SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •
Special Focus

March 2013, Vol. 56 032105:1–032105:12
doi: 10.1007/s11432-013-4790-0

Efficient polygonization of tree trunks modeled by convolution surfaces

ZHU XiaoQiang, GUO XueKun & JIN XiaoGang*

State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou 310058*, China*

Received August 22, 2012; accepted October 16, 2012

Abstract We present an efficient polygonization approach for tree trunks modeled by line skeleton-based convolution surfaces. A quad-dominated non-convex bounding polyhedron is firstly created along the skeleton, which is then tetrahedralized and subdivided into the pre-defined resolution. After that, the iso-surface within each tetrahedron is extracted using marching tetrahedra. Our algorithm can generate polygons with adaptive edge lengths according to the thickness of the trunk. In addition, we present an efficient CUDA-based parallel algorithm utilizing the high parallelism of the tetrahedron subdivision, the potential field calculation, and the iso-surface extraction.

Keywords trunk modeling, skeleton-based modeling, convolution surface, tetrahedral subdivision, CUDA

Citation Zhu X Q, Guo X K, Jin X G. Efficient polygonization of tree trunks modeled by convolution surfaces. Sci China Inf Sci, 2013, 56: 032105(12), doi: 10.1007/s11432-013-4790-0

1 Introduction

Realistic modeling and rendering of plants makes virtual scenes more photo-realistic, which has always been a hot topic in the computer graphics community. Trees are the most popular adopted plants for their superior feature of huge size. There is limited bandwidth for people to gather information through the sense of vision, so observers usually focus on the outline of the structure instead of minor details when watching a tree. Moreover, the outline of a tree depends on its trunk and branch distribution, which is the research focus of the paper.

Many methods for tree modeling have been proposed, which can be mainly classified into two categories: virtual tree modeling [1,2] and reconstruction of real trees [3–6]. Virtual tree modeling has been developed for decades and it is still being studied and developed. For example, both the grammar-based procedural modeling in [1] and the sketch-based modeling in [2] fall into the first category. The early L-system [7] generates trees by repetitive application of a small set of rules to an initial structure to create rather complex results. The modeling process can be controlled at a higher level in the more recent grammar-based modeling method [8]. Although complex trees can be modeled using such approaches by professionals, it will be too abstract for novices and non-professionals because of the non-intuitive grammars and rules. Therefore, more and more works on interactive sketch-based tree modeling emerge

*Corresponding author (email: jin@cad.zju.edu.cn)

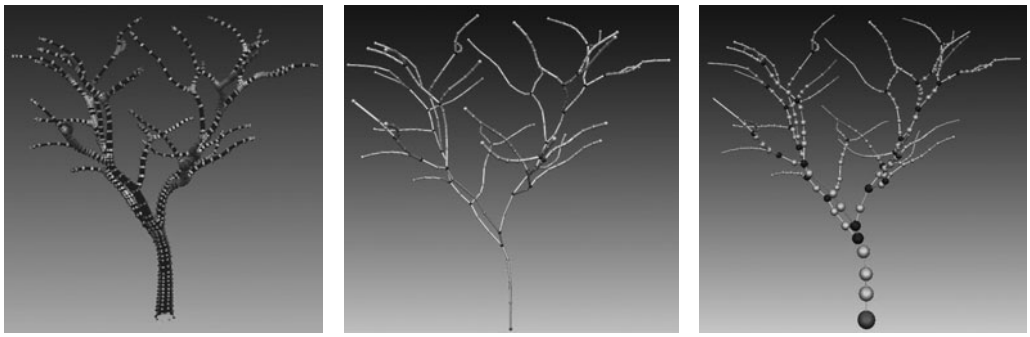
in recent years [9–11]. The increasing passion for photo-realistic perfection gives an impetus to more and more researches on reconstruction from real data. One of the most important approaches is image-based modeling. Reche-Martinez et al. [12] propose an algorithm which purely depends on input images. The scheme in [3] combines the input image and sketches to reconstruct the 3D trees in the original image. Similarly, the image-based reconstruction of Tan [13,14] also allows for some user intervenes. With the recent progress in 3D laser scanning technology, tree reconstruction from point clouds attracts more and more attention. In [5,6], the trunks and branches are firstly reconstructed from point clouds before adding the leaves. Bucksch et al. [15,16] firstly group scanned point clouds into clusters, and then link them together with adjacent ones to produce the final skeletons. Since skeleton is an excellent abstract of a shape and it is easy and convenient to edit, our algorithm is based on the skeleton-based implicit surfaces. The adopted skeletons of tree trunks can be extracted from point clouds and polygonal meshes (see, for example, [5,6,17,18]).

Trees have branches and line skeletons [19], so we represent them using convolution surface [20] because of its advantages of compact skeleton representation, smooth surface, shape suggestion, superposition, and well-blending. RBF-based implicit surface can also express shapes with complex topology [21]. However, it is not easy to control the RBF-based method compared with the skeleton-based convolution surface. A convolution surface is an iso-surface in a scalar field defined by convolving a skeleton with a potential kernel function. Theoretically, any geometric primitives can be used as skeletons to produce convolution surfaces. However, the existence of analytical solutions for convolution surfaces depends on both the skeleton primitives and the kernel functions. Closed-form solutions exist only for limited skeletons such as points, line segments, triangles, arcs, and quadratic spline curves [22]. Although complex skeletons can be utilized to create rich convolution surfaces [23–29], line skeletons are preferred because of its efficient computation and simple editing, which fits our tree modeling requirement.

Convolution surface is a commonly-used implicit surfaces. Its prevalent rendering method is to extract the polygonal iso-surface and then to render the resulting polygon mesh by making use of the hardware support for fast polygon rendering. Extraction of the iso-surface has been extensively studied. In marching cubes algorithm [30], a 3D space is firstly made partitioned into sub-hexahedra, and then triangular iso-surfaces are extracted according to an iso-surface threshold and the potential field values at the hexahedral vertices. With the development of GPU, parallel marching cubes algorithms have been developed.[1)] However, the extracted mesh cannot be guaranteed to be topologically consistent with the original iso-surface, which may result in ambiguity in some hexahedra. One reason for that is that the partition space in marching cubes must be a hexahedral structure. As a substitute, marching tetrahedra [31,32] can overcome above-mentioned drawbacks. Any shape can be decomposed into tetrahedra which are the most fundamental voxels. No topological ambiguities exist within a tetrahedron. Therefore, we take tetrahedra as the sampling structure for the iso-surface extraction. A challenging task is how to create adaptive voxel sizes according to the thickness of trunks so that large triangles are extracted for thick trunks and small triangles are produced for thin parts. Although some adaptive algorithms have been developed [33–35], most of them have to partition the space adaptively. As a result, they suffer from complex topological relationship between different levels of resolution, which will bring about troubles for parallel computation on the GPU. In this paper, a non-convex bounding polyhedron based on the skeletons of tree trunks is created whose size will vary adaptively with the thickness of trunks. The sizes of the tetrahedralized and subdivided tetrahedra are also self-adaptive. Therefore, our approach avoids space partition and can generate tetrahedra tightly enclosing the object. In the meantime, our approach significantly reduces both the time and the memory required. Our approach does not need to maintain the complex topological consistence relationship for adaptive space partition. As a result, our scheme is well-suited to parallel implementation on the GPU.

Our paper makes the following contributions for the tree modeling: 1) We have developed a new algorithm for the polygonization of tree trunks based on convolution surfaces. A tight bounding polyhedron is created to limit the extraction space within an effective range. The edge lengths of the bounding polyhedra and the subdivided tetrahedra are adaptive to the trunk thickness. As a result, our approach

1) http://developer.nvidia.com/cuda-toolkit-32-downloads.



(a) (b) (c)

Figure 1 (a) A set of point cloud of an input tree model; (b) the extracted skeletons; (c) the thickness information.

neither misses too small twigs nor produces too many triangles for thick trunks. That is, the generated triangles are dependent on the curvatures of the iso-surface. 2) We have designed a CUDA-based parallel scheme to fully exploit the parallelism of the tetrahedral subdivision, the potential field value computation, and the iso-surface extraction. The proposed algorithm is of high performance because the most time-consuming stages are executed in parallel on the GPU.

The remainder of the paper is organized as follows. After introducing the definition of convolution surfaces based on tree trunk skeletons in Section 2, we describe both the creation of the skeleton-based bounding polyhedron and its tetrahedral subdivision in Section 3. In Section 4, the extraction and the rendering of tree trunk skeleton-based convolution surfaces are presented. Experimental results and examples are presented in Section 5. Our paper ends with the conclusion section.

2 Tree trunk skeleton-based convolution surfaces

Branch structures such as vasculature and tree trunks can be conveniently represented by skeleton-based convolution surfaces, which produce smooth surfaces and pleasing blending. Therefore, line skeleton-based convolution surfaces are employed to create tree trunks in this paper. This section mainly introduces the generation of tree trunk skeletons and the definition of such trunk skeleton-based convolution surfaces.

2.1 Generation of tree trunk skeleton

Tree trunks have natural line skeletons because of their cylinder-like shapes. Our approach is based on provided trunk skeletons, which can be acquired through existing approaches, such as: 1) inputs from user interfaces of modeling packages [36]; 2) skeleton extraction from exiting tree models, including point cloud-based [5,18] and mesh-based [17] skeleton extraction. The resulting skeletons usually contain radii (distances between skeletons and trunk surfaces) at skeletal nodes. Figure 1 illustrates the extracted skeletons and the corresponding radius information.

2.2 Reconstruction using convolution surfaces

2.2.1 *Definition of convolution surfaces*

A convolution surface is an iso-surface in 3D scalar field defined by convolving a skeleton with a potential kernel function, which can be mathematically defined as

$$S = \left\{ (x, y, z) \left| \sum_{i=1}^{n} \lambda_i F_i (x, y, z) - T = 0 \right. \right\}, \tag{1}$$

where $F_i(x, y, z)$ is the field function of the ith skeleton segment, λ_i is the weight of field contribution from the ith skeleton segment, and T is the threshold value for the convolution surface.

Let $P(x, y, z)$ be a point in $\mathbb{R}^3$, and $g : \mathbb{R}^3 \to \mathbb{R}$ be a function representing a geometric skeleton V:

$$g(P) = \begin{cases} 1, & P \in V, \\ 0, & P \notin V. \end{cases} \tag{2}$$

Let Q be a point in the skeleton V, and let $f : \mathbb{R}^3 \to \mathbb{R}$ be a potential function generated by a single point in V. Then the total field contribution at P from the skeleton V is defined as the convolution of f and g:

$$F(P) = \int_V g(Q) f(P - Q) \, \mathrm{d}V = (f \otimes g)(P). \tag{3}$$

Therefore, f is usually called the convolution kernel. In this paper, we adopt convolution surfaces to create tree trunks. The independent calculation can be guaranteed by the superposition property of convolution surfaces:

$$f \otimes (g_1 + g_2) = (f \otimes g_1) + (f \otimes g_2), \tag{4}$$

which indicates that the total field contributions at a point P from multiple skeleton segments can be simply calculated as the sum of the contribution from each skeleton segment. Therefore, we can focus on the field calculation for a single skeleton rather than the fusion and branches for multiple skeletons.

There are several choices for convolution kernels f [22]. Here, we adopt the Cauchy kernel developed in [28]:

$$f(P - Q) = \frac{1}{(1 + s^2 r^2)^2}, \tag{5}$$

where $r = \|P - Q\|$, and s is a control parameter for adjusting the width of the kernel function.

2.2.2 *Approximation with convolution surfaces*

We use the following notations:
$\boldsymbol{L} = \{L_1, \ldots, L_n\}$ are line skeletons;
$F^{ij} = F_{L_i}(p_j)$ is the field value contributed by the ith skeleton at the jth constraint position;
$\boldsymbol{F^i} = \left[F^{i1}, \ldots, F^{im}\right]^{\mathrm{T}}$ is a vector of field values produced by the ith skeleton at all constraint positions;
$\boldsymbol{F} = \left[\boldsymbol{F}^1, \ldots, \boldsymbol{F}^n\right]$ are field values produced by all skeletons at all constraint positions;
$\boldsymbol{T} = [T, \ldots, T]$ is a vector of threshold values for the convolution surfaces.

To solve the convolution surface approximation with given constraint positions $\{p_1, \ldots, p_m\}$, the approach of LS (least squares) in [37,38] is employed for the unknown weights of field contributions from each skeleton $\boldsymbol{\Lambda} = [\lambda_1, \ldots, \lambda_n]^{\mathrm{T}}$:

$$\min_{\boldsymbol{\Lambda} \geqslant 0} (\boldsymbol{F\Lambda} - \boldsymbol{T})^{\mathrm{T}} (\boldsymbol{F\Lambda} - \boldsymbol{T}), \tag{6}$$

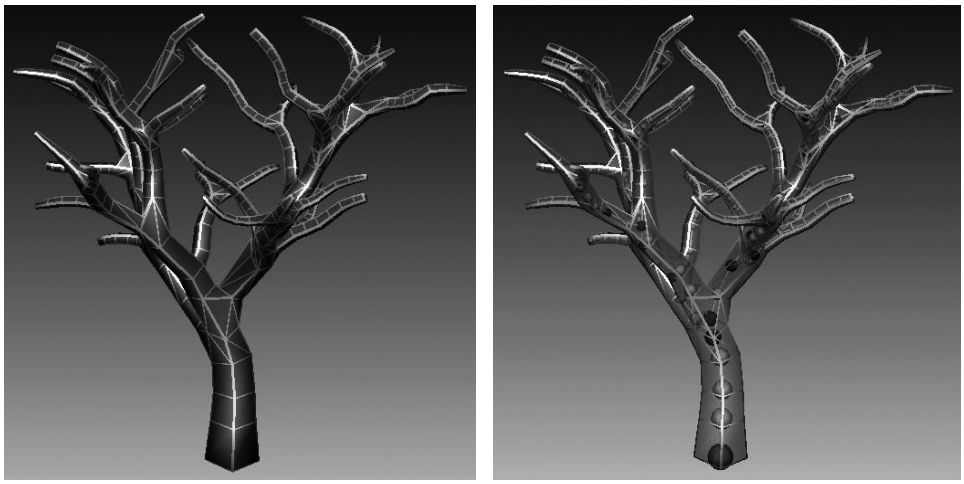
and the NNLS (non-negative least squares) can be iteratively solved to obtain the weight of each line skeleton.

3 Tetrahedral subdivision of skeletal bounding polyhedron

To reduce the computational time and the occupied space during the polygonization of convolution surfaces, the extraction space which will be decomposed into voxels for extracting the iso-surface should be kept as small as possible.

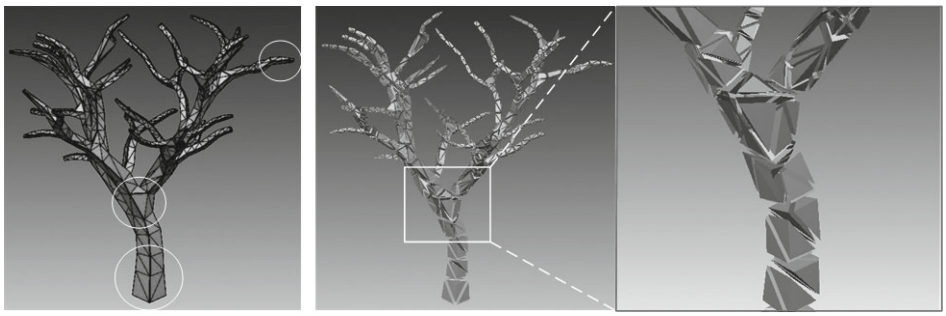
3.1 Skeleton-based bounding polyhedron

From the provided graph-based trunk skeletons and radii, the method in [36] is employed to create a non-convex bounding polyhedron. During the creation of the bounding polyhedron, the skeletal sphere with radius R must be within the polyhedron. To reach the goal, the edge length of the cross-section at each skeletal node should be greater than two times of the skeletal radius (the edge length of the circumscribed



(a) (b)

Figure 2 (a) The bounding polyhedron; (b) the embedded skeleton.



(a) (b)

Figure 3 (a) The original tetrahedralized bounding polyhedron; (b) the offset tetrahedrons.

square of the circle with radius R), and we use three times of the radius in our implementation. However, the process of merging triangles into quadrilaterals described in [36] is avoided in our approach because it becomes unnecessary for our next tetrahedralization step. The generated bounding polyhedron is shown in Figure 2. From the figure, it is obvious that the lengths of the polyhedral edges are proportional to the radii at skeletal nodes.

3.2 Tetrahedral decomposition and subdivision

3.2.1 *Tetrahedral decomposition*

Several open source libraries for tetrahedral decomposition have been developed such as CGAL (Computational Geometry Algorithms Library),[2)] NetGen,[3)] and TetGen.[4)] In our approach, we employ the TetGen library because of its efficiency and no new point insertion, which are critical for our use. The tetrahedralization process is performed on the CPU, and the resulting tetrahedra are transferred into the GPU for the next parallel subdivision. Figure 3 illustrates the tetrahedralized elements and it shows that the sizes of the generated tetrahedra are proportional to the radii of the skeletal spheres.

3.2.2 *Tetrahedral subdivision*

The tetrahedral subdivision step is necessary because the initial tetrahedralized results are too coarse for the direct iso-surface extraction. The tetrahedral subdivisions involved in NetGen and TegGen are all

2) CGAL. http://www.cgal.org/.
3) NetGen. http://www.hpfem.jku.at/netgen/.
4) TetGen. http://wias-berlin.de/software/tetgen/.

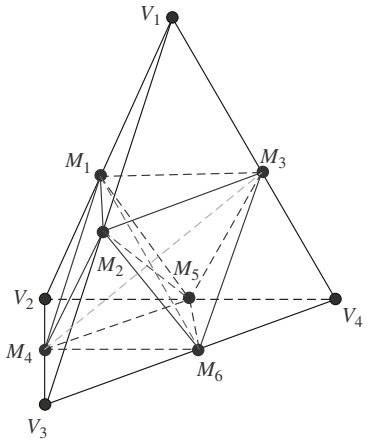


Figure 4 Schema of the tetrahedral subdivision.

implemented on the CPU, and the parallelization is not taken into consideration. In our approach, we have designed a parallel solution based on CUDA to exploit the parallelism in the tetrahedral subdivision process, and the output tetrahedra can be directly employed for the following iso-surface extraction on the GPU.

Our CUDA-based parallel tetrahedral subdivision can be divided into two steps:

1) Edge point generation. We have designed a kernel function *__global__ CalcEdgePoints*() which distributes a thread for each tetrahedral edge to calculate the midpoint of the current edge (generated vertices $M_1, \ldots, M_6$ in Figure 4).

2) Sub-tetrahedra generation. In this phase, the kernel function *__global__ SubdivideTetrahedra*() distributes a thread for each tetrahedron to subdivide the current tetrahedron into eight sub-tetrahedra. Taking tetrahedron $V_1V_2V_3V_4$ in Figure 4 as an example, the first four new sub-tetrahedra $V_1M_1M_2M_3$, $V_2M_1M_4M_5$, $V_3M_2M_4M_6$, and $V_4M_3M_5M_6$ are produced by assembling the original vertices V_1, V_2, V_3, V_4 and their directly connected edge points, respectively. After that, the left central region, which is an octahedron $M_1M_2M_3M_5M_4M_6$, will be subdivided into another four sub-tetrahedra based on the shortest diagonal as the splitter. Let $|M_1M_6| = \min\{|M_1M_6|, |M_2M_5|, |M_3M_4|\}$. Then the newly generated sub-tetrahedra from the octahedron are $M_2M_1M_4M_6$, $M_2M_6M_3M_1$, $M_5M_1M_4M_6$, and $M_5M_6M_3M_1$. Other cases can be subdivided similarly.

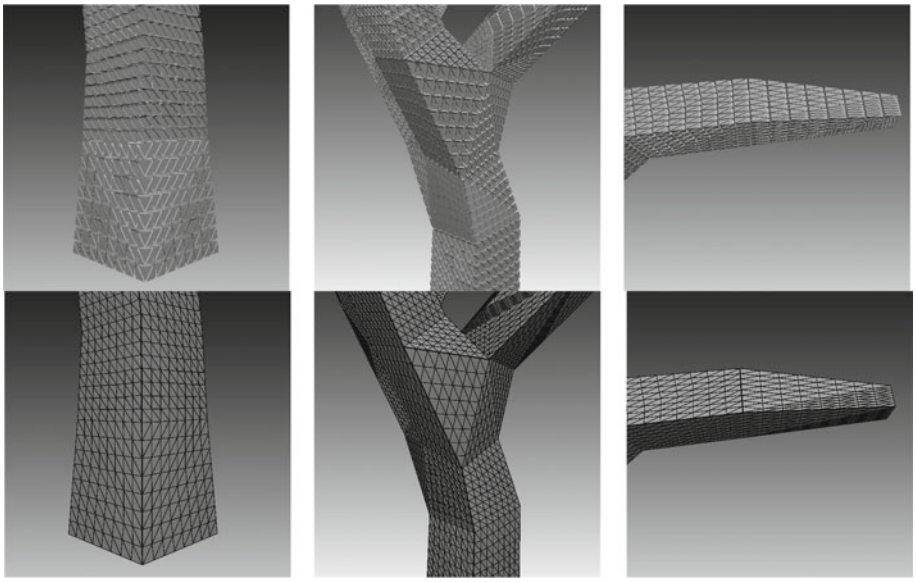
Our experiments show that a three-level subdivision is fine enough. Detailed subdivision results can be found in Figure 5. It is easy to observe that the sizes of the sub-tetrahedra are still proportional to the skeletal radii due to the same depth of subdivision for each tetrahedron.

4 Extraction of convolution surfaces

After the tetrahedral subdivision, the following task is to polygonize the convolution surface. The polygon extraction can be decomposed into two separate stages: 3D scalar field calculation, and triangle extraction according to the iso-surface threshold value and the potential field values at the tetrahedral vertices.

4.1 3D scalar field computation

Before the iso-surface extraction, the field value at each tetrahedral vertex has to be calculated. To reach the goal, the skeletal information should be transferred into the GPU. There is a high-speed cache for texture memory and constant memory for CUDA, so we store skeletal information in texture memory (CUDA has large texture memory) and store the total skeletal count in constant memory (only small constant memory in CUDA for storing data) for frequent accesses during the potential field calculation. Field value calculation usually exhibits prohibitively long computing time on the CPU. Due to the inde-



(a) (b) (c)

Figure 5 Results of the 3rd tetrahedral subdivision (highlighted region in Figure 3(a)). (a) Root; (b) branch; (c) treetop.

pendent field value calculation at each vertex, we employ kernel *__global__ CalcFields*(), which distributes a thread responsible for the potential field value computation at each vertex, for the parallel execution.

4.2 Iso-surface extraction and rendering

For the iso-surface extraction with marching tetrahedra, the iso-surface threshold value is also frequently accessed. Therefore, we store it as a constant variable in video memory. The polygonal iso-surface generation processes within different tetrahedra are independent and have high parallelism. Similar to the tetrahedral subdivision, the CUDA-based parallel extraction can be described as follows:

1) Tetrahedral index generation. In the kernel *__global__ CalcTetrahedralIndice*(), one thread is dispatched for each tetrahedron to count the indices of vertices whose fields are larger than the iso-surface threshold and a 4-bit index *nTetIndex* is formed, each bit corresponding to a vertex. Then both *nTetIndex* and the lookup table *triCntTable* of triangle counts in marching tetrahedra are utilized to search for the count of iso-surface triangles within the current tetrahedron.

2) The function *cudppScan* in the CUDA SDK library cudpp is employed to parallelly compute the prefix-sum of extracted iso-surface triangle counts *nTrisCntPreSum* within all tetrahedra, which is essential for storing extracted triangles in a compact array in parallel.

3) We perform the extraction kernel *__global__ ExtractIsoTriangles*() to generate a thread for each tetrahedron. Each separate execution unit extracts triangles (0, 1 or 2) within the current tetrahedron based on *nTetIndex* in 1) and the lookup table *triExtractTable* of triangle extraction in marching tetrahedra. After that, the extracted triangles are tightly stored in a pre-allocated global array *triArray* in video memory based on the prefix-sum *nTrisCntPreSum* in 2).

To render the extracted triangle mesh, the interoperability between the resources of CUDA and OpenGL is applied to improve the rendering efficiency. That is, the extracted iso-surface vertices and polygonal information by CUDA are stored in VBO (vertex buffer object) of OpenGL, which can be directly rendered efficiently. The results can be seen in Figure 6.

5 Experimental results and comparisons

The created bounding polyhedron using the method in [36] cannot enclose the whole iso-surface at some branches because of the branching trends, which produces holes as shown in Figure 7 (a) and (d). In order

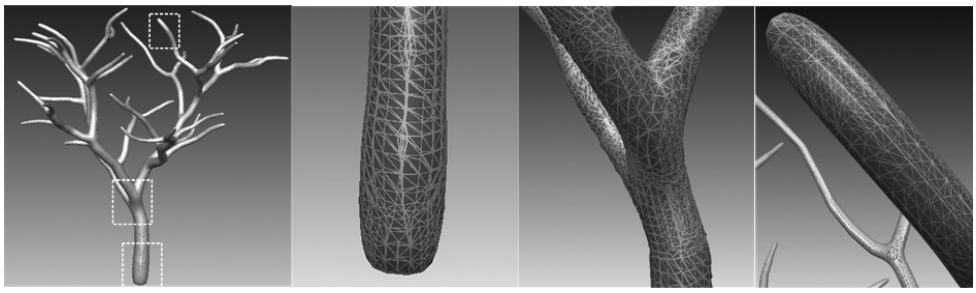


Figure 6 Extracted iso-surfaces.

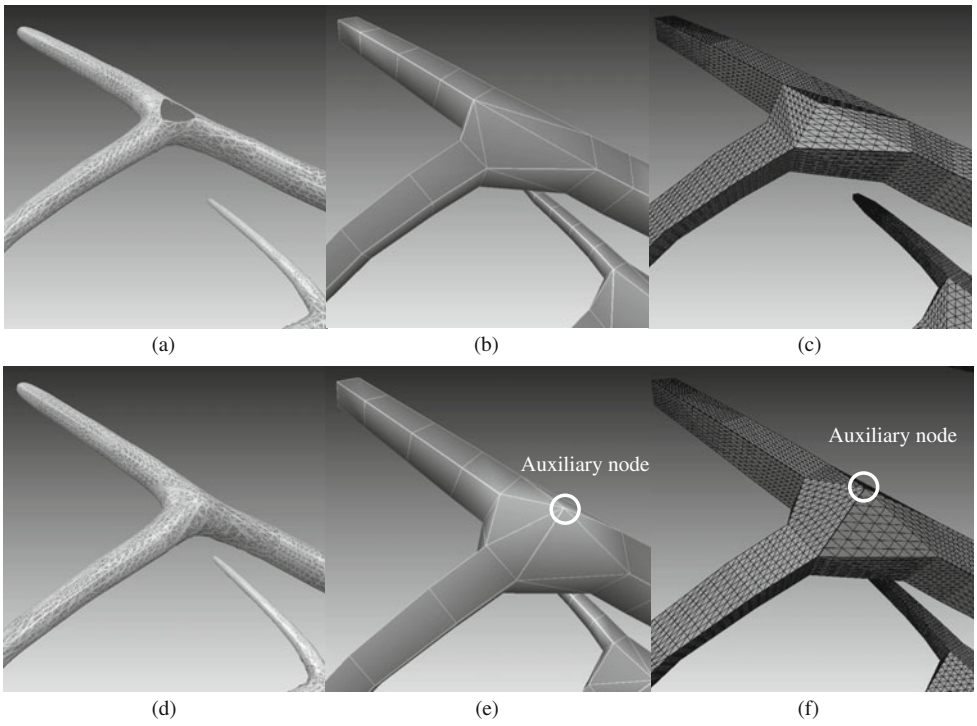


Figure 7 Processing of holes at branches. (a)–(c) Holes of iso-surface, the bounding polyhedron and subdivided tetrahedra at branches; (d)–(f) iso-surface, the bounding polyhedron and subdivided tetrahedra at branches after inserting auxiliary nodes.

Figure 8 Eucalyptus. (a) Skeleton; (b) radii; (c) bounding polyhedron; (d) subdivided tetrahedrons; (e) so-surface.

to prevent such holes, we add an auxiliary point at each side of every sub-tree plane (spanned by two child trees). Let the vector from the branch node to the auxiliary point be $vNodeToAux$. The direction of $vNodeToAux$ is defined as the cross product of two child trees and its length can be obtained by averaging the edge lengths of the child tree skeletal segments. An example of auxiliary point is illustrated in Figure 7 (b), (e) and (c), (f). Figure 8 shows an iso-surface extraction example from an eucalyptus.

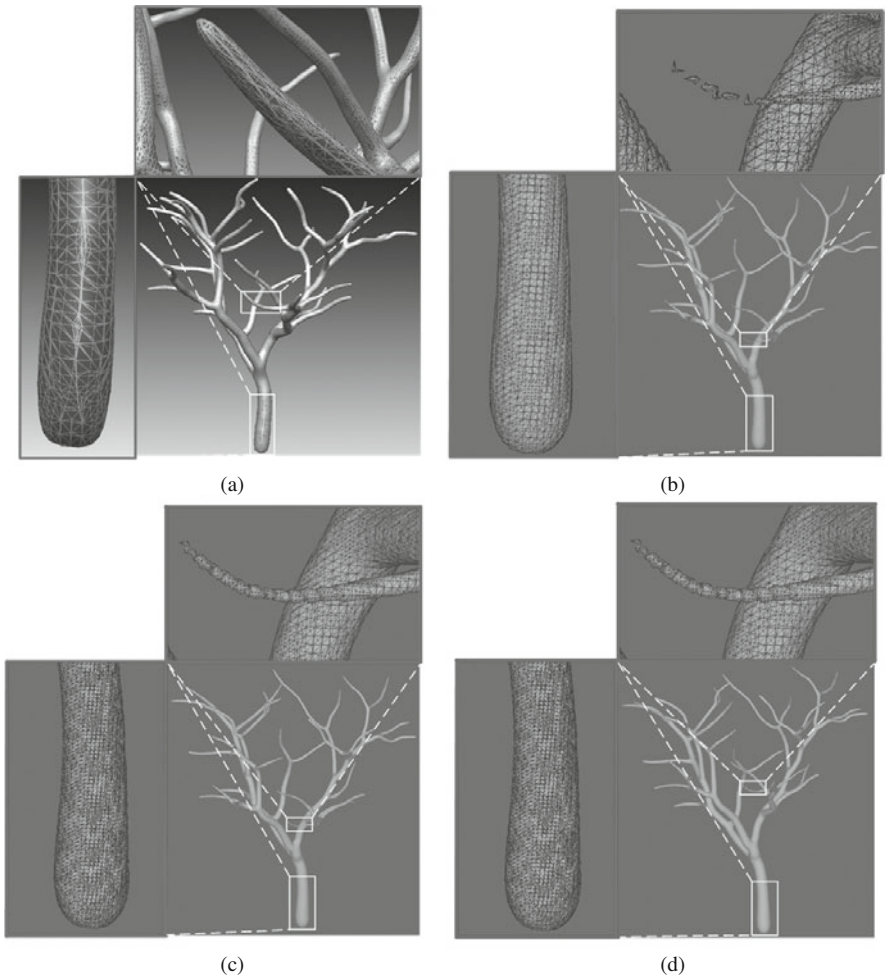


Figure 9 Comparison between our method and marching tetrahedra. (a) 362,740 triangles are extracted using our method; (b) 168,166 triangles are extracted using MT; (c) 364,068 triangles are extracted using MT; (d) 381,284 triangles are extracted using MT.

To evaluate our proposed algorithm, we carry out the comparison between our approach and marching tetrahedra from [39]. The whole experimental procedure is implemented on a desktop computer equipped with Intel Q9400 CPU @2.66 GHz with 4 GB of memory, and the GPU of Nvidia GeForce GTX 260 with an 896 MB of dedicated memory. We employ the CUDA-based parallel architecture on the GPU. Figure 9 illustrates the comparisons for the rhus example shown in Figure 1. The extracted 362,740 triangles are uniformly distributed across the entire tree, instead of too many triangles at thick trunks (with small curvatures) or too few triangles at thin trunks (with large curvatures). To some extent, the extracted triangles are curvature-dependent (see Figure 9(a)). On the contrary, for the marching tetrahedra algorithm [39], if only 168,166 triangles are extracted (with $220 \times 220 \times 220$ cube resolution, smaller number of triangles than ours), nice triangles are produced at thick trunks but fracture emerges at thin trunks because of a too low resolution of cubes to capture thin parts (see Figure 9(b)). If 364,068 triangles are extracted (with $320 \times 320 \times 320$ cube resolution, almost the same number of triangles as ours), too many triangles are produced at thick trunks and tooth-shaped results are produced at thin parts (see Figure 9(c)). Even if the cube resolution is increased to $330 \times 330 \times 330$ (maximum resolution for our system) and 381,284 triangles are generated (more triangles than ours), the part of small twigs cannot produce satisfactory results (see Figure 9(d)).

Apart from producing nicer visual effects, our approach also consumes less computational time than marching tetrahedra. For the polygonization process of tree trunks using convolution surfaces, we can

Table 1 Results using marching tetrahedra. Abbreviations: CubeRes (cube resolution), CubeVerts (cube vertices), ExtrTris (extracted triangles), CalcField (field calculation), ExtrIso (iso-surface extraction)

Models	CubeRes	CubeVerts	ExtrTris	Time (s)		
				CalcField	ExtrIso	Total
Rhus	$220 \times 220 \times 220$	10,648,000	168,166	5.020	0.128	5.148
	$320 \times 320 \times 320$	32,768,000	364,068	14.264	0.411	14.675
	$330 \times 330 \times 330$	35,937,000	381,284	14.307	0.429	14.736
Eucalyptus	$220 \times 220 \times 220$	10,648,000	156,216	3.556	0.130	3.686
	$320 \times 320 \times 320$	35,937,000	243,496	7.025	0.267	7.292
	$330 \times 330 \times 330$	21,952,000	326,968	10.088	0.436	10.524

Table 2 Results using our method. Abbreviations: TetV (tetrahedral vertices), ExtrTris (extracted triangles), BPC (bounding polyhedral creation on the CPU), Tetz (tetrahedralization on the CPU), TetSub (tetrahedral subdivision), CalcField (field calculation), ExtrIso (iso-surface extraction)

Models	Tetrahedra	TetV	ExtrTris	Time (s)					
				BPC	Tetz	TetSub	CalcField	ExtrIso	Total
Rhus	1,056,768	905,228	362,740	0.349	0.084	0.029	0.310	0.018	0.790
Eucalyptus	747,008	639,841	235,241	0.205	0.056	0.027	0.151	0.014	0.453

(a) (b)

Figure 10 Trunks decorated with twigs, leaves and fruits. (a) Rhus; (b) eucalyptus.

see from Tables 1 and 2 that most of the computational time is spent in potential field value calculation at vertices of cubes or tetrahedra. The computational time is reduced in our method because the numbers of subdivided tetrahedra and their vertices are efficiently reduced through the generated bounding polyhedron. Although our tetrahedral voxelization of bounding polyhedron is performed on the CPU, it is not the bottleneck of the whole algorithm since the voxelized object is a quite coarse bounding polyhedron consisting of only a small number of vertices. To sum up, for the same order of magnitude of the extracted triangles, our approach is more than 10 times faster than marching tetrahedra while producing meshes with higher quality.

Figure 10 shows photo-realistic trees by adding tiny twigs, leaves and textures to the extracted tree trunks using our method.

6 Conclusions

The features of convolution surfaces make it good for modeling branching shapes. However, small branchlets are usually lost using traditional iso-surface extraction algorithms. Although multi-resolution or curvature-adaptive iso-surface extraction algorithms partly solve the problem, they need to deal with

complex topologies, which greatly hinders the parallelism. Considering parallel computation on the GPU is an excellent solution to speeding up the calculation of convolution surfaces, we have designed a novel polygonization scheme on the GPU for tree trunks modeled by convolution surfaces. First, a bounding polyhedron enclosing the whole iso-surface is created based on the skeletons of the tree trunk, and the TetGen library is employed to tetrahedralize the polyhedron into tetrahedra adaptively according to the thickness of the trunk. Second, we repeatedly subdivide the tetrahedra into a user-specified depth and perform the iso-surface extraction within the sub-tetrahedra parallelly on the GPU. As the sizes of subdivided tetrahedra are proportional to the trunk radii, the extracted polygonal iso-surfaces have adaptive edge lengths. Our approach neither misses small twigs nor produces many triangles at thick trunks. That is, the generated triangles are dependent on the curvatures of the iso-surface. In the meantime, our approach produces better polygonized meshes. Third, the bottleneck of the polygonization of tree trunks using convolution surfaces lies in the computation-intensive potential field calculation, as lots of unnecessary vertex field computation is efficiently avoided, our approach is more than one order of magnitude faster than marching tetrahedra.

Although the sizes of triangles are adaptive to the radii of tree trunks, long and narrow triangles will be produced in our result. This is the limitation of our approach. In the future, we will try to optimize the triangles on the GPU. Also, our GPU implementation can be further optimized to improve the efficiency of the presented algorithm.

Acknowledgements

This work was supported by National Key Basic Research Foundation of China (Grant No. 2009CB320801), National Natural Science Foundation of China (Grant No. 60933007), and Zhejiang Provincial Natural Science Foundation of China (Grant No. Z1110154).

References

1 Deussen O, Lintermann B. Digital Design of Nature: Computer Generated Plants and Organs. New York: Springer, 2005
2 Okabe M, Owada S, Igarashi T. Interactive design of botanical trees using freehand sketches and example-based editing. Comput Graph Forum, 2005, 24: 487–496
3 Neubert B, Franken T, Deussen O. Approximate image-based tree-modeling using particle flows. ACM Trans Graph, 2007, 26: 88
4 Livny Y, Pirk S, Cheng Z L, et al. Texture-lobes for tree modelling. ACM Trans Graph, 2011, 30: 53
5 Livny Y, Yan F L, Chen B Q, et al. Automatic reconstruction of tree skeletal structures from point clouds. ACM Trans Graph, 2010, 29: 151
6 Xu H, Gossett N, Chen B Q. Knowledge and heuristic-based modeling of laser-scanned trees. ACM Trans Graph, 2007, 26: 19
7 Lindenmayer A. Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. J Theor Biol, 1968, 18: 300–315
8 Talton J O, Lou Y, Lesser S, et al. Metropolis procedural modeling. ACM Trans Graph, 2011, 30: 11
9 Anastacio F, Sousa M C, Samavati F, et al. Modeling plant structures using concept sketches. In: Proceedings of 4th International Symposium of Nonphotorealistic Animation and Rendering. New York: ACM, 2006. 105–113
10 Chen X J, Neubert B, Xu Y Q, et al. Sketch-based tree modeling using markov random field. ACM Trans Graph, 2008, 27: 109
11 Palubicki W, Horel K, Longay S, et al. Self-organizing tree models for image synthesis. In: Proceedings of ACM SIGGRAPH. New York: ACM, 2009. 1–10
12 Reche-Martinez A, Martin I, Drettakis G. Volumetric reconstruction and interactive rendering of trees from photographs. ACM Trans Graph, 2004, 23: 720–727
13 Tan P, Fang T, Xiao J X, et al. Single image tree modeling. ACM Trans Graph, 2008, 27: 108
14 Tan P, Zeng G, Wang J D, et al. Image-based tree modeling. ACM Trans Graph, 2007, 26: 87
15 Bucksch A, Lindenbergh R. Campino-a skeletonization method for point cloud processing. ISPRS-J Photogramm Remote Sens, 2008, 63: 115–127
16 Bucksch A, Lindenbergh R, Menenti M. Skeltre-fast skeletonization for impefect point cloud data of botanic trees. In: Proceedings of the 2nd Eurographics conference on 3D Object Retrieval. Switzerland: Eurographics Association

Aire-la-Ville, 2009. 13–27
17 Au O K C, Tai C L, Chu H K, et al. Skeleton extraction by mesh contraction. In: Proceedings of ACM SIGGRAPH. New York: ACM, 2008. 44
18 Cao J J, Tagliasacchi A, Olson M, et al. Point cloud skeletons via laplacian-based contraction. In: Proceedings of IEEE Conference on Shape Modeling and Applications. Washington DC: IEEE Computer Society, 2010. 187–197
19 Ma W, Xiang B, Zha H B, et al. Modeling plants with sensor data. Sci China Ser F-Inf Sci, 2009, 52: 500–511
20 Bloomenthal J, Shoemake K. Convolution surfaces. Comput Graph, 1991, 25: 251–256
21 Pan R J, Meng X X, Whangbo T K. Hermite variational implicit surface reconstruction. Sci China Ser F-Inf Sci, 2009, 52: 308–315
22 Sherstyuk A. Kernel functions in convolution surfaces: a comparative analysis. Visual Comput, 1999, 15: 171–182
23 Zhu X Q, Jin X G, Liu S J, et al. Analytical solutions for sketch-based convolution surface modeling on the GPU. Visual Comput, 2012, 28: 1115–1125
24 Alexe A, Barthe L, Cani M P. Adaptive implicit modeling using subdivision curves and surfaces as skeletons. In: Proceedings of the 7th ACM Symposium on Solid Modeling and Applications. New York: ACM, 2002. 45–52
25 Hubert E. Convolution surfaces based on polygons for infinite and compact support kernels. Graph Models, 2012, 74: 1–13
26 Jin X G, Tai C L. Analytical methods for polynomial weighted convolution surfaces with various kernels. Comput Graph, 2002, 26: 437–447
27 Jin X G, Tai C L. Convolution surfaces for arcs and quadratic curves with a varying kernel. Visual Comput, 2002, 18: 530–546
28 Jin X G, Tai C L, Feng J Q, et al. An analytical convolution surface model for line skeletons with polynomial weighted distributions. J Graph Tools, 2001, 6: 1–12
29 Jin X G, Tai C L, Zhang H L. Implicit modeling from polygon soup using convolution. Visual Comput, 2009, 25: 279–288
30 Lorensen W, Cline H. Marching cubes: a high resolution 3D surface construction algorithm. Comput Graph, 1987, 21: 163–169
31 Gueziec A, Hummel R. Exploiting triangulated surface extraction using tetrahedral decomposition. IEEE Trans Vis Comput Graph, 1995, 1: 328–342
32 Payne B A, Toga A W. Surface mapping brain function on 3D models. IEEE Comput Graph Appl, 1990, 10: 33–41
33 Schaefer S, Warren J. Dual marching cubes: primal contouring of dual grids. In: Proceedings of the 12th Pacific Conference on Computer Graphics and Applications. Washington DC: IEEE Computer Society, 2004. 70–76
34 Poston T, Wong T T, Heng P. Multiresolution isosurface extraction with adaptive skeleton climbing. Comput Graph Forum, 1998, 17: 137–148
35 Kazhdan M, Klein A, Dalal K, et al. Unconstrained isosurface extraction on arbitrary octrees. In: Proceedings of 5th Eurographics Symposium on Geometry Processing. Switzerland: Eurographics Association, 2007. 125–133
36 Ji Z P, Liu L G, Wang Y G. B-mesh: a modeling system for base meshes of 3D articulated shapes. Comput Graph Forum, 2010, 29: 2169–2178
37 Tai C L, Zhang H X, Fong C K. Prototype modeling from sketched silhouettes based on convolution surfaces. Comput Graph Forum, 2004, 23: 71–83
38 Kravtsov D, Fryazinov O, Adzhiev V, et al. Embedded implicit stand-ins for animated meshes: a case of hybrid modeling. Comput Graph Forum, 2010, 29: 128–140
39 Wang M, Feng J Q. 2D-manifold boundary surfaces extraction from heterogeneous object on GPU. J Comput Sci Technol, 2012, 27: 862–871