Sketch-based Shape-preserving Tree Animations

Yutong Wang[1], Luyuan Wang[1], Zhigang Deng[2,3], Xiaogang Jin[1*]

[1] State Key Lab of CAD&CG, Zhejiang University, Hangzhou, 310058, China

[2] Virtual Reality and Interactive Technique Institute, East China Jiaotong University

[3] Computer Science Department, University of Houston, Houston, TX, USA

jin@cad.zju.edu.cn

Abstract

We present a novel and intuitive sketch-based tree animation technique, targeting on generating a new type of special effect of smoothly transforming leafy trees into morphologically different new shapes. Both topological consistencies of branches and meaningful inbetween crown shapes are preserved during the transformation. Specifically, it takes a leafy tree and a user's sketch describing the silhouette of the desired crown shape under a certain viewpoint as the input. Based on a self-adaptive multiscale cage tree representation, branches are locally transformed through a series of topology-aware deformations, and the resulting tree conforms to the user-designed shape, demonstrating better aesthetics compared to global single cage-based methods. By interpolating the transformations, we are able to create visually pleasing shape-preserving animations of trees transforming between two crown shapes. Our proposed framework also provides an efficient way to interactively edit leafy trees towards desired shapes, demonstrating its potential to leverage existing tree modeling frameworks by providing flexible and intuitive tree editing operations.

Keywords: sketch-based editing, tree animation, shape-preservation, special effects

1 Introduction

Despite its long history, the study of tree animations remains active. Other than animating real-wrold scenarios, such as trees' growth process [1, 2] and their interactions with environment [3, 4], special requirements of animating stylized trees arise in the film and animation industry in order to nurture the mysterious atmosphere with compelling visual effects [5]. Consequently, topology-aware approaches have been proposed to generate smooth transformations between trees, and achieved visually compelling special effects [6, 7, 8]. However, they do not put explicit efforts on preserving a sequence of morphologically continuous crowns when transforming between trees with specialized crown shapes, leading to less meaningful inbetween trees, see Figure 10 (top row).

In this paper, we propose a novel sketch-based shape-preserving tree animation method, which smoothly transforms a leafy tree into custom crown shapes defined by designers' sketches while preserving the morphological meanings of inbetween trees. Our method contributes to a new type of special visual effect for leafy trees. Figure 1 shows an example of fluidly transforming a *Salix* with sphere-like crown into a heart-shaped one. It is noteworthy that both the topological consistencies of branches and the morphological meanings of crowns are preserved during the animation. One additional benefit of our method is that it also provides an intuitive way to efficiently control and edit leafy trees into desired crown shapes using one user-designed sketch alone.

The key technical challenge to achieve shape-preserving tree animations is how to transform tree branches to form morphologically meaning-

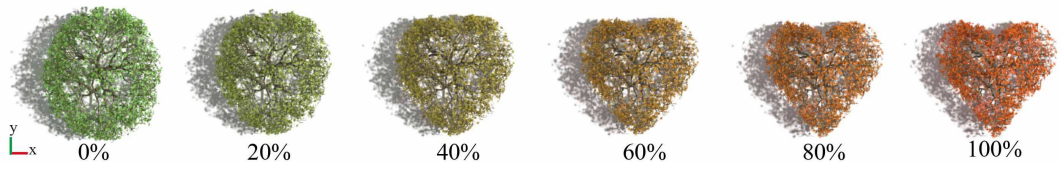


Figure 1: Shape-preserving transformations of a sphere-shaped *Salix* into a heart-shaped one.

ful crown shapes while preserving their topological hierarchies. To tackle the challenge, we propose a self-adaptive multiscale cage-based deformation strategy to hierarchically propagate the transformations of crowns to tree branches. Specifically, the input leafy tree is first abstracted into a skeleton-lobe representation [7], where the foliage is clustered into canonical geometries (i.e., *lobes*) in order to describe the tree in a hierarchical multiscale manner. The morphologically meaningful and continuous inbetween trees are achieved by first gradually deforming the tree's crown according to the designer's sketch, and then hierarchically propagating the deformation to tree branches based on the self-adaptive multiscale cage tree, in which the nodes are the view-dependent silhouettes serving as cages and the edges maintain their topological hierarchies. Since branches are transformed under both topology-aware and shape-aware deformations, the morphological meanings of the resulting trees are guaranteed.

Contributions. To the best of our knowledge, our work creates the first not only topologically-consistent but also shape-preserving animations of transforming leafy trees into morphologically different new shapes. The contributions are twofold: (i) a novel tree animation algorithm that smoothly transforms a tree to crowns with custom shapes while preserving the topological consistency of branches and morphological meanings of crown shapes; (ii) an intuitive tree editing method that allows designers to directly control the shape of the tree through view-dependent oversketching operations.

2 Related Work

Shape-guided tree modeling. To date, a large number of shape-guided tree modeling methods have been proposed to generate trees conforming to custom crown shapes. Specifically, there are 3D shape-guided methods [9, 10, 11, 12] and sketch-based interactive methods [13, 14, 15, 16]. However, both of them only generate static trees and are sensitive to the change of the guidance shapes. In other words, whenever the guidance shape is changed, the algorithm just rerun to regenerate a new tree. Because of the low efficiency and no warranty of topologically consistent tree branches, they are not suitable for generating shape-preserving tree animations.

Tree animations. Existing literature on tree animations can be roughly classified into two categories: animating trees' development and their interactions with environment. Procedural methods such as [1, 17, 2] have been proposed to create botanically plausible tree growth animations by interpolating a number of predefined developmental parameters. However, none of them is capable of generating shape-preserving animations of transforming leafy trees into custom shapes. The main reason is that these developmental parameters have no direct association with on trees' crown shapes, and interpolating them cannot guarantee to generate morphologically continuous and meaningful inbetween crowns. In addition to growth animations, various efforts have been conducted to deform and prune branches of a tree in order to generate realistic responses to environmental factors, such as wind [18, 4], light [3], fire [19, 20], and obstacles [21, 3]. Despite that the shape of the tree is changed according to the environment, it cannot be smoothly transformed into a user-defined shape since existing methods do not put explicit efforts on preserving the morphological meanings of trees' crown shapes.

Topology-aware tree animations. Recent works [6, 7] create topologically consistent animations between two topologically varying trees. However, [6] does not handle the animations of foliage. Although [7] addresses the smooth transformations of foliage, it fails to preserve the morphological meanings of inbe-

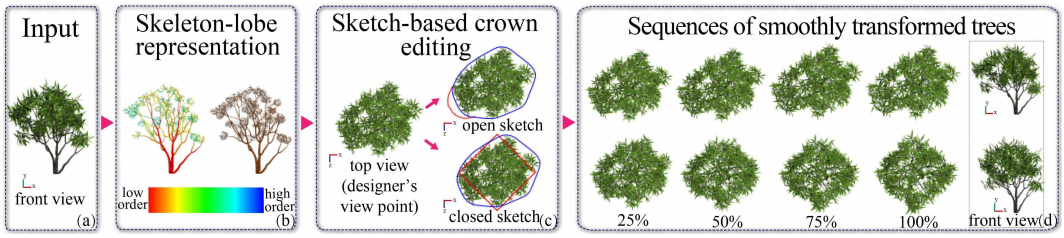


Figure 2: The pipeline of our method.

tween trees' crowns, leading to less compelling visual effects when transforming between trees with custom crowns. In addition, this method requires exact two trees as input whereas ours only requires one tree and a designer's sketch describing the silhouette of the desired crown.

3 Approach Overview

Figure 2 illustrates the main steps of our shape-preserving tree animation approach. Given a leafy tree (Figure 2(a)), it is first converted into a branching-pattern aware, skeleton-lobe representation using [7] (Figure 2(b)). From a certain viewpoint, designers are allowed to change the shape of the tree by either oversketching the detected crown's silhouette (open sketch) or draw a new shape describing the desired crown shape under the viewpoint (closed sketch), see Figure 2(c), where the crown silhouettes are plotted in blue whereas designers' sketches are plotted in red. The crown deforms according to the new silhouette, and hierarchically propagates the deformation to tree branches based on a multi-scale cage tree (*MCT*), which preserves the hierarchical topology of both tree branches and their corresponding lobes. As a result, the tree is gradually transformed, and a fluid animation of transforming a tree into the sketched crown shape is generated (Figure 2(d)).

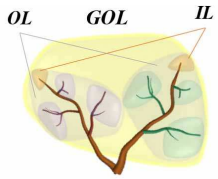
4 Skeleton-lobe Representation

Given a leafy tree, we first abstract it into a branching-pattern aware, skeleton-lobe representation as in [7]. Below we summarize the essential terms pertinent to our work. Readers are referred to [7] for more algorithm details.

Chain. To describe the branching hierarchies of a tree, we label the tree branches with hybrid orderings [7] and define the consecutive branches without ordering changes as a *chain*.

Branching pattern. In real world, the arrangement of branches at a branching point generally follows one of three patterns: *alternative*, *opposite* and *whorled* [22]. In this paper, we encode the branching patterns into *chain groups*, which consist of chains growing at the same branching points while sharing the same orderings.

Lobe. The general term used for canonical geometry formed by a cluster of leaves.



Multiscale lobe descriptions. Following the work of [7], three types of lobes (see inset) are defined in our work. Specifically, the *inner* lobe (*IL*), which only contains leaves growing at the tip of a chain, is interpreted as the finest scale of description; the *outer* lobe (*OL*) conveys a coarse description of a chain's foliage by describing the shape formed by the leaves that covers all its substructures; the *group outer* lobe (*GOL*), defined as the coarsest scale of description, depicts the general shape of a chain group's foliage and mathematically equals to the union of *OL*s of the chains within the group. It is noteworthy that the *GOL* is equivalent to *OL* if and only if the chain group consists of only one chain. This leads to a multiscale description of a tree's foliage, see Figure 3(a). At the coarsest scale, the tree is only described by the root chain group and the *GOL*, which is also referred as the *crown* in this paper. More detailed foliage is described at finer scales, where the tree is described by higher-ordered chain groups and their *GOL*/*OL*s, and at the finest scale, the tree is described by all the chain groups and the *IL*s.

Branching pattern-aware topology tree. With the multiscale lobe descriptions, we encode their topological hierarchies into a branching

pattern-aware topology tree (i.e., the *BPTT*) [7], see Figure 3(b). Nodes of the *BPTT* are defined as a tuple $N_i :=< GOL_i, GC_i >$, where GOL_i is the *GOL* of the chain group GC_i exhibiting a certain branching pattern. Edges in the *BPTT* are defined in the form of (N_i, N_j), implying a hierarchical relationship between N_i and N_j on the condition that GOL_i is the parent of GOL_j.

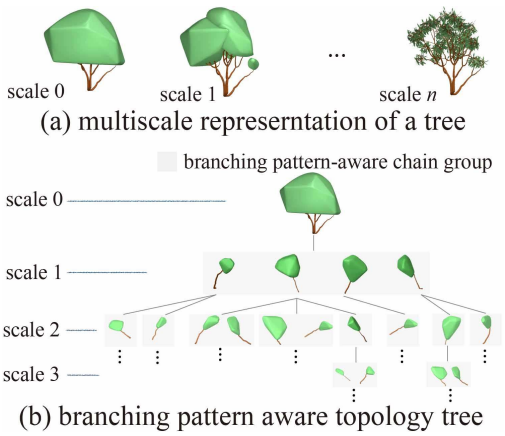


Figure 3: Multiscale representations of a tree.

5 Sketch-based Tree Animation

5.1 Sketch-based Crown Editing

The shape-preserving tree animation is initiated by the change of the tree's crown. In our method, we employ view-dependent sketching operations to interactively edit a leafy tree's crown shape. The edit of a crown is accomplished in two steps: (i) detect the crown's silhouette; (ii) oversketch a part of the silhouette (open sketch) or redraw a new (closed) sketch to describe the desired shape, see Figure 2(c).

Silhouette detection. Among various silhouette detection approaches [23], we choose the object-space silhouette (or simply the silhouette) for its accuracy and convenience to potential editing operations. Specifically, the silhouette is piecewise linear, denoted as $S = \{p_i \mid i \in (1, \ldots, n)\}$, and p_i satisfies $n_i \cdot (p_i - C) = 0$, where n_i is the vertex normal and C is the viewing position.

Silhouette oversketching. Designers are allowed to either draw an open sketch (red line in Figure 2(c) top row) to slightly modify the crown shape (Figure 2(d) top row) or redraw a closed sketch (red line in Figure 2(c) bottom row) to define a new shape (Figure 2(d) bottom row). Both of the sketches are view-dependent, representing the projection of the desired crown shapes in the screen space, which are later transformed to the object space by establishing a mapping to the crown silhouette using screen space arc-length parameterization [24].

5.2 View-dependent Multiscale Cage Tree

According to the *BPTT*, under a certain viewpoint, the silhouettes of a tree's lobes, including *GOL*s, *OL*s and *IL*s, also exhibit hierarchical features. In addition, the fact that coarser scale lobes fully envelop the finer scale lobes makes it reasonable to consider the coarser scale silhouettes as cages of the finer scale silhouettes. Therefore, we formalize the hierarchical silhouettes into a multiscale cage tree (in graph-theoretic sense), i.e., *MCT*, in which the nodes are the view-dependent silhouettes, viewed as cages, and the edges encode the parent-child relations between nodes, see Figure 5. Based on the objects that are directly influenced by the changes of cage vertices, two types of cages are distinguished: *leaf cages*, corresponding to the *IL*'s silhouettes, that directly deform tree branches, and *internal cages* (none-leaf cages), the silhouettes of the *GOL* and *OL*, that only affect the finer scale cages.

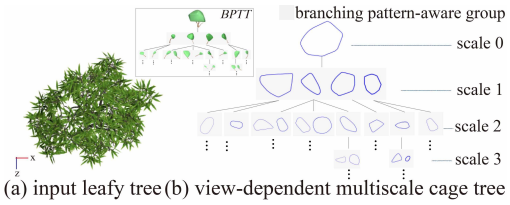
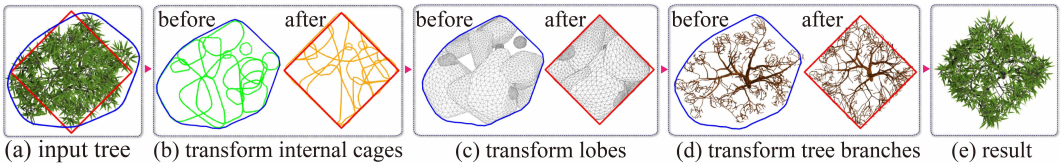


Figure 5: The view-dependent multiscale cage tree.

It is noteworthy that the hierarchical and multiscale nature of the *MCT* implies that a coarser scale cage only directly controls the deformation of its child cages at the finer scale and any cage vertex can only be controlled by the parent cage. In other words, when a cage at a certain scale changes, the transformation should only be propagated downwards the *MCT* hierarchy. In addition to maintain the topological hierarchies of lobes' silhouettes, the *MCT* brings two additional benefits. It not only preserves the morpho-



(a) input tree (b) transform internal cages (c) transform lobes (d) transform tree branches (e) result

Figure 4: The main steps of the baseline transformation.

logical consistencies of branches and lobes by ensuring that transformations can only be propagated from the coarser scale cages to the finer scale cages, but also provides the capability of producing *local* transformations at any scales, see the self-adaptive cages in Section 5.4.1.

5.3 The Baseline Transformation

Our algorithm hierarchically propagates the transformation induced by the sketch downwards the *MCT* to the finer scale internal cages and finally to leaf cages, resulting in a shape-preserving transformation of the tree. Specifically, it works in three steps, see Figure 4.

Step 1: transform internal cages (Figure 4(b)). We employ mean value coordinates (*MVC*) [25] to compute the new cage vertices resulted from the transformation of the coarser cages. By definition, the *MVC* of a point p_i inside a given cage $\mathcal{C}$ is a set of parameters w_{ij} satisfying $p_i = \sum_{v_j \in \mathcal{C}} w_{ij} v_j$, where v_j is the cage vertex. This implies that any point inside a cage can be represented as a linear combination of the cage vertices. Therefore, when the coarser scale cage is transformed, the new position of a finer scale cage vertex $p_i^{'}$ is calculated using the *MVC* w_{ij} as $p_i^{'} = \sum_{v_j \in \mathcal{C}} w_{ij} v_j^{'}$, where $\mathcal{C}$ is the coarser scale cage and $v_j^{'}$ is the changed cage vertex.

Step 2: transform lobes (Figure 4(c)). The change of an *internal* cage automatically induces the corresponding lobe to deform. By fixing the cage vertices as constraints, we use the Laplacian deformation algorithm [24] to compute the deformed shape by minimizing the following linear system:

$$\sum_{v_i^{'} \in \mathcal{M}} \|\mathcal{L}\left(v_i^{'}\right) - \sigma_i\|^2 + \sum_{v_j \in \mathcal{C}} \|v_j^{'} - v_j\|^2, \quad (1)$$

where $\mathcal{L}(\cdot)$ is the Laplacian operator, σ_i is the Laplacian coordinate computed using cotangent weights, $v_i^{'}$ is the transformed surface point, and v_j is the constraint cage vertex with transformed new position $v_j^{'}$. The first term perserves the local details of the shape in $\mathcal{L}(\cdot)$, and the second term penalizes the changes of the constraint points during the transformation.

Step 3: transform tree branches (Figure 4(d)). Tree branches are not transformed until the transformation is propagated to the *leaf* cages. To maintain the topological consistency, they are transformed by two consecutive deformation propagations: the *backward* and *forward* propagation. Given a branch and its transformed IL, the target position of the branch tip is calculated using the *MVC* coordinates [25]. Served as a constraint point, the branch tip is transformed to the target position and the transformation is propagated backwards to the root of the branch using Eq. (1). Once done, the branch forwards the transformation to its child branches by fixing the child roots as the constraint points and performing the transformation in the similar way using Eq. (1).

By interpolating the transformation process, the tree is smoothly transformed into the designer's custom shape, without violating branches' topological consistencies.

5.4 Shape-preserving Transformation

Although maintaining the topological consistency, the above propagation does not guarantee the shape-preserving transformations of tree branches. In other words, the resulting trees might poorly match the designer's expectations, see the circled regions in Figure 6(b). Two major factors that weaken the aesthetics of the resulting trees are *poor crown coverage* and *unnaturally deformed branches*, which can be solved by employing shape-aware self-adaptive cages and pruning branches outside the designer's desired shapes, respectively.

5.4.1 Self-adaptive Cages

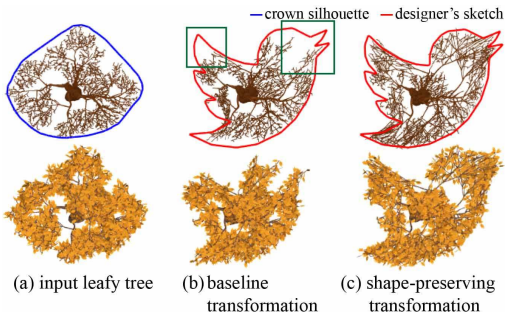


Figure 6: Comparison of the baseline transformation (b) and the self-adaptive shape-preserving transformation (c).

The issue that a resulting tree poorly conforms to the target crown shape (green regions in Figure 6(b)) arises when the branches of the input leafy tree do not give a full coverage of the crown (Figure 6(a)). Unfortunately, this is a natural phenomenon of real world trees since the growth of both branches and leaves are influenced by environmental factors, which might shed branches and leaves because of lights, winds, etc. In our baseline transformation algorithm, cages and tree branches are transformed according to the constraint points computed by *MVC* coordinates that preserve the positions of the constraint points relative to their corresponding cages. Consequently, the uncovered regions of the input crown remain uncovered after the transformation, leading to unsatisfying result, see Figure 6(b).

To solve the problem, we introduce the shape-preserving self-adaptive cage into the baseline transformation, which automatically deforms the finer scale cages to "fill-in" the uncovered regions of their parental cages at the coarser scale. Specifically, when an *internal* cage is transformed and prior to forwarding the transformation to its children, we analyze and partition the cage into segments based on the coverage of the child cages, see Figure 7(a). Each of the segments is characterized as one of the two types: the *covered* or *uncovered* segments, denoted as O and F, respectively. A segment is label as O if there are at least one child cage that is partially matched within a distance threshold (10 pixels in the screen space), otherwise it is labeled as F, see Figure 7(a).

To maximize the coverage, we greedily deform the child cages. For instance, given an F segment with neighboring segments O_i and O_j, for every child cage that is partially matched to either O_i or O_j within a threshold (10 pixels in the screen space), we perform Laplacian deformation [24] to adjust the cage with strategically selected constraint points. With the entire child cage considered as the region of interest, the vertices of the cage that are partially matched to the O_i or O_j segment are automatically selected as constraint points. Furthermore, with the mappings induced by the screen-space arc-length parameterization [24] of the child cage and the F segment, the cage vertices that are matched with the F segment are also selected as additional constraint points. As a result, the cage is deformed to cover the F segment and then propagates the transformation to the finer scale cages based on the *MCT*.

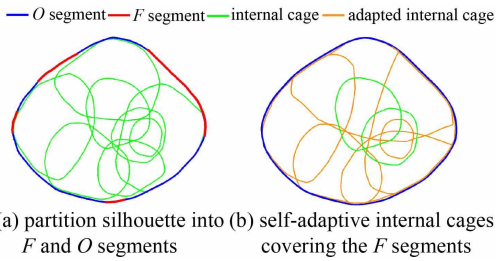


Figure 7: Self-adaptive cages.

5.4.2 Pruning Branches

In order to avoid unnaturally transformed branches, we propose a pruning algorithm to strategically wilt several branches, see the blue branches in Figure 8(b). The branch pruning is hierarchically performed based on the *MCT*. Starting from the coarsest scale, we clip the cage $\mathcal{C}$ with the designer's sketch using *Vatti*'s polygon clipping algorithm [26]. In case that the cage is clipped into multiple pieces, we choose the one with the largest area as the clipped cage $\mathcal{C}'$. The areas of both $\mathcal{C}$ and $\mathcal{C}'$, as well as the ratio r of the $\mathcal{C}'$'s area to the $\mathcal{C}$'s area, are calculated. Based on r, branches are pruned by performing the following tests:

(i) if r is smaller than a certain threshold, which is experimentally set to 30%, the cage is marked as pruning. Based on the *MCT*, all of the child cages are also marked as pruning to avoid the violation of the topological consistency;

(ii) if r is approximate to 1, it implies that the

cage is nearly completely inside the sketch and requires no more pruning tests;

(iii) otherwise, we test the finer scale cages based on the *MCT* and repeat the process until the *leaf* cages are tested or either the above condition (i) or (ii) is satisfied.

In the end, branches whose corresponding cages are marked as pruning are gradually wilted using the topology-aware method [7]. Figure 8 demonstrates an example of the branch pruning algorithm.

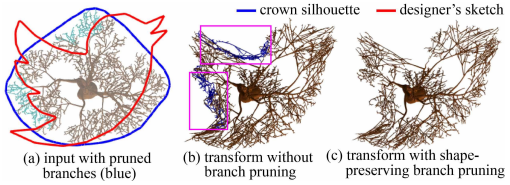


Figure 8: Comparison of pruning branches.

Ex.	Input		Time
	# Branches	# Leaves	(per frame)
Fig. 1	5,031	3,830	1.14s
Fig. 2	1,800	1,702	0.32s
Fig. 6	3,917	3,787	2.05s
Fig. 10	3,976	3,852	0.67s
Fig. 12	1,774	1,766	0.57s

Table 1: Runtime statistics.

6 Results and Discussion

6.1 Results

We collected 60 real-world trees from the Internet (i.e., http://www.evermotion.org/). In addition, 10 designers' hand-modeled virtual trees were also incorporated into the dataset to evaluate the effectiveness and flexibility of our algorithm. Figure 2 shows the result of smoothly transforming an oval-shaped *Cabbage* tree into slightly modified crown shape and a completely new diamond shape. Figures 1, 10 and 13 demonstrate the visually pleasing shape-preserving transformations for different tree species. Please refer to our animation results in the supplemental video.

Performance. We implemented our algorithm in C++ on a desktop equipped with Intel© i7 4.0GHz CPU and 32GB RAM. The runtime statistics are presented in Table 1. In sum, our method demonstrated its efficiency of generating smooth results on an off-the-shelf computer.

6.2 Discussion

Comparison with space colonization method. Figure 9 compares our work with the space colonization method [10], which generates trees with custom crown shapes using a procedural growth model to simulate the natural competitions between branches. Although targeting on generating static trees, the method of [10] demonstrates the possibility of generating a shape-preserving animation of tree transformation from one crown shape to another when a sequence of consistently transformed crown shapes is provided. Unfortunately, it failed to create a smoothly transformed tree sequence and generated discontinued branches (circled in red), as shown in Figure 9 (top row).

The reason is that the algorithm [10] must regenerate a new tree whenever the crown changes. Even with smoothly transformed crowns, the procedural nature of the algorithm cannot guarantee the generation of topologically consistent tree branches between consecutive frames, leading to unsuccessful animation. In contrast, our algorithm (the bottom row in Figure 9) avoids regenerating trees for each frame. Instead, it gradually deforms the crown and hierarchically propagates the crown's transformation to tree branches based on the self-adaptive *MCT*, therefore generates more visually pleasing animation.

Comparison with topologically consistent morphing. We compared our work with the most recent and relevant work by Wang et al. [7], which is capable of generating fluid morphing effects between two leafy trees. As shown in the bottom row of Figure 10, our method demonstrates the equal capability of preserving branches' topological consistency when transforming a tree into a new shape, but exhibits an evident advantage over the preservation of the meaningful in-between crown shapes. In addition, our algorithm only takes a leafy tree and a designer's sketch to achieve a visual effect, whereas [7] requires at least two leafy trees to create a morphing animation.

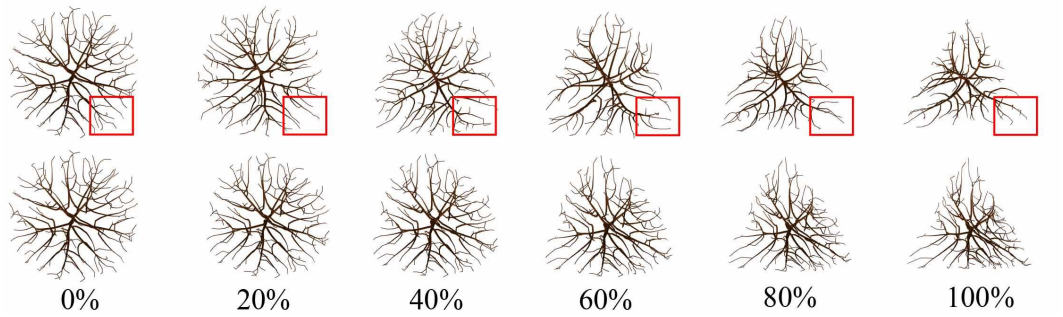


Figure 9: Comparison between results generated by our method (bottom row) and by the space colonization algorithm [10] (top row). The latter produces discontinuous tree branches between frames (marked in red) due to a lack of preservation of branches' topological consistencies.

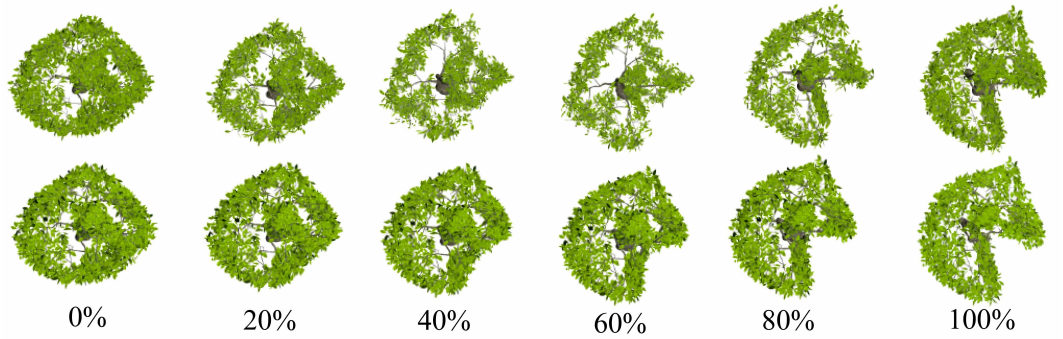
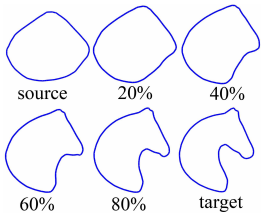


Figure 10: Comparison between our method (bottom row) and topologically consistent morphing [7] (top row). The latter generates less meaningful inbetween crowns due to the branch correspondences unaware of crown's morphological meanings.



In addition to visual comparisons, we quantify the shape preservation of the resulting tree sequences by measuring the similarities between the inbetween crowns' silhouettes and the morphologically meaningful reference silhouettes (see inset), which are achieved by linearly interpolating the source and the target crown silhouettes. The similarities between two crown silhouettes is measured by the discrete *Fréchet* distance [27], and the smaller the distance is, the greater the similarity is. Figure 11 plots the *Fréchet* distances of both methods against the time during the animation. Evidently, our method (red plot) demonstrates smaller distances to the reference silhouettes, implying greater similarities to the morphologically meaningful inbetween crown shapes compared to [7] (blue plot).

The main reason is that [7] directly transforms tree branches based on topology-aware correspondences, but does not put explicit efforts on maintaining the morphological meanings of the transformed crown shapes. In contrast, out method interpolates the crown shapes during the animation and hierarchically propagates the transformation of crowns to tree branches based on the *MCT*. Therefore, not only the topological hierarchy but also the morphologically consistent and the meaningful in-between crown shapes are preserved by our algorithm.

Comparison with single cage transformation. In Figure 12, we compared our self-adaptive multiscale transformation (Figure 12(d)) with the single-cage based global transformation (Figure 12(c)). It is clear that our method creates better results because of the locally transformed branches based on the self-adaptive *MCT*. Another noteworthy feature of our algorithm is the preservation of tree's natural branches and foliage. This can be explained by the multiscale skeleton-lobe representation, which provides an efficient way to reconstruct botanical-

ly meaningful trees from transformed skeletons, radii, and lobes [7].

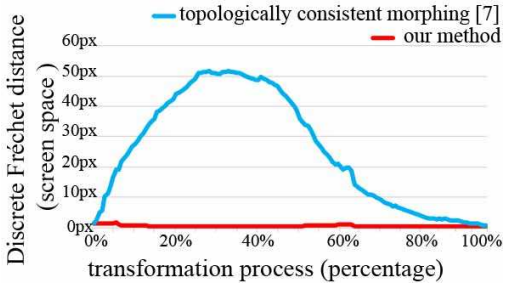
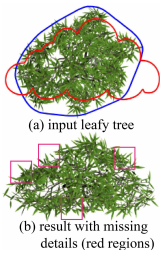


Figure 11: Comparison of the discrete Fréchet Distance between our method and topologically consistent morphing [7].

7 Conclusion

Aiming at generating a new type of visual effect, we present a novel and intuitive sketch-based tree animation method. It takes a leafy tree and a designer's sketch as input and smoothly transforms the tree into the custom crown shape. Both topologically consistent tree branches and morphological meaningful inbetween crowns are maintained by the *MCT*. Local deformations are supported by the self-adaptive multiscale cages, resulting in aesthetically pleasing results. In addition, our method could be potentially integrated into existing sketch-based tree modeling frameworks, such as [15, 14, 16], as an efficient and convenient way to generate special effects for leafy trees.



Limitations and Future Work. Our method still leaves room for improvement. Because of the cage-based transformation strategy, the results approximate to the designers' sketches by conforming to the dominant features and ignoring the subtle details (inset). In addition, since we employ a view-dependent sketch to edit a tree's crown shape, due to the lack of 3D information, the crowns of the resulting trees only transform meaningfully when observed from the viewpoint from which they are modified. Despite of this, thanks to the multiscale skeleton-lobe representation, the transformed trees remain to be botanically meaningful when viewed from other viewpoints, see the top view results shown in Figure 2(d). In the future, we plan to improve the crown editing operations by introducing direct three dimensional controls, and make the results meaningful under multiple viewpoints. Another promising future direction is to extend the current framework by generating shape-preserving animations between two topologically different trees as in [7].

References

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

[2] Sören Pirk, Till Niese, Oliver Deussen, and Boris Neubert. Capturing and animating the morphogenesis of polygonal tree models. *ACM Transactions on Graphics (TOG)*, 31(6):169, 2012.

[3] Sören Pirk, Ondrej Stava, Julian Kratt, Michel Abdul Massih Said, Boris Neubert, Randomir Mech, Bedrich Benes, and Oliver Deussen. Plastic trees: interactive self-adapting botanical tree models. *ACM Transactions on Graphics (TOG)*, 31(4):50, 2012.

[4] Sören Pirk, Till Niese, Torsten Hädrich, Bedrich Benes, and Oliver Deussen. Windy trees: Computing stress response for developmental tree models. *ACM Transactions on Graphics*, 33(6):204, 2014.

[5] Arthur Shek, Dylan Lacewell, Andrew Selle, Daniel Teece, and Tom Thompson. Art-directing disney's tangled procedural trees. *ACM SIGGRAPH 2010 Talks*, 53, 2010.

[6] Yutong Wang, Xiaowei Xue, Xiaogang Jin, and Zhigang Deng. Creative virtual tree modeling through hierarchical topology-preserving blending. *IEEE Transactions on Visualization and Computer Graphics*, 23(12):2521–2534, 2017.

[7] Yutong Wang, Luyuan Wang, Zhigang Deng, and Xiaogang Jin. Topologically consistent leafy tree morphing. *Computer Animation and Virtual Worlds*, 28(3-4), 2017.

[8] Guan Wang, Hamid Laga, Ning Xie, Jinyuan Jia, and Hedi Tabia. The shape space of 3d botanical tree models. *ACM Transactions on Graphics (TOG)*, 37:1–18, 2018.

[9] Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. Synthetic topiary. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*, pages 351–358. ACM, 1994.

[10] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling trees with a space colonization algorithm. In *In Proceedings of Eurographics Workshop on Natural Phenomena*, pages 63–70, 2007.

[11] Rui Wang, Yinhui Yang, Hongxin Zhang, and Hujun Bao. Variational tree synthesis. *Computer Graphics Forum*, 33(8):82–94, 2014.

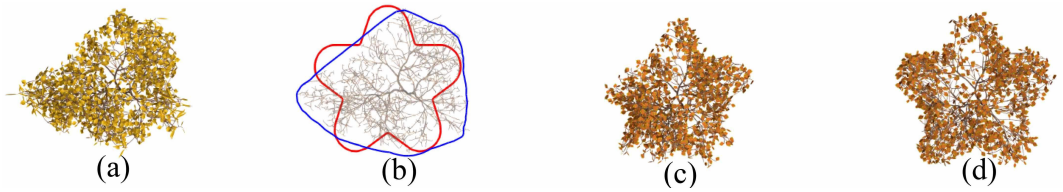


Figure 12: Comparison with the single cage-based global transformation. The input (a), the crown silhouette (blue) and the designer's sketch (red) (b), the results by the single cage-based global transformation (c) and our method (d).

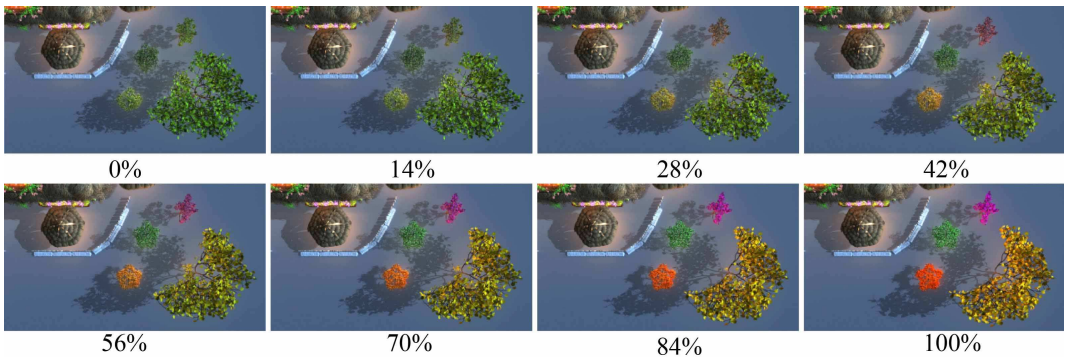


Figure 13: A fairy scene of transforming leafy trees into a moon and three stars.

[12] Ondrej Stava, Sören Pirk, Julian Kratt, Baoquan Chen, R Měch, Oliver Deussen, and Bedrich Benes. Inverse procedural modelling of trees. In *Computer Graphics Forum*, volume 33, pages 118–131. Wiley Online Library, 2014.

[13] Makoto Okabe, Shigeru Owada, and Takeo Igarash. Interactive design of botanical trees using freehand sketches and example-based editing. *Computer Graphics Forum*, 24(3):487–496, 2005.

[14] Jamie Wither, Frédéric Boudon, M-P Cani, and Christophe Godin. Structure from silhouettes: a new paradigm for fast sketch-based design of trees. *Computer Graphics Forum*, 28(2):541–550, 2009.

[15] Xuejin Chen, Boris Neubert, Ying-Qing Xu, Oliver Deussen, and Sing Bing Kang. Sketch-based tree modeling using markov random field. *ACM Transactions on Graphics (TOG)*, 27(5), 2008.

[16] Steven Longay, Adam Runions, Frédéric Boudon, and Przemyslaw Prusinkiewicz. Treesketch: interactive procedural modeling of trees on a tablet. *Proceedings of the International Symposium on Sketch-based Interfaces and Modeling*, 107–120, 2012.

[17] Przemyslaw Prusinkiewicz. Modeling plant growth and development. *Current Opinion in Plant Biology*, 7(1):79–83, 2004.

[18] Yili Zhao and Jernej Barbič. Interactive authoring of simulation-ready plants. *ACM Transactions on Graphics (TOG)*, 32(4):84, 2013.

[19] Shiguang Liu, Tai An, Zheng Gong, and Ichiro Hagiwara. Physically based simulation of solid objectsâĂŹ burning. In *Transactions on Edutainment VII*, pages 110–120. Springer, 2012.

[20] Sören Pirk, Michał Jarząbek, Torsten Hädrich, Dominik L Michels, and Wojciech Palubicki. Interactive wood combustion for botanical tree models. *ACM Transactions on Graphics (TOG)*, 36(6):197, 2017.

[21] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch, and Przemyslaw Prusinkiewicz. Self-organizing tree models for image synthesis. *ACM Transactions on Graphics (TOG)*, 28(3):58, 2009.

[22] Michael A Dirr. *Manual of woody landscape plants: their identification, ornamental characteristics, culture, propagation and uses.* Stipes Publishing Co, 1990.

[23] Aaron Hertzmann. Introduction to 3d non-photorealistic rendering: Silhouettes and outlines. *Non-Photorealistic Rendering, SIGGRAPH Course Notes*, 99(1), 1999.

[24] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics (TOG)*, 24(3):1142âĂŞ–1147, 2005.

[25] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics (TOG)*, 24(3):561–566, 2005.

[26] Bala R Vatti. A generic solution to polygon clipping. *Communications of the ACM*, 35(7):56–63, 1992.

[27] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Tech. Report CD-TR 94/64, Information Systems Departmen-t, Technical University of Vienna, 1994.