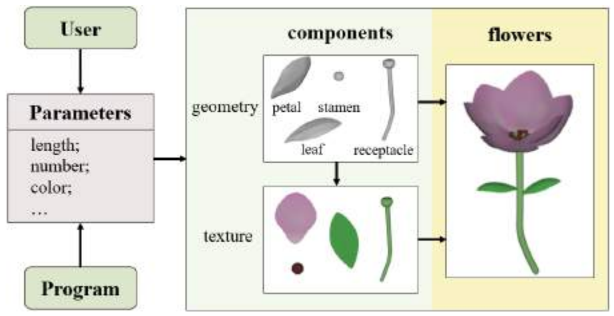


Figure 1: System overview.

2 Related Work

In this section, we first investigate the procedural modeling approaches for plants. Then we review image-based or video-based modeling techniques. These techniques have become a hot research topic in recent years. Finally, we inspect the methods of texture generation for plants.

Procedural modeling for plants. Procedural modeling is often employed to generate 3D plants based on specific rules. The most famous one is the L-system, which was proposed by Lindenmayer [13] and introduced to the computer graphics community by Prusinkiewicz and Lindenmayer [24]. Subsequently, plenty of modeling techniques based on the L-system were proposed [3,12,22,23,25]. Boudon et al. [3] utilized the L-system to model potted plants, and Qin et al. [25] proposed a flower modeling method based on the L-system and Bezier curve. Furthermore, another type of rule-based modeling method was proposed to make the designing process more convenient [2,5–9,17,18,20]. Owens et al. [18] presented a suite of biologically-motivated algorithms for inflorescence modeling and its animation process. Ijiri et al. [9] provided a graphical interface to create a flower based on the sketch of botanical structures.

Image/video-based modeling for plants. With the popularity of digital cameras and 3D scanning equipments in recent years, a great number of data-driven methods were proposed for trees or flowers [10,16,26,31,33–35]. In order to faithfully reproduce the plant, these methods reconstructed the plant model by 3D point clouds or images. Yan et al. [33] presented a semi-automatic framework for reconstructing flower model from a single photograph. A revolution surface is used to model the structure of flowers, and the shapes of petals are created based on real photos. Zhang et al. [34] obtained the flower information from 3D point clouds at a single view, and then it created realistic flower model based on the information. Ijiri et al. [10] utilized an X-ray computed tomography (CT) system to scan the real flowers and completed the modeling with user interactions. Zheng et al. [35] captured the visible parts of a blooming flower into a point-cloud sequence, then it reconstructed the geometry of the flower and the deformation over time. Most of the above methods require users to provide real photos or scanning data and interaction parameters to reconstruct the flowers. They can hardly produce imaginary or surreal flowers, and they usually take a long time in the modeling process. As a consequence, it is difficult to integrate these techniques into a lightweight application, such as VR/AR in smartphones.

Texture synthesis for plants. Automatic generation of leaf textures is investigated by some researchers [1,11,15,21,27,28]. Given the producing rules and the leaves' age, Peyrat et al. [21] synthesized the changes of leaf in color and texture. Rodkaew et al. [27] presented a particle transport system, which generates leaf texture and color by randomly scattering particles within a given leaf shape. Alsweis et al. [1] proposed a procedural technique for the simulation of leaf contour growth and venation development based on biological principles. The leaf contour is extracted from the given photo, and the pigment is diffused in the 2D space according to the leaf shape. In our paper, an automatic generation method is proposed for the petal's texture based on the shape of petal.

3 Geometry Generation

The geometric model of flowers can be separated into four components: petals, stamens, receptacles and leaves. We first generate each component respectively, then we combine these components to assemble a complete flower model. The generation process is fully controlled by parameters, which describe the physical attributes of the flowers. We provide a simple but intuitive interface for users to adjust parameters. After the model generation, users can still edit the 3D flowers manually according to their needs.

3.1 Floral Components

In our modeling framework, the floral components consist of petals, leaves, receptacles and stamens. All the components can be controlled by parameters.

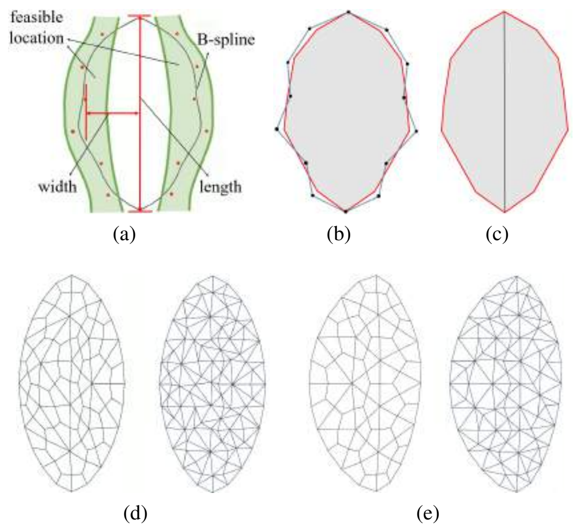


Figure 2: Generation of 2D model. (a) The green region is the feasible location of the control points (red dots). The control points are chosen randomly. (b) The blue contour is created by connecting the control points directly and the red contour is generated by sampling on the B-spline. (c) The black line is the central vein of the petal. (d) The subdivision result and the triangulation result of the symmetric petal. (e) The subdivision result and the triangulation result of the asymmetric petal.

Petals and leaves are created using two boundary curves on 2D planes, and the boundary curves are generated by control points. To keep the natural characteristics, the region for the boundary curves is defined and the control points must locate in this region. Users are able to decide the number of the control points. As shown in Fig. 2a, the length and the width of the model are also provided by the user, and the feasible location of the control points is adjusted based on the width. There might exist unnatural sharp and rough shape if the edge is created by connecting the control points directly (Fig. 2b). To remove the artifact, we generate smooth boundary curves by sampling on B-spline which is defined by the control points. As shown in Fig. 2c, the central vein of the model is connected from the head to tail of the contour. We also provide a parameter to control the symmetry of the model (Fig. 2d and Fig. 2e). To obtain a smooth geometry, the model is subdivided using [4] and triangulated using [30].

Serrate leaves often appear in nature, so we add sawtooth to the boundaries of leaves. Various patterns are defined for sawtooth. As demonstrated in Fig. 3, the pattern is defined in a unit region on 2D plane. After the user selects a pattern, our program applies the pattern to the boundaries of the leaves. Also, the user can define the number and the size of the sawtooth. The positions of the sawtooth are determined based on the number that the sawtooth are distributed evenly on the boundaries. As depicted in Fig. 3, the results controlled by different parameters are presented.

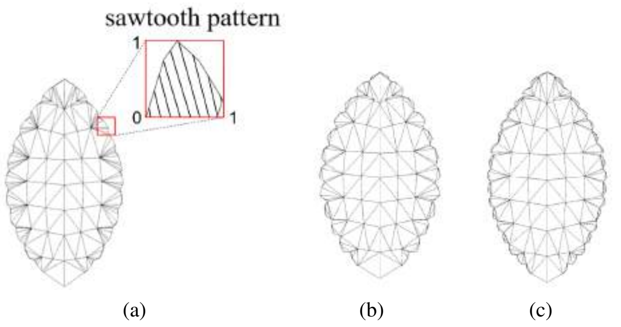


Figure 3: The serrate edges of the leaf. (a) A leaf model with 11 small sawtooth. (b) A leaf model with 11 large sawtooth. (c) A leaf model with 17 smaller sawtooth.

Since the models are created on 2D plane, we employ linear blending skinning (LBS) [14] to produce the 3D geometry.

We divide all the points into three sets: $P = \{\mathbf{p} \mid \mathbf{p} \text{ is in the plane model}\}$, $V = \{\mathbf{p} \mid \mathbf{p} \text{ is on the central vein}\}$, and $S = P - V$. Then the rigging weights for points in S are calculated using Algorithm 1. The weight is defined according to the Euclidean distance between the points in S and the points in V. Furthermore, as shown in Fig. 6, a parameter ω is defined to control the curvature of the model. To change the shape in another way, we also make the central vein to fit a 3D curve which is demonstrated in Fig. 4. As a result, the user is capable of editing the 3D shape of the model.

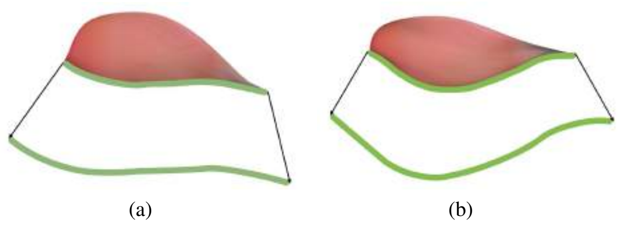


Figure 4: The deformation of shape based on the 3D curve.

The flower stem is treated as a receptacle component. It shares a similar structure with the stamen, which includes an ellipsoid and a tube. The tube is controlled by a parametric curve which consists of a diameter and a length for the tube. The ellipsoid is controlled by three inner diameters in x-y-z directions. Finally, a random function [19] is utilized to the vertices to add fluctuation, which can make the geometry more real.

3.2 Components Assembly

A complete flower model can be assembled after each component is created. As shown in Fig. 1, the leaves are distributed around the stem. User can specify the number of leaf layers, the interval height between two layers, the number of leaves in each layer and the rotation angle α of the leaves, which are illustrated in the Fig. 5. The size of the leaves also can be adjusted to improve the visual effect.

Algorithm 1 Calculate Rigging Weights

```
Input: $P$, $S$, $V$ and $\omega$
Output: $W$ – the weights for $S$
 1: $W = \Phi$
 2: for each $\mathbf{p}_s \in S$ do
 3:     $Distance(\mathbf{p}_s) \leftarrow \Phi$
 4:     for each $\mathbf{p}_v \in V$ do
 5:         $Distance(\mathbf{p}_s, \mathbf{p}_v) \leftarrow \text{Dist}(\mathbf{p}_s, \mathbf{p}_v)$
 6:     end for
 7:     sort $Distance(\mathbf{p}_s)$ in ascending order;
 8:     $num \leftarrow 3$;
 9:     $sum \leftarrow \text{SUM}(Distance(\mathbf{p}_s), num)$
10:     $W(\mathbf{p}_s) = \Phi$;
11:     for $i = 0 \rightarrow num - 1$ do
12:         $W(\mathbf{p}_s, i) \leftarrow Distance(\mathbf{p}_s, i)/sum * \omega$;
13:     end for
14: end for
15:
16: function $\text{SUM}(Array, number)$
17:     $result \leftarrow 0$;
18:     for $i = 0 \rightarrow number - 1$ do
19:         $result \leftarrow result + Array[i]$;
20:     end for
21:     return $result$
22: end function
```

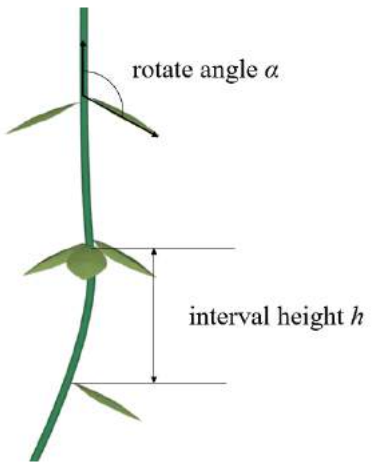


Figure 5: Assembling the leaves. The flower has 3 layers with 6 leaves. The parameter h controls the distance between different layers, and α controls the rotation.

Given the number of the stamens, the stamens are distributed randomly on the receptacle. To generate the stamens, the receptacle is divided into small cells at first, then the stamens are located into these cells. The width of the cell is twice of the radius of the stamen, and all the cells must be in the receptacle. The center of the central cell and the center of the receptacle are coincident.

Starting from the central cell whose index is 0, the index increases along a spiral curve (in yellow). When the user decide the number of stamens n, the cell index of the i-th stamen $s_i \in [0, m]$ is created by

$$s_i = \lfloor \frac{i \times m}{n} \rfloor, \tag{1}$$

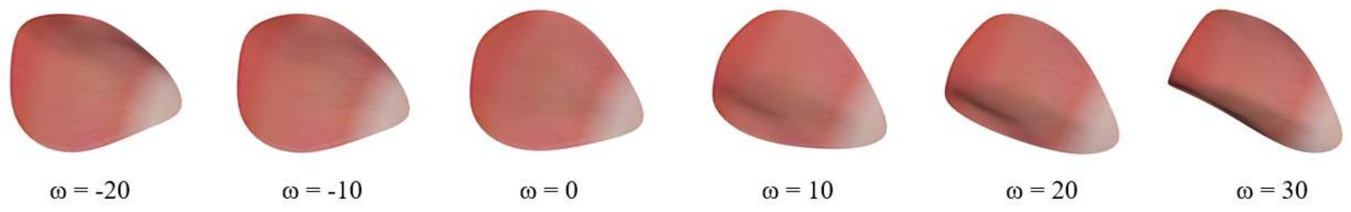


Figure 6: The models in different curvatures. When $\omega = 0$, the petal is a plane. When $\omega > 0$, the petals curl up. When $\omega < 0$, the petals curl down.

where m is the maximum index of the cells and $i \in [0, n)$. The position of the i-th stamen is the same with the s_ith cell (Fig. 7). The rotation angle of each stamen is set randomly, which looks more natural. If n is larger than m, we set $n = m$.

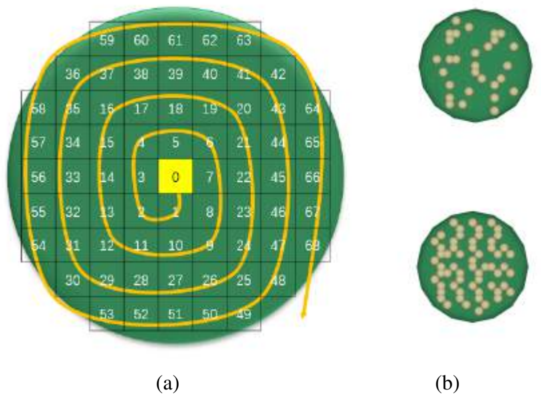


Figure 7: The distribution of the stamens. (a) The cells and the indices in the receptacle. (b) The result when $n = 30$ (top) and $n = 60$ (bottom).

The petals are distributed around the receptacle. User can define the number of petal layers, the number of petals and the sizes of petals. A rotation angle is computed to ensure the petals are evenly distributed. The opening state is dominated by an input angle β, which depicts the angle between the petals and the stem (Fig. 9).

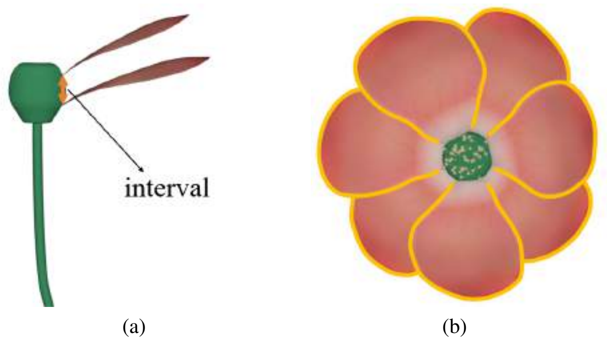


Figure 8: The collision handling for petals.

Collisions and overlaps may appear between different petal layers. To handle the problem, a straightforward but effective strategy is designed. As illustrated in Fig. 8a, we set a small interval between adjacent layers. In the same layer, there is also an interval between adjacent petals. These two intervals are chosen empirically, but users can change the values in the system to obtain different models. We define a maximum circle to define the flower bud. As depicted in Fig. 10 The shape is controlled by the radius and the position of the maximum circle. A closing angle is provided to adjust the entire shape when the petals are closed simultaneously. Also, users can change the closing angle to edit the closed shape.

To generate more realistic flowers, we add random factors to some procedures. For example, the size of the petal is set as 0.8, and the random factor is set as 0.1, then the size of petals is generated within $[0.7, 0.9]$. More experimental results can be found in Section 5.

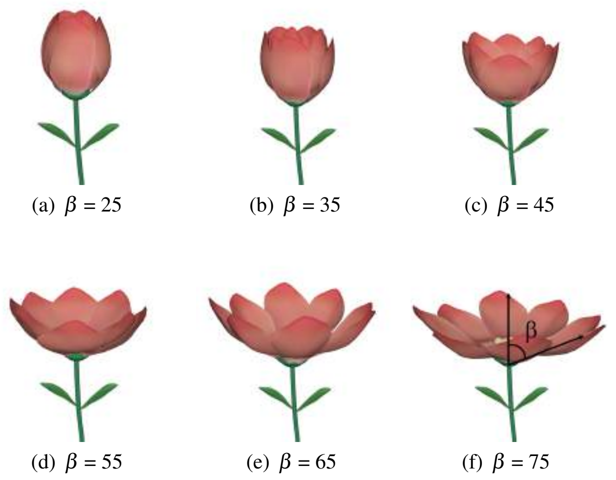


Figure 9: Different opening angle β are applied in the flower model.

3.3 Blooming Animation

The blooming animation can be generated using linear interpolation of the corresponding points of the closed flower P_a and the opening flower P_b. However, we find that the "blooming" is not obvious if we only calculate the interpolation of the vertices positions. Therefore, we also calculate the interpolation of the closed angle β_a and the opening angle β_b. We utilize unit vectors to represent the rotation angles. We introduce a unit vector $\mathbf{d}_a$ to denote β_a that $\mathbf{d}_a$ points to the top of the petal from the root of the petal. Also, a unit vector $\mathbf{d}_b$ is defined to represent the opening angle β_b. After the number of interpolation frames m and the speed are determined, the interpolated frames can be calculated. We use $\tau_t = \{P_t, \beta_t\}$ to describe the state of the t-th frame, where $P_t = \{\mathbf{p}_i^t | i = 1, ..., N\}$ and β_t represent the positions of the vertices and the rotation angle. N is the number of the petal vertices. In practice, we use a unit vector $\mathbf{d}_t$ to represent β_t. The position and the rotation of the central vein at the t-th frame can be calculated as:

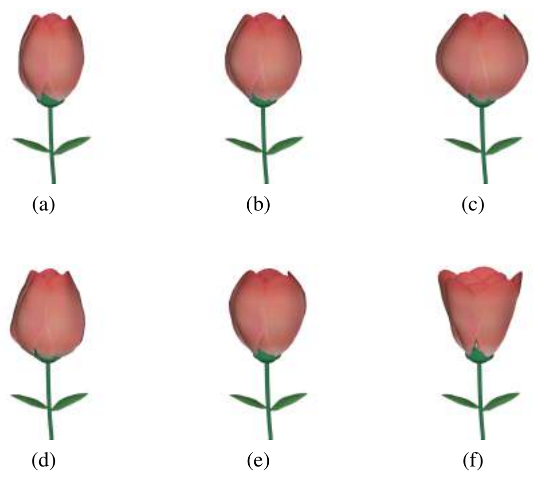


Figure 10: The closed states of the flower bud. The maximum circle controls the shape. Different radius generates different shapes in (a), (b) and (c). Different position also generates different shapes in (d), (e) and (f).

$$\begin{aligned}
\mathbf{p}_i^t &= (1-x_t)\times \mathbf{p}_i^a + x_t \times \mathbf{p}_i^b, && (2)\\
\mathbf{d}_t &= \frac{\sin(x_t\times\Omega)}{\sin\Omega}\times \mathbf{d}_a + \frac{\sin[(1-x_t)\times\Omega]}{\sin\Omega}\times \mathbf{d}_b, && (3)\\
\Omega &= \arccos(\mathbf{d}_a\cdot\mathbf{d}_b), && (4)
\end{aligned}$$

where Ω is the angle between the two directions $\mathbf{d}_a$ and $\mathbf{d}_b$. x_t controls the interpolation state based on the frame index and the speed, and $x_t \in [0,1]$, bounded by $x_0 = 0$ and $x_m = 1$. We provide a curve to control the blooming speed, as shown in Fig. 11, and the speed of frames $\mathbf{v} = \{v_1, v_2, ..., v_m\}$ can be sampled on the curve. The x_t can be calculated by

$$x_t = \frac{\sum_{j=1}^{t} v_j}{\sum_{j=1}^{m} v_j}. \quad (5)$$

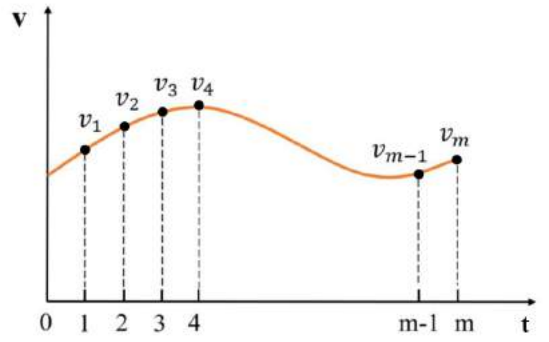


Figure 11: The speed curve to control blooming animation.

4 Textures Generation

Users can define the colors for each component, then our system automatically produce the corresponding texture based on the geometry. We propose a heuristic method to generate the texture. The textures of stamens and the receptacle are created using the input colors. Then the system adds a disturbance to make the stamens and receptacle more natural.

4.1 Textures for Petals

Through observation of the real petals, we define a set of rules for the texture generation. Generally, the color of petals is not evenly distributed, and there are some streaks on the petals. To create the textures, we define a mask for the petal component. In the mask, the pixel saves the density of the color for the vertex on the petal. Each pixel is assigned with a density parameter $\tau \in [0,1]$. We utilize a heuristic method to simulation the diffusion of the pigments on the petal. The generation rules are defined as follows.

- The bottom vertex $\mathbf{p}_s = (x_s, y_s, z_s)$ and the top vertex $\mathbf{p}_e = (x_e, y_e, z_e)$ of the petal are selected.
- We employ the backtracking method to find the shortest path from $\mathbf{p}_s$ to $\mathbf{p}_e$ on the geometry. Most paths are pruned because the results exceed the length of the shortest path. The final path is shown in Fig. 12b.
- The densities τ_s and τ_e are specified for $\mathbf{p}_s$ and $\mathbf{p}_e$.
- The density τ_c for the vertex $\mathbf{p}_c$ is calculated based on the distances to $\mathbf{p}_s$ and $\mathbf{p}_e$:

$$\begin{aligned}
\tau_c &= \tau_s + \frac{dist(\mathbf{p}_s,\mathbf{p}_c)}{dist(\mathbf{p}_s,\mathbf{p}_e)} * (\tau_e - \tau_s), && (6)\\
dist(\mathbf{p}_s,\mathbf{p}_e) &= \sqrt{(x_e-x_s)^2 + (z_e-z_s)^2}, && (7)
\end{aligned}$$

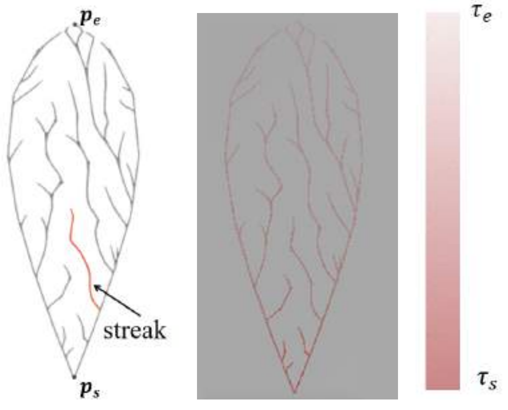


Figure 12: The formation of the petal streaks. (a) The final path. (b) The density decreases when $\mathbf{p}_c$ is far from $\mathbf{p}_s$. (c) The density map.

In general, the colors at the edge and the root are different from the center. Therefore, we set some parameters to control the density and the range as shown in Fig. 13. We also add some randomness to the mask to make the final texture more realistic.

4.2 Texture for Leaves

The texture generation for the leaves is similar to the petals, which is created according to the specified rules. However, the leaves usually have regular veins. We specify the pattern of leaf veins, and users can set the number of branches and the width of veins, as illustrated in the Fig. 14. Also, we add some noises to the output textures.

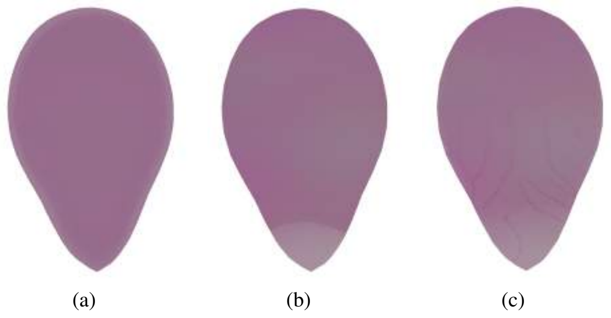


Figure 13: The texture of the petal. (a) The color at the edge is lighter than the center. (b) The color at the root is lighter than the other parts. (c) The final texture.

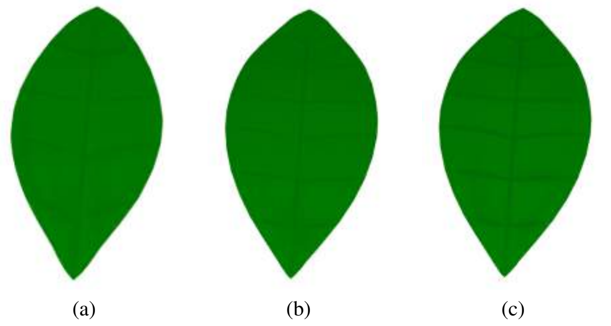


Figure 14: (a) The leaf veins with five narrow branches on each side. (b) The leaf veins with six narrow branches on each side. (c) The leaf veins with six broad branches on each side.

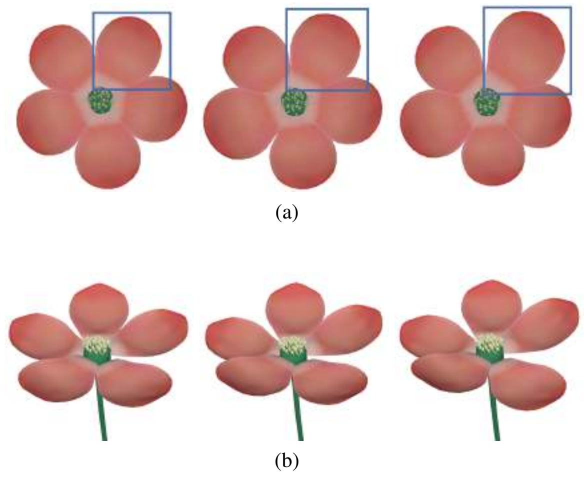


Figure 15: Different flower models based on the random factors. The left one is the neutral flower. (a) Different petal size. (b) Different opening angle β.

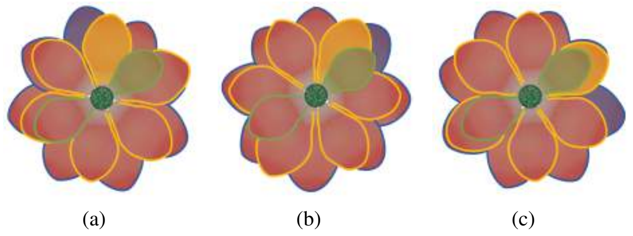


Figure 16: To avoid the overlaps, the petals in different layers are rotated. The first, second and third layers are marked in green, yellow and blue.

5 Experiments and Validation

5.1 The Effectiveness of the Parameters

To adjust the parameters, we designed a simple but intuitive user interface that is easy-to-use for novice users. Our system provides 84 parameters to generate 3D flowers in different styles. In most cases, users only need to adjust part of the parameters to create a new model, since some parameters are related to other parameters. In this section, we present experimental results for several important parameters.

Different geometry. By modifying the parameters, different shapes can be produced, including the components in different shapes, the quantity of the components and the assembly patterns. Fig. 15 lists three flower models in different random factors. These models are different in petal size and opening angle β. For the leaves, the parameters are the same.

The petals in different layers may overlap. As shown in Fig. 16a, the first layer (in green) is almost on top of the second layer (in yellow). To avoid the overlaps, we change the distributions for different layers. As shown in Fig. 16b, and Fig. 16c, the lower petals are rotated to avoid the upper petals.

As shown in Fig. 17, our system is capable of generating multiple flowers in a scene. After producing various flowers, we put them together to obtain a blossom cluster. It takes 20 minutes to produce the blossom cluster, including the component generation, the texture generation and the layout. To reduce the modeling time, each parameter is defined in a reasonable range initially so that we can produce the flowers with little adjustment. It makes our system suitable for generating large gardens in VR/AR environment. Moreover, it is efficient and convenient for 3D designers.

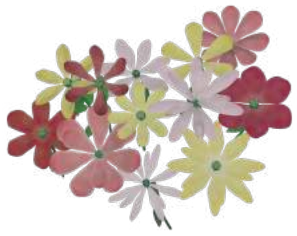
Different texture. As shown in Fig. 18, we can obtain various textures for the flowers by modifying the parameters. After the user specifies the basic color, the system calculates the densities for each vertex automatically, then it generates the textures for the petals. Moreover, the user is able to change the parameters at any time and get real-time feedback. According to our survey, designers usually need 1–2 hours to draw a texture for the flowers or edit the image to create a texture. In our system, it only takes a few seconds to produce the textures for the entire flower. Also, designers can export the textures for further modification.

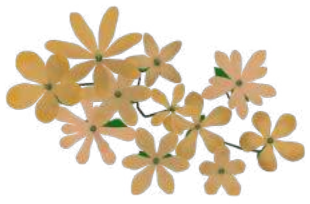
Blooming animation. As shown in the Fig. 19, we present a 30-frame blooming animation. Fig. 19a contains all the states. The complete animation can be seen in the supplementary video. Although the animation is implemented using linear interpolation, the visual result looks good. However, real blooming process is much more complex than this one, more works can be investigated to improve the animation.

Modeling results. Users is able to generate flowers quickly using this system. It only takes a few seconds to generate flowers even if users set all the parameters. Also, our system provides real-time feedback when the parameters are changed. As mentioned before, it is capable of producing a garden rapidly using our system.

We create 10 different flowers in Fig. 20. At the same time, our system is not limited to real flowers, i.e., it is able to create imaginary and surreal flowers (Fig. 20i and Fig. 20j). We document the model information and time consumption in Table 2. Even for complex flowers, it only takes less than 5 seconds.

(a)


(b)


(c)

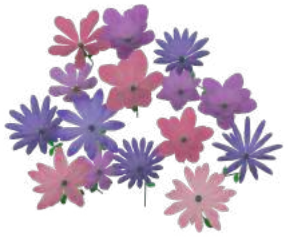

(d)

Figure 17: The scene with blossom clusters.

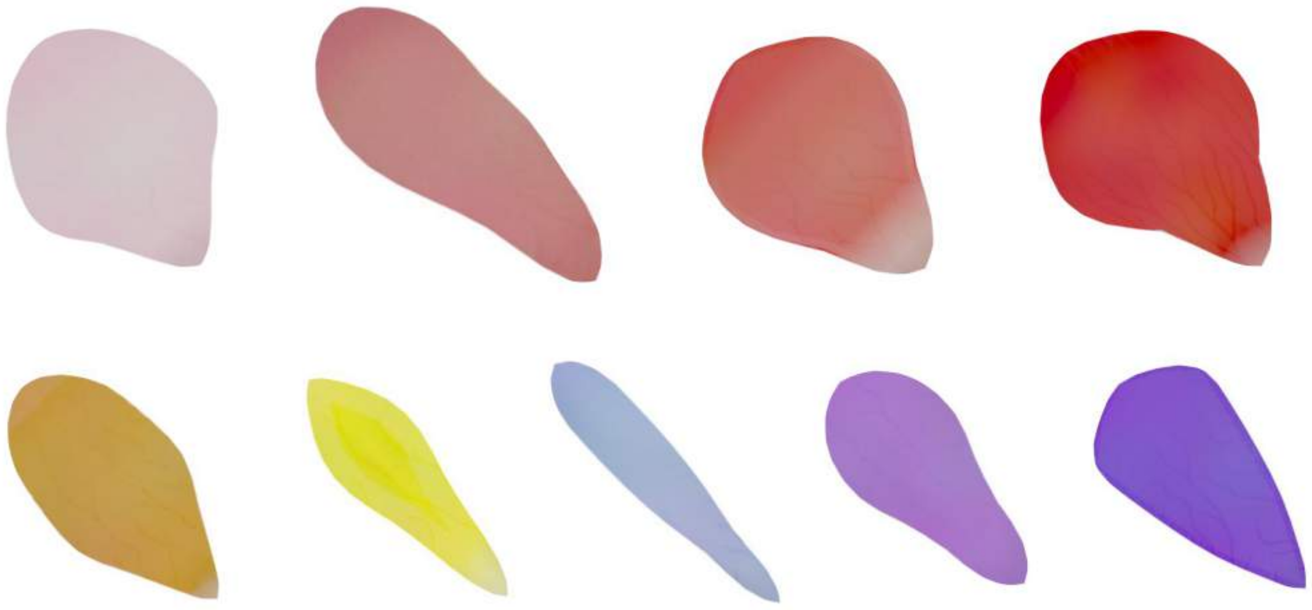


Figure 18: Different textures for the petals.

Table 1: The comparison of computation time between different approaches.

Method	#Petals	#Frames	Total time
Zheng et al. [35]	14	73	11min55s
Ours	25	50	4.21s

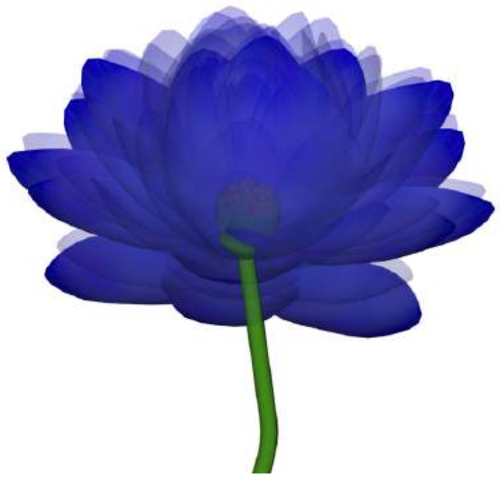
5.2 Comparison

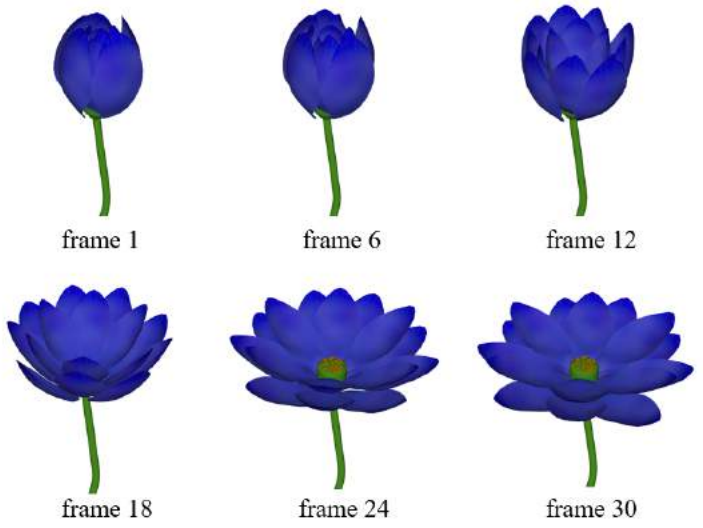
To investigate the efficiency, we make several experiments for the generation process. As shown in Fig. 21, the model generated by our method is compared with the model in [35], which creates the blooming sequence according to the captured point clouds. The model information and computation time are listed in Table 1. It takes 9.80 seconds per frame to generate the sequence, i.e., it takes more than 11 minutes to generate the bloom animation. However, it only takes 4.21 seconds to generate a entire blooming animation for a similar 3D flower using our method. Our method also generates components such as stamens, receptacle and leaves.

We also invited 3D designers to create similar flowers using their familiar tools. The designers employ Houdini [29] to model the flowers, and the results are listed in Fig. 22. In this experiment, we produced two template flowers and asked the designers to generate similar ones. The geometries are similar, but the textures are slightly different from the texture drawn by designers. It takes 4 hours to create each flower for the designers, including the modeling and the texture generation. But it only takes 3 minutes to produce similar flowers for experienced users with our system. The time comparison is documented in Table 4.

5.3 User study

Finally, we conduct a user study to evaluate the effectiveness of our system. We invited 16 designers and ordinary users to test and evaluate our system. We design a questionnaire which reflects six qualities of our system. Table 3 documents the average scores for the six qualities. It shows the strengths of our system and where need to be enhanced. The scores range from 0 to 10, where 0 is the worst and 10 is the best. The results show that our system is


(a)



(b)

Figure 19: The blooming animation.

Table 2: The parameters and information of the flowers.

Example	#Petals	#Stamens	#Leaves	#Vertices	#Faces	Computation time
Fig. 20a	8	21	2	4186	8111	3.22s
Fig. 20c	12	11	2	4865	9327	3.24s
Fig. 20d	13	50	2	8071	15487	4.50s
Fig. 20e	21	22	2	5073	9495	2.98s
Fig. 20f	8	56	4	7906	15551	2.67s
Fig. 20g	55	35	0	13518	25495	4.46s
Fig. 20h	7	6	2	3891	7519	3.06s

easy to learn (7.87) and easy to use (8.06). Also, the parameters are reasonable (8.31) and the flowers are diverse (7.62). However, the visual aesthetics (6.75) and the fidelity (6.68) are relatively low.

Intraclass Correlation Coefficient (ICC) [32] is employed to evaluate the consistency of the questionnaire. We calculated ICC for the user study, and the average measurement is 0.932. If the measurement is larger than 0.75, the consistency is high. This confirms the consistency and credibility of our result.

Table 3: The result of the user study.

Attribute	Average score (0-10)
Easy to learn	7.87
Easy to use	8.06
Rationality of parameters	8.31
Diversity of flowers	7.62
Visual aesthetics	6.75
Fidelity of results	6.68

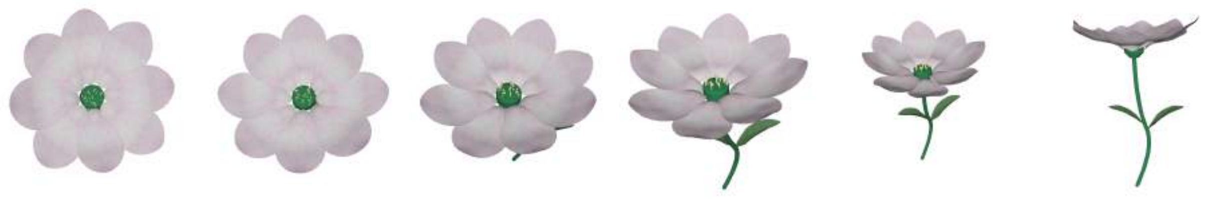
5.4 Limitations

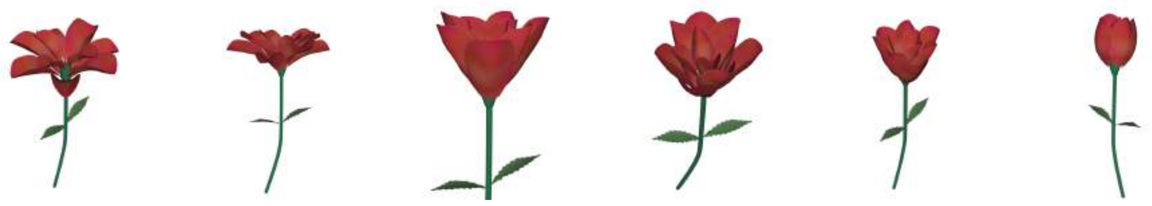
Our system is not without limitations. Firstly, our approach cannot generate inflorescence directly at present like [9, 18]. It also does not include the modeling of carpel and sepal. Secondly, our system does not employ biological knowledge for the modeling. As a result, it can not produce flowers in actinomorphic or zygomorphic manner. Thirdly, our system cannot produce flowers directly from photos, which is more convenient for designers to generate real flower models.

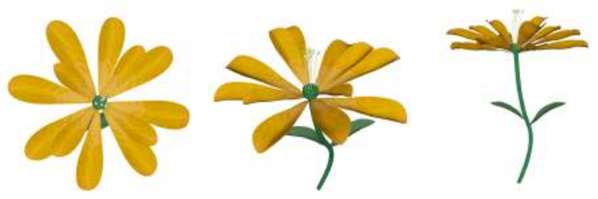
6 Conclusion

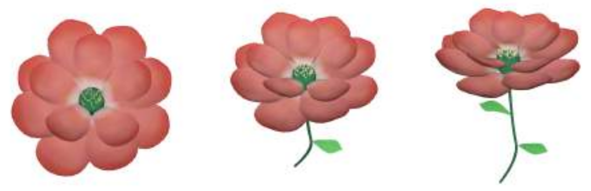
In this paper, we propose a framework for rapid modeling of flowers. It is able to generate 3D components for flowers and assemble them in various ways. The textures of the flowers can be synthesized automatically. The geometries and the textures can be controlled by adjusting the system parameters. This greatly simplifies the work for designers and improves their modeling efficiency. Artists can use our system to build a virtual garden landscape quickly with various flowers. They just need to finetune the parameters to obtain the desired styles, or they can modify the textures to meet their expectations. With an intuitive and easy-to-use interface, even novices without any modeling experience can create 3D flowers rapidly. Due to the low computational cost, our system can be integrated into a mobile application.

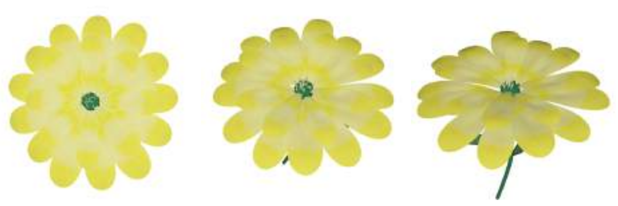
In the future, we plan to improve our system from three aspects. First, an image-based interface is helpful for the flower generation. This can further simplify the parameter setting procedure. Second, biologically-based rules may enhance the geometry and texture generation. The produced flowers may look more real and natural. Third, we consider to add more functions into our system such as inflorescence.

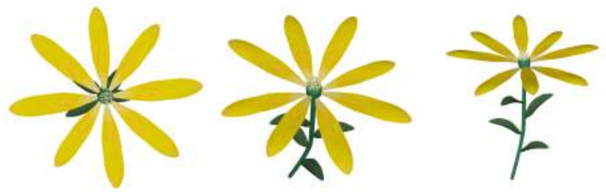

(a)

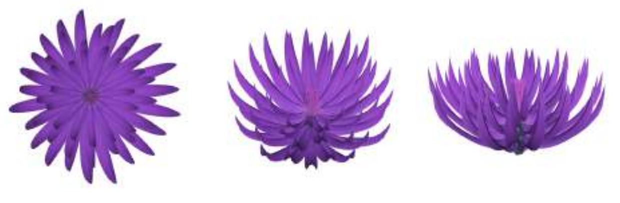

(b)

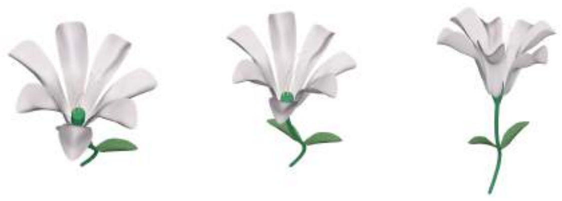

(c)

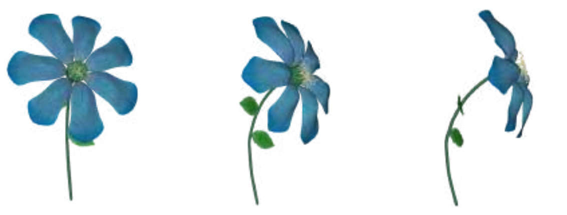

(d)


(e)


(f)


(g)


(h)


(i)

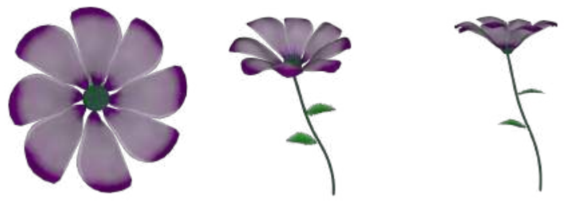

(j)

Figure 20: Generation of different flower models. The models in (b) are given with different opening angles. The other flowers are in different views.

Table 4: The comparison of modeling efficiency.

Example	Tool	Time				Total time
		Geometry	Texture	Parameters	Computation	
Fig. 22a,b	Our system	/	/	3.4min	4.2s	3.5min
	Houdini	2.8h	1.4h	/	/	4.2h
Fig. 22c	Our system	/	/	2.9min	3.7s	3min
	Houdini	2.4h	1.5h	/	/	3.9h

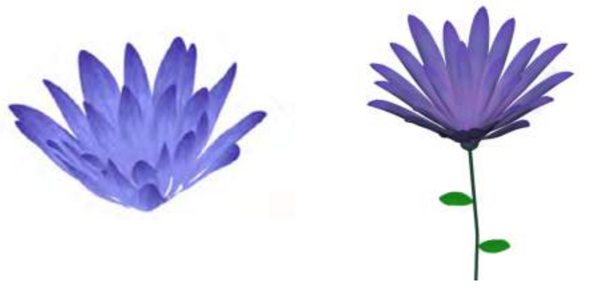


Figure 21: The comparison of modeling results between different approaches. (a) The 3D flower in [35]. (b) Similar flower generated using our approach.

Acknowledgments

This research is supported by National Key R&D Program of China (No.2018YFC0115102), National Natural Science Foundation of China (NO.61872020, 61532002, 61672149), Beijing Natural Science Foundation Haidian Primitive Innovation Joint Fund (L182016), Shenzhen Research Institute of Big Data, Shenzhen, 518000. We also thank the Faculty of Media and Communication, Bournemouth University (UK) with its support of Global Visiting Fellowship for Dr. Junjun Pan.

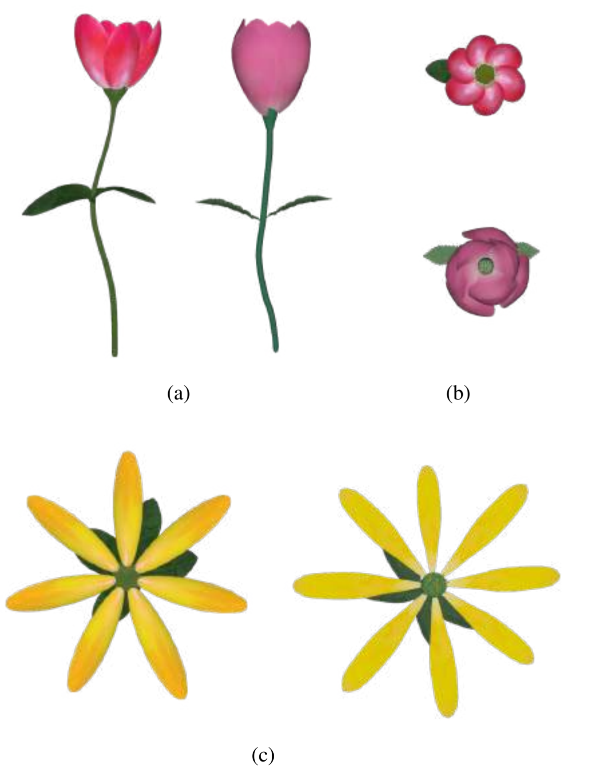


Figure 22: The comparison between 3D designers' flowers and ours. (a) The front view of the model. The left one is produced by designer and the right one is produced with our system. (b) The top view of the model. The above flower is produced by designer and the below flower is produced with our system. (c) The top view of the flower. The left one is produced by designer and the right one is produced with our system.

References

[1] M. Alsweis and O. Deussen. Procedural techniques for simulating the growth of plant leaves and adapting venation patterns. In *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology, VRST 2015*, pp. 95–101, 2015.

[2] F. Anastacio, M. C. Sousa, F. F. Samavati, and J. A. Jorge. Modeling plant structures using concept sketches. In *Proceedings of the 4th International Symposium on Non-Photorealistic Animation and Rendering 2006*, pp. 105–113, 2006.

[3] F. Boudon, P. Prusinkiewicz, P. Federl, C. Godin, and R. Karwowski. Interactive design of bonsai tree models. *Comput. Graph. Forum*, 22(3):591–599, 2003.

[4] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350 – 355, 1978.

[5] E. Chaudhry, A. Noreika, L. You, J. J. Zhang, J. Chang, H. Ugail, A. Malyshev, A. Carriazo, A. Iglesias, Z. Habib, A. B. Sargano, and H. Haron. Modelling and simulation of lily flowers using PDE surfaces. In *The 13th International Conference on Software, Knowledge, Information Management and Applications, SKIMA 2019*, pp. 1–8, 2019.

[6] Z. Ding, S. Xu, X. Ye, Y. Zhang, and S. Zhang. Flower solid modeling based on sketches. *Journal of Zhejiang University-SCIENCE A*, 9:481–488, 2008.

[7] T. Ijiri, T. Igarashi, S. Takahashi, and E. Shibayama. Sketch interface for 3d modeling of flowers. In R. Barzel, ed., *The 31st International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2004*, p. 6, 2004.

[8] T. Ijiri, S. Owada, and T. Igarashi. Seamless integration of initial sketching and subsequent detail editing in flower modeling. In *Comput. Graph. Forum*, vol. 25, pp. 617–624, 2006.

[9] T. Ijiri, S. Owada, M. Okabe, and T. Igarashi. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. *ACM Trans. Graph.*, 24(3):720–726, 2005.

[10] T. Ijiri, S. Yoshizawa, H. Yokota, and T. Igarashi. Flower modeling via x-ray computed tomography. *ACM Trans. Graph.*, 33(4):48:1–48:10, 2014.

[11] S. Jeong, S.-H. Park, and C.-H. Kim. Simulation of morphology changes in drying leaves. *Comput. Graph. Forum*, 32(1):204–215, 2013.

[12] B. Lintermann and O. Deussen. Interactive modeling of plants. *IEEE Computer Graphics and Applications*, 19(1):56–65, 1999.

[13] A. Lyndenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–315, 1968.

[14] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88*, p. 26–33, 1989.

[15] T. Miao, C. Zhao, X. Guo, and S. Lu. A framework for plant leaf modeling and shading. *Math. Comput. Model.*, 58(3-4):710–718, 2013.

[16] B. Neubert, T. Franken, and O. Deussen. Approximate image-based tree-modeling using particle flows. *ACM Trans. Graph.*, 26(3):88, 2007.

[17] M. Okabe, S. Owada, and T. Igarash. Interactive design of botanical trees using freehand sketches and example-based editing. *Comput. Graph. Forum*, 24(3):487–496, 2005.

[18] A. Owens, M. Cieslak, J. Hart, R. Classen-Bockhoff, and P. Prusinkiewicz. Modeling dense inflorescences. *ACM Trans. Graph.*, 35(4):136:1–136:14, 2016.

[19] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988.

[20] O. Petrenko, M. Sbert, O. Terraz, and D. Ghazanfarpour. *3Gmap L-Systems Grammar Application to the Modeling of Flowering Plants*, vol. 441, pp. 1–21. 2013.

[21] A. Peyrat, O. Terraz, S. Mérillou, and E. Galin. Generating vast varieties of realistic leaves with parametric 2gmap l-systems. *Vis. Comput.*, 24(7-9):807–816, 2008.

[22] J. L. Power, A. J. B. Brush, P. Prusinkiewicz, and D. Salesin. Interactive arrangement of botanical l-system models. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics, SI3D '99*, pp. 175–182. ACM, 1999.

[23] P. Prusinkiewicz, M. Hammel, and E. Mjolsness. Animation of plant development. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1993*, pp. 351–360, 1993.

[24] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. The virtual laboratory. Springer, 1990.

[25] P. Qin and C. Chen. Simulation model of flower using the integration of l-systems with bezier surfaces. *Computer Engineering & Applications*, 42(16):6–8, 2006.

[26] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang. Image-based plant modeling. *ACM Trans. Graph.*, 25(3):599–604, 2006.

[27] Y. Rodkaew, P. Chongstitvatana, S. Siripant, and C. Lursinsap. Modeling plant leaves in marble-patterned colours with particle transportation system. In *Proceedings of the 4th International workshop on functional-structural plant models*, pp. 391–397, 2004.

[28] A. Runions, M. Fuhrer, B. Lane, P. Federl, A. Rolland-Lagan, and P. Prusinkiewicz. Modeling and visualization of leaf venation patterns. *ACM Trans. Graph.*, 24(3):702–711, 2005.

[29] SideFX. Houdini. `http://www.sidefx.com`, 2020.

[30] S. Sloan. A fast algorithm for generating constrained delaunay triangulations. *Computers & Structures*, 47(3):441 – 450, 1993.

[31] P. Tan, T. Fang, J. Xiao, P. Zhao, and L. Quan. Single image tree modeling. *ACM Trans. Graph.*, 27(5):108, 2008.

[32] J. P. Weir. Quantifying test-retest reliability using the intraclass correlation coefficient and the sem. *Journal of strength and conditioning research*, 19(1):231–40, 2005.

[33] F. Yan, M. Gong, D. Cohen-Or, O. Deussen, and B. Chen. Flower reconstruction from a single photo. *Comput. Graph. Forum*, 33(2):439–447, 2014.

[34] C. Zhang, M. Ye, B. Fu, and R. Yang. Data-driven flower petal modeling with botany priors. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014*, pp. 636–643, 2014.

[35] Q. Zheng, X. Fan, M. Gong, A. Sharf, O. Deussen, and H. Huang. 4d reconstruction of blooming flowers. *Comput. Graph. Forum*, 36(6):405–417, 2017.

APPENDIX

Table 1: Parameter list for petal.

Parameters	Type	Min value	Max value	Attribute
Width	float	0.05	0.4	width of petal
Length	float	0.1	0.5	length of petal
Symmetry	bool	/	/	symmetry or not
ControlPoints	int	3	10	the number of control points of edge curve

Table 2: Parameter list for receptacle.

Parameters	Type	Min value	Max value	Attribute
MinWidth	float	0.03	0.08	min width of receptacle
MaxWidth	float	0.05	0.1	max width of receptacle
Height	float	0.02	0.1	height of receptacle
DivideExtent	int	4	7	extent of subdivision
BranchRadius	float	0.02	0.7	radius of branch
BranchLength	float	0	2	length of branch
BranchCurve	float	0	1	choose the curve of branch

Table 3: Parameter list for stamen.

Parameters	Type	Min value	Max value	Attribute
CylinderHeight	float	0.01	0.08	height of cylinder in stamen
CylinderWidth	float	0.01	0.05	width of cylinder in stamen
DivideExtent	int	4	6	extent of subdivision
TubeRadius	float	0.02	0.7	radius of tube
TubeLength	float	0	2	length of tube
TubeCurve	float	0	1	choose the curve of tube

Table 4: Parameter list for leaf.

Parameters	Type	Min value	Max value	Attribute
Width	float	0.05	0.4	width of leaf
Length	float	0.1	0.5	length of leaf
Symmetry	bool	/	/	symmetry or not
ControlPoints	int	3	10	the number of control points of edge curve
SerrateMode	bool	/	/	serrate or not
SawtNum	int	5	30	the number of saw-teeth in each edge
SawHeight	float	0.01	0.3	the height of sawteeth in each edge

Table 5: Parameter list for animation.

Parameters	Type	Min value	Max value	Attribute
FrameNum	int	10	100	the number of frames
FPS	int	20	80	the number of frames transmitted per second
SpeedCurve	float	0	1	speed curve of animation

Table 6: Parameter list for assembling petals.

Parameters	Type	Min value	Max value	Attribute
LayerNum	int	1	5	the number of petal layers
LayerXNum	int	1	20	the number of petals in layer X
LayerXSize	float	0.1	1.2	size of petals in layer X
Radius	float	0.02	0.08	radius of the circle surrounded by petals
CenterHeight	float	-0.1	0	height of petals in layer 1
DeltaRadius	float	0	0.3	radius difference between adjacent layers
DeltaCenter	float	-0.3	0	height difference between adjacent layers
DeltaCenterIn	float	0	0.1	height difference between adjacent petals in same layer
DeltaAngle	float	0	3.14	angle difference between adjacent layers
CloseAngle	float	15	50	the angle when the flower is closed
OpenAngle	float	40	100	the angle when the flower is opened
BudRadius	float	0.03	0.2	the maximum circle radius of flower bud
BudPosition	float	0	0.95	the position of the maximum circle of flower bud
CentralCurve	float	0	1	curve of petal's central vein
Weight	float	-30	30	weight in Fig. 6
CloseRandom	float	0	20	randomization level of close angle
OpenRandom	float	0	20	randomization level of open angle
SizeRandom	float	0	0.2	randomization level of petal's size

Table 7: Parameter list for assembling stamens.

Parameters	Type	Min value	Max value	Attribute
Num	int	0	200	the number of stamens
RandomSeed	int	0	10	seed for random number generation

Table 8: Parameter list for leaf texture.

Parameters	Type	Min value	Max value	Attribute
Color	vector	(0,0,0)	(1,1,1)	base color of leaf
ContourWidth	float	0	1	contour width of leaf's texture
ContourWeight	float	0.7	1.3	weight of contour
VeinWidth	float	0.02	0.2	width of leaf's vein
VeinNum	int	3	10	the number of veins
VeinWeight	float	0.8	1.2	weight of vein
CurveNum	int	0	5	the number of curves to change color
CurveWeight	float	0.9	1.1	weight of curve
WeightRandom	float	0	0.1	randomization level of weight

Table 9: Parameter list for assembling leaves.

Parameters	Type	Min value	Max value	Attribute
LayerNum	int	1	5	the number of leaf layers
LayerXNum	int	0	5	the number of leaves in layer X
LayerXSize	float	0.3	1	size of petals in layer X
Radius	float	0	0.6	radius of the circle surrounded by leaves
CenterHeight	float	-0.1	0	height of leaves in layer 1
DeltaAngle	float	0	3.14	angle difference between adjacent layers
RotateAngle	float	30	150	rotate angle of leaves
IntervalHeight	float	0.1	0.5	the interval height of two layers
DeltaHeight	float	0	0.1	height difference between adjacent leaves in same layer
CentralCurve	float	0	1	curve of leaf's central vein
Weight	float	-30	30	weight in Fig. 6
SizeRandom	float	0	0.2	randomization level of petal's size
AngleRandom	float	0	10	randomization level of rotate angle

Table 10: Parameter list for petal texture.

Parameters	Type	Min value	Max value	Attribute
Color	vector	(0,0,0)	(1,1,1)	base color of petal
ContourWidth	float	0	1	contour width of petal's texture
ContourWeight	float	0.7	1.3	weight of contour
StreakWidth	float	0.02	0.2	width of petal's streak
MaxWeight	float	1	1.2	max weight of streak
MinWeight	float	0.8	1	min weight of streak
RootWidth	float	0	1	width of petal's root
RootWeight	float	0.8	1.2	weight of root
CurveNum	int	0	5	the number of curves to change color
CurveWeight	float	0.9	1.1	weight of curve
WeightRandom	float	0	0.1	randomization level of weight

Table 11: Parameter list for receptacle texture.

Parameters	Type	Min value	Max value	Attribute
Color	vector	(0,0,0)	(1,1,1)	base color of receptacle
WeightRandom	float	0	0.2	randomization level of weight

Table 12: Parameter list for stamen texture.

Parameters	Type	Min value	Max value	Attribute
Color	vector	(0,0,0)	(1,1,1)	base color of stamen
WeightRandom	float	0	0.2	randomization level of weight