American Journal of Advanced Research, 2017, 1–1
Sept. 2017, pages 1-6
doi: 10.5281/zenodo.807664
http://www.ibii-us.org/Journals/AJAR/
ISBN 2572-9829 (Online), 2572-9810 (Print)

A Feature Enhancement Method for 3D Tree Synthesis

Ling Xu[1, *]

[1]Department of Computer Science and Engineering Technology, University of Houston-Downtown, USA

*Email: xul@uhd.edu

Received on May 05, 2017; revised on May 29, 2017; published on June 13, 2017

Abstract

Computer synthesized 3D tree models often appear in the virtual worlds of 3D movies and games. The creation of tree models with controllability of realistic features including branch irregularities and gaps between branch clusters is a challenging task. This paper presents a procedural method for 3D tree feature enhancement. The tree model is generated in a weighted graph and the basic geometric tree branches are least-cost paths from path planning. Through disconnecting edges in specified regions of holes in the graph, the method can control the tree branches to avoid the hole regions thus demonstrate crooked shapes. The distribution and the sizing of holes can be used to enhance the tree features of irregularity and heterogeneity.

Keywords: Procedural modeling, Tree modeling, Geometry synthesis

1 Introduction

With the fast development of computer-generated movies and games, the audiences and game players have high demands of realistic experience for the synthetic virtual scenes. Synthesizing realistic 3D worlds becomes a more and more important topic in both computer research and industrial areas. As a commonly seen component of the nature world, trees play an important role in decorating the 3D natural scenes, by attracting the viewer attention and invoking a realistic sense of beauty and complexity. An example of virtual scenes with trees is shown in Figure 1, where the high trees and small bushes decorate the environment and make it realistic. The trees have complicated and hierarchical structures, composed of a variety of branches including thick trunks, multiple primary branches, and many slim twigs. For computer modelers, the synthesis of the numerous branches is a tedious work. In addition, different species of trees have different features such as branching patterns, branch shapes, and branch distributions. Some trees, such as spruce trees (as shown in the left of Figure 2), have straight branches and side primary branches attach to a straight trunk; some trees, such as Japanese maple trees (as shown in the right of Figure 2), have multiple primary branches without a distinct central trunk and the branches are highly crooked. How to efficiently synthesize 3D trees with controllability of a variety of branch features is also a challenge for computer modelers.

This paper presents a method to synthesize 3D tree models with effective enhancement of tree features. The method is based on the idea of path planning introduced in our earlier work (Xu and Mould, 2007): a graph is built by placing graph nodes uniformly in a 2D plane or a 3D space, and edges connect nodes and are set with specified weights. A path between two nodes is defined as those successive edges that connect one node to the other, whose cost is the sum of the weights of its constituent edges,

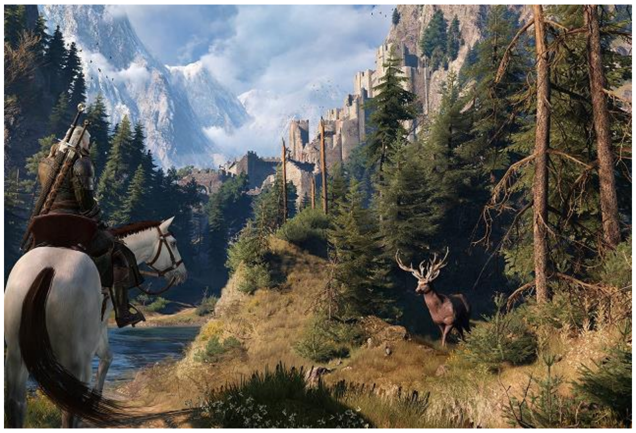

Fig. 1. A virtual world with trees in the game "*The Witcher 3*". Image from Flickr.com.

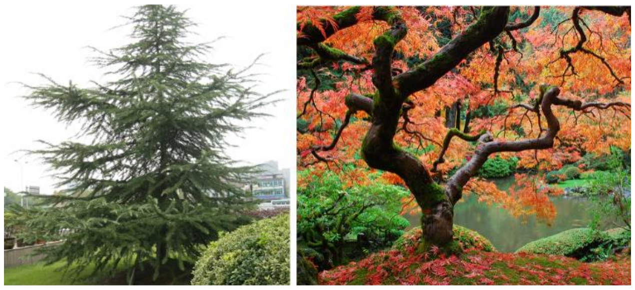

Fig. 2. Two real trees. Left: a spruce tree; right: a Japanese maple tree. Images from Flickr.com.

and the least-cost path is the one which has the minimum cost among all paths. By finding the least-cost paths connecting a node (which represents the tree root) in the lower region of the graph and multiple nodes (which represent branch ends) in the upper region of the graph, we can get a collection of least-cost paths that resemble a branching tree structure where each individual least-cost path is a tree branch. In this paper the technique is refined by involving a graph modification method, which can effectively enhance the features of resulting 3D tree models. To be specific, edges in a few regions of the weighted graph are disconnected, thus preventing the least-cost paths from crossing the regions. As a result, the synthesized tree branches will avoid the regions and demonstrate crooked shapes, making the tree appearance more irregular. In addition, the size and the distribution of the regions can contribute to create different scales of the irregularities, such as big gaps between branches and small crookedness along individual branches. These are helpful to simulate the corresponding features that can be often observed in real trees.

This paper makes the following main contributions.

- First, it proposes a method of modifying the weighted graph by disconnecting edges in specified regions. The prior methods (Xu and Mould, 2007; Xu and Mould, 2015) do not have an effective way to control the branch irregularities; the individual branches are generally smooth. The method of disconnecting edges in specified regions of a graph can effectively enhance the irregular features of branches and make the branch shape more crooked and realistic.
- Second, the exploration of varying the size and distribution of regions for disconnected edges contributes a mechanism of building different scales of branch irregularities. Large sizes of compactly placed regions contribute to create large scale of branch irregularities such as gaps between branch clusters; small sized regions contribute to build fine crooked branch shapes. This feature controlling method is novel and effective.

The paper is organized as follows. Following the introduction, the section of background will review related methods in tree modeling. The details of the feature enhancement method will be introduced in the algorithm section. Results will be given and discussed in section 4. The last section includes a conclusion and also proposes the future work.

2 Background

Due to the wide uses of tree models in computer applications such as video games and 3D movies, many methods have been devised for tree synthesis. Scanning is a common method. Based on a real tree, scanning can obtain point clouds of tree data which will then be used to reconstruct a virtual tree structure. Scanning methods (Xu et al., 2007; Li et al., 2013; Livny et al., 2010) can produce high quality tree models with precise descriptions of tree features due to the data directly from real trees. However, because of the occlusion of tree branches and leaves, scanning needs pre-processing and post-processing work for the raw data. The availability and operations of scanning devices are also not easy for novice users (Capizzi, 1982). In addition, scanning requires a real tree as an example, and it is hard to create a 3D virtual tree without an existing tree in the natural world.

Another category of widely used methods is procedural methods. Procedural methods (Ebert, 2005; Prusinkiewicz and Hanan, 1989) use algorithms with a set of parameters to depict the complex details of tree structures. By changing the rules or formulas and parameter settings, procedural methods can automatically generate a large amount of variations of results. For synthesizing complicated structures such as trees, procedural methods have their strength in the data amplification ability – "a few parameters (or a small amount of geometry) magically expand into a large, detailed model" (Ebert, 2005). A famous group of procedural methods for tree modeling is L-systems. L-systems are a grammar that generates strings that can be subsequently interpreted as branching structures such as trees. The basic idea of L-systems is based on rewriting rules. A rewriting rule is composed of strings of symbols. Each symbol represents a graphical operation, such as the following.

F	Move forward a step of length d
+	Turn left by angle δ
−	Turn right by angle δ
[	Push the current state onto a pushdown stack
]	Pop the current state onto a pushdown stack

After applying the replacement rules for a few iterations, the initial structure can be developed into a large complex shape. An example is shown in Figure 3, where the initial vertical line (at iteration 0) can be developed into a branching structure after applying the rewriting rules for 5 iterations.

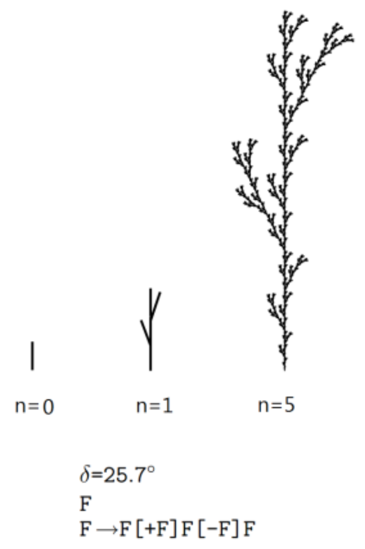

Fig. 3. Structures generated by iterations of rewriting. The example is taken from "*The Geometric Beauty of Plants*" (Prusinkiewicz and Lindenmayer, 1990)

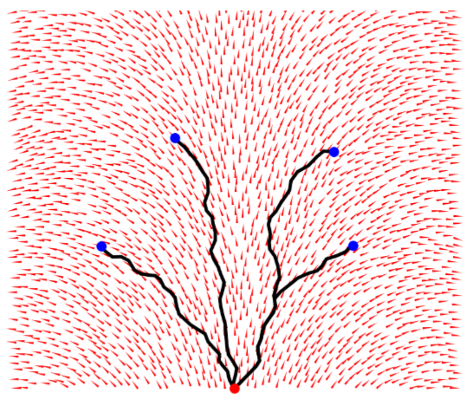

Fig. 4. Least-cost paths in a 2D graph with guiding vectors (Xu and Mould, 2015)

One issue with L-systems is the difficulty to design the rewriting rules to create a desired tree shape. It is not easy to relate the long string of symbols with the resulting tree structure.

The method introduced in this paper is built on our earlier procedural algorithms of path planning (Xu and Mould, 2007; Xu and Mould, 2015). Here we briefly review their basic ideas.

The basic tree modeling mechanism (Xu and Mould, 2007) is based on the idea of path planning. Path planning is the problem of finding the least-cost paths in a weighted graph (Millington and Funge, 2009). By selecting a node in the lower region of the graph and multiple nodes in the upper region of the graph, the least-cost (or shortest) paths connecting each upper node and the lower node form a branching structure. The purpose to select a lower node (which we call the root point) and a few upper nodes (which we call endpoints) is to simulate the tree shape – a root located at the bottom and multiple branches connecting the root and the upper ends. An example of least-cost paths is shown in Figure 4, where the red point is the root point, and the blue points are endpoints. The black paths are least-cost paths that connect the root point and the endpoints.

There are a few elements in the above path planning algorithm that can affect the resulting tree structures: the placement of graph node, the distribution of the root point and the endpoints, and the setting of edge weights. In order to make the coarse branching structure resemble real tree structures, the challenges lie in the control of branching patterns and intermediate scale architectures through the above elements, which are addressed by using guiding vectors (Xu and Mould, 2015). In the method of guiding vectors, a graph is built by connecting nodes scattered in a specified volume (e.g. a cube) with edges. Each graph node has a vector, i.e. guiding vector, used to set the weights of the edges that connect the node and its neighboring nodes. Based on a large or small deviation of the edge direction to the guiding vector direction, the weight of an edge can be expensive or cheap, thus leading to the resulting least-cost paths to follow the guiding vector directions. The least-cost paths in a graph with guiding vectors at each node are shown in Figure 4, where the red arrows are guiding vectors. The guiding vectors are set in the process of applying Dijkstra's algorithm (Dijkstra, 1959) in a brushfire way. The Dijkstra's algorithm is an algorithm to compute the least-cost paths from a source node (i.e. the root node in our case) to other nodes in a weighted graph, by proceeding from the source node to reach other graph nodes and computing the path cost from each graph node to the root node. When a graph node is visited, its guiding vector is set as a rotation from its parent node's guiding vector, leading to the smooth flow of guiding vector directions. Because the method uses the positions of endpoints to control the global tree shape and guiding vectors to vary the branching patterns, which are based on the user's geometric understanding of the target tree shape rather than abstract symbols or rules, it is straightforward to create a desired tree structure.

A complicated tree can be synthesized by adding more branches through iterations. A tree example is shown in Figure 5. The left is a tree skeleton and the right is a complete tree after adding more branches to the previous structure. The tree shows a reasonable structure and details. However, due to the setting of guiding vectors is based on an incremental rotation and the flow of guiding vectors is smooth, the branch shapes are generally smooth and lack natural irregular crookedness. Another observation of the tree model is that its branches are generally evenly distributed, however the distribution of branches in natural trees are often more heterogeneous – they have big gaps between clusters of branches. Although the guiding vectors can affect the branch shapes, making branches possess the above features through setting guiding vectors is difficult: unlike incremental rotations, making the flow of guiding vectors crooked requires a much more tedious and careful setting mechanism. In this paper, these target features are obtained by modifying the graph rather than setting guiding vectors, which can effectively address the problem. More details are introduced next.

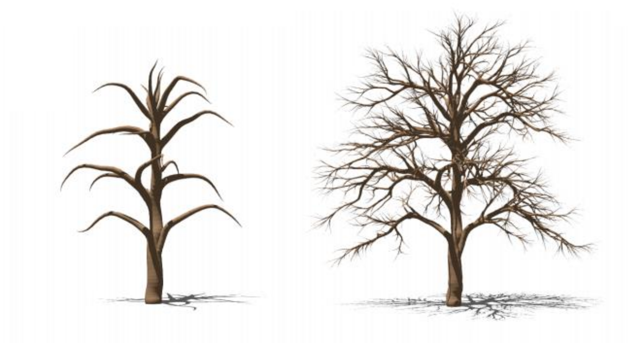

Fig. 5. A tree model created in the earlier work (Xu and Mould, 2015).

3 Algorithm

The algorithm for feature enhancement is based on the influence of edge connections to the resulting least-cost path shapes. To be specific, since a least-cost path is composed of edges in the graph, if the graph has some holes where edges are disconnected, the path between two nodes will not cross the holes and take a direct route, but tends to take available edges in the vicinity of holes. As the result, the path shape is less smooth but becomes more irregular. The idea is illustrated in Figure 6. In this figure, the left shows a graph composed of nodes (in blue) and edges (in black). A least-cost path (in red) connects two points from P_1 and P_2. On the right side, the edges that connect nodes inside the green circle and between inner nodes and outer nodes are disconnected. Because there is no edge connecting the nodes inside the hole (the green circle) and other nodes, the least-cost path that connect P_1 and P_2 will not take the original route that crosses the hole region, but takes the edges close to the hole and demonstrates a more irregular shape than the left path. Although there are other optional paths that can connect P_1 and P_2, the least-cost path tends to take the shortest route, i.e. the edges tightly follow the contour of the hole, making it feasible to control the path irregularities through the properties of holes.

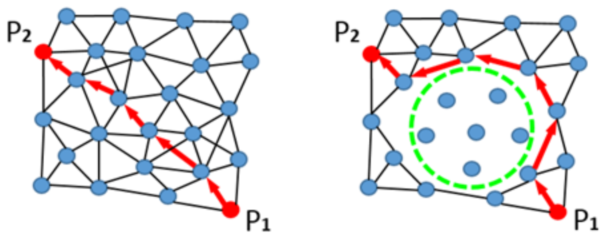

Fig. 6. Left: a generally smooth path connecting two nodes in a graph without holes; Right: a more irregular path in a graph with a hole (the green circle).

The above described algorithm involves two further questions: how to decide the placement of a hole in the graph and how to decide the size of a hole. The following introduces the explorations with respect to the above questions.

Since the least-cost path does not take in-hole edges, through controlling the placement and sizes of holes, we can change the features of resulting least-cost paths. The holes in a 2D graph (as shown in Figure 6 – 9) are defined as circles, and in a 3D graph (as those used in Figure 10 and 11) are spheres. Edges are disconnected if they locate inside a hole or cross the boundary of a hole. The purpose of doing so is to prevent the least-cost path to take the edges in the hole. The positions of holes are decided according to the Poisson Disc distribution (Fiume, 1995), where holes must keep a minimum distance to each other. This decision helps to control the distribution of holes and the shapes of the resulting least-cost paths.

The principle to decide the size and the number of holes is not that strict. Since this work does not aim at a precise simulation of a specific tree branch shape, a hole at a radius of 3 nodes will not have big difference with a radius of 4 nodes or 5 nodes. We intend to achieve a general enhancement of branch features that are absent in the previous work, so the possession of crookedness and branch gaps are the main concerns. According to our experience, depending on the target branch features, the size of holes can be set in a flexible range: in a graph of 150,000 nodes, holes at a radius of 2-6 nodes can be used for small-scale crookedness, and 10-20 nodes for large-scale crookedness. Further adjustment is possible for specific trees. Fortunately due to the benefits of procedural methods, adjusting the hole sizes does not require extra effort but changing a parameter setting. Changing the hole size and re-running the program take less than 10 seconds for most tree structures. The number of holes in the graph is decided by the minimum distance in the Poisson Disc distribution and the radii of holes. The principle to follow is to make the holes tightly and evenly filled the graph but also leave narrow non-hole regions (at a width of 2-5 nodes) for least-cost paths.

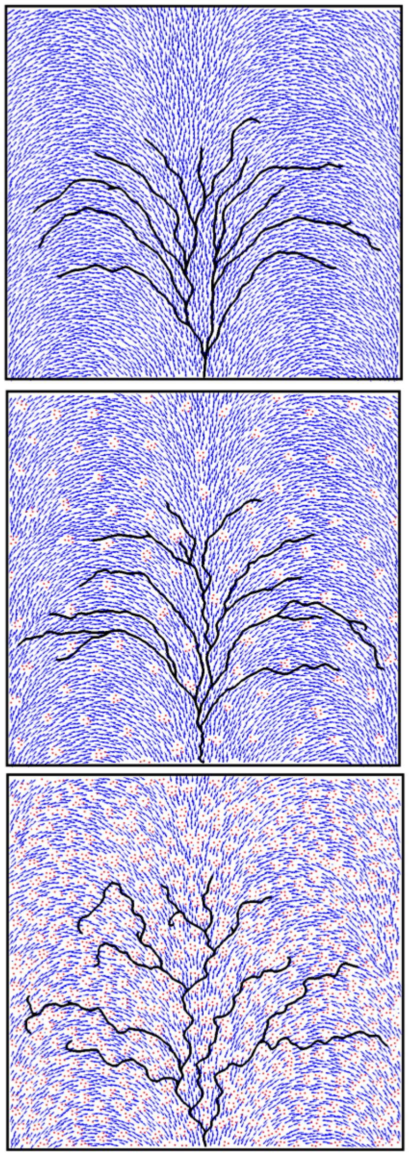
Fig. 7. Top: a tree-like structure in a graph without holes; middle: the graph has sparse holes; bottom: the graph with denser holes

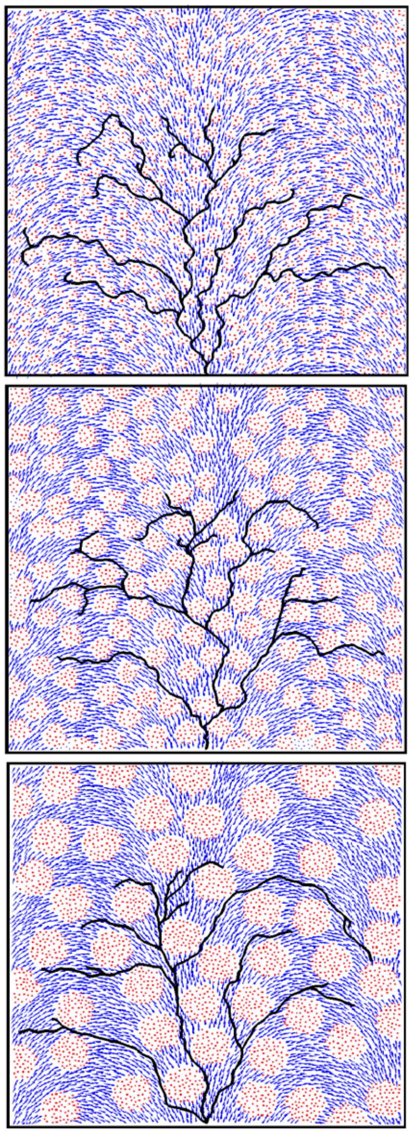
Fig. 8. From top to bottom: least-cost paths obtained in graphs with small sized holes, intermediate sized holes, and large sized holes.

In Figure 6 – 9 we use least-cost paths in a 2D graph to illustrate the algorithm for the ease of viewing the path shapes and the holes, which in 3D may be difficult because of node shading and occlusions. Figure 7 shows how different densities of holes affect the resulting path shapes in a 2D graph. In the top, least-cost paths in a graph without holes tend to follow the flow of guiding vectors (blue arrows). In the middle, the sparsely distributed holes do not have much influence on the path shapes, because in addition to the edges close to the holes there are still plenty of other edges in the non-hole regions. It is possible that the least-cost paths do not take the edges around the holes but in a more direct route in the big non-hole regions. In the lower, the graph has many holes and the resulting paths demonstrate the crooked shapes, due to the very few available edges in the non-hole regions. A least-cost path must avoid many holes to take available edges in order to reach the endpoint, leading to the irregular path feature.

While the densely distributed holes contribute to the feature of path irregularity, the sizes of holes affect the scale of irregular features. Figure 8 shows three examples. The top graph has dense small holes in which edges are disconnected. The paths have small-scale irregularities. The holes in the middle graph are larger, leading to larger crooked path shapes. The paths in the bottom graph show large-scale features – big gaps between some paths, and large turning path shape due to the path traveling around big holes. Figure 9 shows the use of a combination of small holes and big holes. The graph in the right has 12 big holes and 240 small holes whose radii are only 1/5 of the big holes. Compared with the paths in the left graph without any holes, the right paths demonstrate both big scale irregularities such as gaps between paths and crookedness of individual paths.

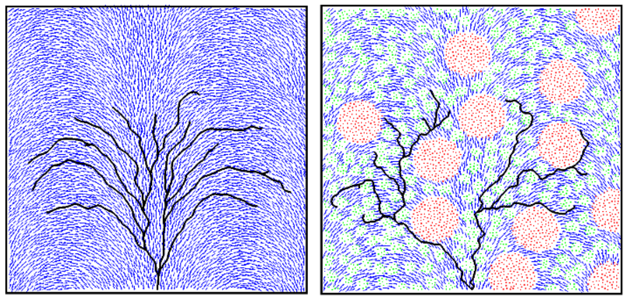
Fig. 9. A **2D tree structure with (left) and without (right) feature enhancement.**

4 Results and Discussion

The feature enhancement algorithm introduced above has been applied to create 3D tree models, shown in the following figures. The evaluation of the results is based on the side by side comparison with a tree model without the feature enhancement. This comparison is built on visual inspection, which has been used for evaluating tree modeling since the earliest synthetic trees in 1980s. It is worth mentioning that in the area of tree modeling there does not exist an objective function or standard for evaluating the quality of tree models. Since the main applications of the tree models are for entertainment purposes, not for precise duplication or detailed simulation, the visual inspection of the presence/absence of target features can be used to evaluate the quality of results and the effectiveness of algorithms. Actually the method of visual inspection and comparison has been widely used by many researchers (Stava et al., 1998; Runions, 2008; Xu and Mould, 2012). Here it is used to evaluate the results in this paper. Figure 10 shows a 3D tree with feature enhancement (on the left side) and a tree created using the previous method (Xu and Mould, 2015) without feature enhancement (on the right). The top row shows the tree skeleton

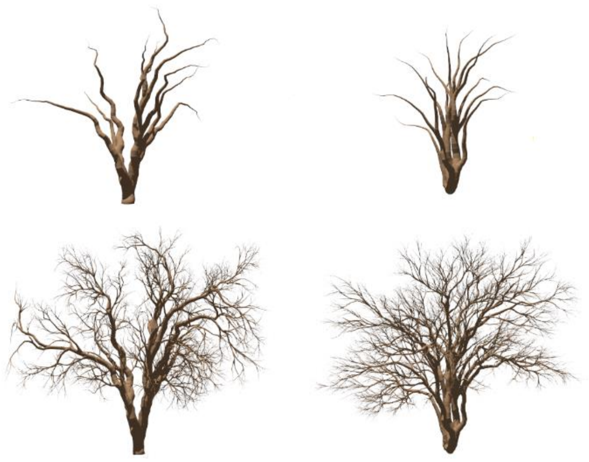
Fig. 10. Left: a tree with feature enhancement; right: a tree without feature enhancement from the early work of guiding vectors (Xu and Mould, 2015).

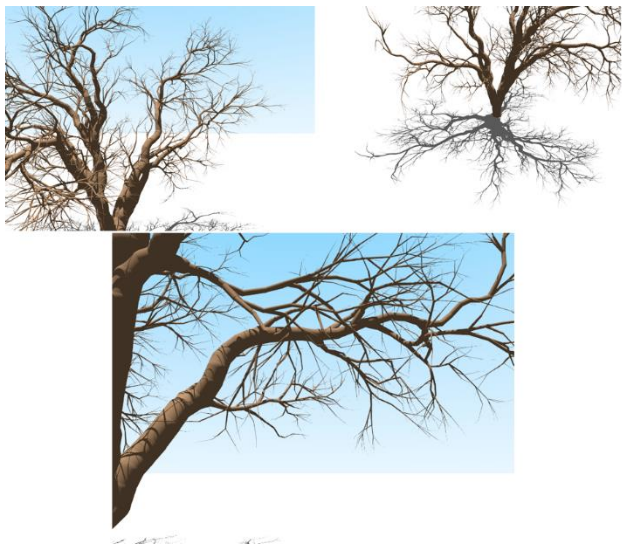
Fig. 11. A 3D tree with enhanced features viewed from three different directions.

and the lower row shows the trees after adding twigs. The left tree has more crooked branches which enhance the natural irregularities. The big gap in the central region of the left tree also makes the whole tree structure different from the right regular structure. This feature is hard for the previous method of guiding vectors. While the right tree shows more homogeneous properties, the left tree possesses more heterogeneous features. Figure 11 shows a 3D tree viewed from three different directions. A close view of individual branches is in the lower image, where due to the feature enhancement, the branch demonstrates a crooked shape that spreads from the lower left to the upper right and curves upwards.

5 Conclusion and Future Work

This paper presents a feature enhancement algorithm for 3D tree synthesis. The algorithm is built on the idea of path planning and tree synthesis with guiding vectors. In this algorithm, a number of holes are placed in the graph, and edges in the holes are disconnected thus prevent the least-cost paths to cross the regions of holes. Due to the limited available edges in narrow non-hole regions, the least-cost paths are forced to take edges in non-hole regions especially those are close to the contour of holes, leading to crooked path shapes. The hole size contributes to control the scale of this irregularity: large holes make branch gaps and small holes make the fine crookedness.

Compared with the previous work, the algorithm can effectively enhance the branch features and make branch shapes more irregular. The use of holes, including the hole distribution and hole sizing, contribute to enhance the heterogeneity of branches especially the natural gaps between branch clusters. These features are helpful to create irregular 3D trees which are needed in movie scenes or game animations especially for special scenes where irregular crooked trees (such as Halloween spooky trees) are required. The algorithm also has its limitations. The current combination of large holes and small holes is tentative. A more sophisticated and systematic investigation of the proportion should be performed in the future work. In addition, the distribution of holes is based on Poisson Disc distribution, which however does not take the overall tree structure into consideration. Different regions of the tree structure can be treated differently, with different densities of holes. In the future work, more interactive controls, such as sketch-based methods, will be explored to make the feature enhancement more controllable.

Acknowledgements

Thanks Dr. David Mould for valuable discussions about graph modification ideas. This work has been supported by the Organized Research and Creative Activities (ORCA) Program of University of Houston-Downtown.

Conflict of Interest: none declared.

References

A. Runions. Modeling biological patterns using the space colonization algorithm, 2008

D. S.Ebert, F. Musgrave, D. Peachey, K. Perlin, and S. Worley. Texturing and Modeling, A Procedural Approach (Third Edition). Elsevier Science, USA, 2005.

E. Fiume. An Introduction to Scientific, Symbolic, and Graphical Computation. A K Peters/CRC Press, 1995.

E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959.

H. Xu, N. Gossett, and B. Chen. Knowledge and heuristic-based modeling of laser-scanned trees. ACM Trans. Graph., 26, 2007.

I. Millington and J. Funge. Artificial Intelligence for Games, Second Edition. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2009.

L. Xu and D. Mould. Modeling dendritic shapes - using path planning. In GRAPP (GM/R), pages 29–36, 2007.

L. Xu and D. Mould. Synthetic tree models from iterated discrete graphs. In Proceedings of Graphics Interface, pages 149–156, 2012.

L. Xu and D. Mould. Procedural tree modeling with guiding vectors. Computer Graphics Forum, 34(7), 2015.

O. Stava, S. Pirk, J. Kratt, B. Chen, R. Mch, O. Deussen, and B. Benes. In search of the right abstraction: The synergy between art, science, and information technology in the modeling of natural phenomena. Art @ Science, pages 60–68, 1998.

P. Prusinkiewicz and J. Hanan. Lindenmayer systems, fractals and plants. Lecture Notes on Biomathematics, 1989.

P. Prusinkiewicz and A. Lindenmayer. The Algorithmic Beauty of Plants. SpringerVerlag, New York, 1990.

T. Capizzi. 3D Modeling And Texture Mapping. Premier Press, 1982.

Y. Li, X. Fan, N. J. Mitra, D. Chamovitz, D. Cohen-Or, and B. Chen. Analyzing growing plants from 4d point cloud data. ACM Trans. Graph., 32(6):157:1–157:10, 2013.

Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana. Automatic reconstruction of tree skeletal structures from point clouds. ACM Trans. Graph., 29(6):151:1– 151:8, 2010.