Capturing, Rendering and Simulation for Large Scale Grassland

Zengzhi Fan, Shanghai Jiao Tong University
Bin Sheng, Shanghai Jiao Tong University

Abstract—Grass is a very import element of nature, however creating, rendering and simulating a large scale grassland is not easy due to extremely high computation complexity and large amount of data needed for simulation and rendering. Common sense tells us grass blades so simple that it is not difficult to set up a grass blade model for rendering and simulation, yet there are numerous kinds of grass blades in this world and it's not possible for any system to store all kinds of grass blades in advance. Capturing grass blade with interactive camera can be an intuitive solution for this problem. We provide a method that can capture grass blade shapes with a depth camera, render large scale grassland efficiently and simulate this grassland with individual response for each single blade on the fly.

Index Terms—Grass, Capture, Render, Simulation, GPU

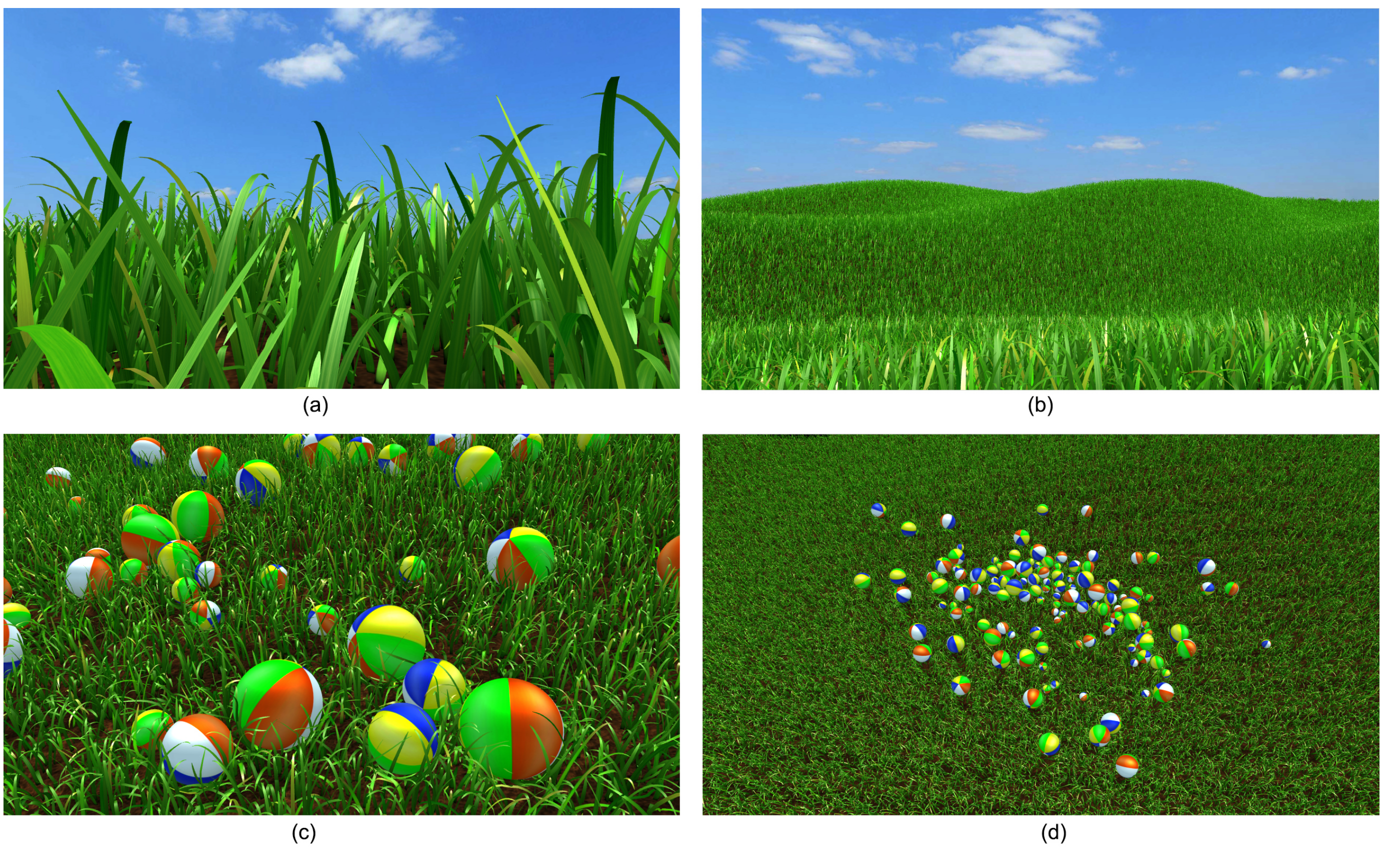

Fig. 1: Large scale grassland with our method: (a) grass detail in the scene; (b) front view of large scale grassland; (c) grass-ball interaction handling in our system; (d) bird's view of test scene.

1 Introduction

GRASS is a significant feature of natural world and it is indispensable in most 3D games and movies .Therefore the modeling, rendering and simulation of grass become an essential field of computer graphics and visualization. When an object passes on a grassland, every grass blade on the path is pushed aside or run over, and gradually recovers its shape afterwards. Individual responses from every single grass blade greatly improve viewers' experience for virtual environment and increase scene's fidelity. Realistic modeling for grass blade rendering is not easy since grass blade shape varies a lot. Artists usually disregard the importance of grass blades modeling for their simple structure, meanwhile the lack of realistic grass blade model becomes a frequent issue when creating virtual scenes for games and other 3D applications.

Grass modeling, rendering and simulation may not be so difficult for a single blade, however challenge comes when quantity is taken into consideration. Geometry-based modeling is used in most methods where structures are represented as triangles. Nevertheless when there is requirement for large quantity of grass blades, vertex amount for a single grass blade is strictly limited to a very small number for the sake of high performance. Modeling realistic grass blade with vertex amount restriction is challenging, no mater manually or recovering from images or videos. Compared with hair, which could be described with line segments, grass blade has far more kinds of shapes and structures meanwhile there are even more grass blades required for rendering and simulation than hairs in most cases. Therefore current hair rendering techniques are not applicable for scenes with extreme large amount of grass blades and various grass shapes. Simulation for hair is rather a group problem since movements for hairs are quite similar. Thus hairs can be divided into groups and guide hairs are usually used as representatives for simulation. Head movement is the major cause of hair movement and head movement range is rather small and highly predictable, which greatly reduces the difficulty of hair simulation. For grass blades, individual response for any other objects is essential and cannot be simply divided into groups for simulation. This becomes the key problem for grass simulation. Besides, grass blades have stationary shapes, they should recover their shapes and positions after collision.

In this paper we present a novel and effective framework for extremely large amount of grass blades modeling, rendering and simulation. Our framework is able to capture grass blade from camera, refine grass blade shape and render large scale grassland scene with high quality as well as realistic simulation response to collisions. Depth cameras are specially valued since they are able to provide depth information of any scene, which makes it much easier to rebuild any object or scene. With depth camera, we are able to capture grass blade shape instantly instead of using a small number of preset shapes. Compared with image-based method, depth-camera-based method is much faster and is able to get far more accurate depth information. More importantly, cameras provide human-computer interaction and allow users to adjust capturing results through instant feedback on the screen. Camera-based method also provide possibility for inexperienced users to create models for their own virtual scenes.

In order to deal with extremely large grass blade amount and achieve real-time performance of rendering and simulation, triangles are abandoned for grass representation. Grass blades are stored as line segments and are expanded to their final shape at run time. Line segments are extracted from depth image as grass blade skeleton meanwhile expansion width for each vertex is calculated from blades' contour obtained from depth camera. Expansion widths partially define blades' shape and structure. A particle-flow method is adopted to partition single blade contour from a cluster blades.

In our system every single grass blade has individual and realistic response to collision. We apply GPU-based instancing to lower the requirement for memory and bandwidth, and pay simulation costs only when needed. Meanwhile, this is a tile-based rendering and simulation system, no per-tile data is store unless simulation for a specific tile is required. We implement the method introduced by Han et al. [1] to simulate our grass blade and extend the method introduced by Fan et al. [2] to do tile management. Unlike any simulation only for key blades or hairs in the scene, each grass blade could have separate movement and independent collision to objects. Real-time performance is achieved with more than one million grass blades and hundreds of objects.

Our main contributions are listed below:

- Interactive grass blade capturing method with depth camera,including skeleton extraction and expansion width calculation;
- Blade skeleton simplification and refinement according to vertices' movement similarity;
- Extension for large scale grassland rendering as well as simulation method with high fidelity, which allow instant shape editing during rendering and simulation.

2 RELATED WORK

The most challenging part of grass modeling, rendering and simulation is caused by extremely large quantity. William Reeves and Ricki Blau [3] addressed those challenges in 1985. Works about grass mainly discuss the following three topics: grass modeling, rendering and simulation. Kajiya et al. [4] introduced volumetric textures(texels) for short fur rendering. Texels can used to solve spatial aliasing problem. Neyret extended this work to simulate natural scenes such as realistic grass [5] [6]. Polygons stacks as well as semi-transparent textures were used in implementation of texels [7]. Brook et al [8] improved this method to obtain high rendering performance. Image-based method was used in grass rendering [9]. This method used bidirectional texture function for grass. Boulanger et al. [10] introduced a level-of-detail(LOD) method for grass rendering with realistic dynamic light and shadow effects. In previous works, grass blades were pushed away when interaction between grass blades and objects happens [11]. Spring-mass system was also used to model grass blade and simulate grass-object interaction [12] [13]. A method to modeling withering grass was introduced by Wen et al. [14] using time-varying texels. We adopt the simulation algorithm introduced by Han et al. [1]. This method treated collision as hard constraint, meanwhile treated length, bending and twisting as soft constraints in the iterative solver for grass-object interaction. We employ the rendering and simulation framework introduced by [2]. With this frame work, we are able to perform collision computation on GPU and do grass blade instancing on the fly. We implement our capture method on the basis of this framework to obtain more accurate and diverse grass types.

A number of works for leaf and flora reconstruction have reference value for our method. There are some previous works about using interactive method to generate or reconstruct leaf shapes and tree shapes [15] [16] [17] [18] [19].Quan et al. [20] introduced a method to model plant from a dozen of images. This method could recover plant's shape automatically while relying on user to provide some hints for segmentation. Tan et al. [21] proposed a method to generate 3D trees from images. They populated tree leaves from segmented source images and used shape patterns of visible branches to predict occluded branches. Tree modeling from single image was introduced afterwards [22]. Yan et al. [23] came up with a method for flower reconstruction from a single image, which used a cone fitting scheme to maintains flower shape.Bradley et al. [24] presented a scale technique to compute 3D structure of foliage and extract leaf shapes.

Capturing grass blade with camera has not been explored much, however plenty of researches about hair capture have been conducted. There are some similarities between hair and grass blade for their shapes and movement features. Chai et al. [25] uses single image to do hair modeling, they captured hair style from image and were able to render origin hair style at different angle, with different hair materials and change hair style . Afterwards, they extended their work through user's high-level knowledge to get more accurate hair that matches image and waa physically real at the same time. By doing it they were able to finish dynamic hair simulation and interactive hair style editing, which made it possible to apply this hair manipulation in video [26]. A structure-aware hair capture method [27] was introduced in 2013, authors adopted a method to generate hair strand segments, set up a connection graph to guarantee hair growth to areas with missed geometry information and connected hair strands with consistent curvature. A method to capture hair using simulated examples was introduced by Hu et al. [28], they used simulated samples as references to generate hair, and this method could be applied with unconstrained and constrained hair. They also came up with a method to capture braided hair style [29]. A data-driven reconstruction method was adopted in this scheme and procedurally generated examples were used to fit captured hair strands. Xu et al. [30] introduced a space-time optimization method to capture dynamic hair. This method could faithfully capture hair strands' shape as well as spatial details.

Level-of-detail(LOD) methods are typical model simplification algorithms, they are used to simplify model geometry complexity and accelerate rendering as well as simulation performance. Framework used to obtain a constant frame rate for visualization of virtual environments was introduced in 1993 [31]. Geometry-based LOD algorithms including quad remeshing methods were summarized in [32]. Field-guided parameterization-based methods split quad remeshing into three steps including cross-field computation, integer-grid parameterization and quad mesh extraction [33] [34] [35] [36]. Except for those geometry-based LOD algorithms, level-of-detail was also used in simulation method. Techniques for reducing the computational complexity for simulations was introduced by Carlson et al. [37]. LOD was also used for animation and rendering of prairies and particle systems [38]. Our skeleton refining method is similar to those LOD algorithms, moreover we take simulation fidelity into account while simplifying grass blade model. This leads to more accurate model for rendering and simulation at the same time and we are able to balance the trade-off between performance and quality.

3 Algorithm Overview

The overview of our algorithm is shown in Figure.2. We employ a depth camera in our system which is able to obtain depth image of target plant. According to this depth image, we are able to extract the contour of grass blades by distance masking method in real time, which is difficult to achieve without depth information. Single blade's contour could be partitioned from a cluster of blades by a particle-flow method [39].

According to contour of grass blade, we calculate blade's skeleton, which is used to describe this blade and is also used in simulation. In vertex shader we expand each knot on the skeleton to form the blade. Expansion widths are calculated from blade's contour and smoothed before rendering.

For skeleton captured from camera, we design an refining algorithm to do simplification. This process is similar to level-of-detail(LOD). Since skeleton knots number is restricted by extremely large amount of grass blades, this refining algorithm could reduce skeleton knots number from over one hundred to 32, 16 or even less, according to knots' movement similarity.

We adopt the GPU instancing scheme used in [2]. We pre-generate a list of grass blade called grass patch. Grass scene is divided into a grid of tiles. Grass blades in each tile are specific number of continuous blades fetched from grass patch. Starting position in grass patch for each tile is calculated by tile's position in the scene. Blade's skeleton is expanded to triangles before rendering. Phong shading model is used in grass blade rendering for the sake of high performance. Subsurface scattering effects is also used in our system to increase fidelity [40]. Tile management guarantee per-blade memory is allocated only when simulation is needed.

4 Algorithm Details

4.1 Blade Capture

we aim to capture grass blade without manual operations and use capture result in the rendering and simulation

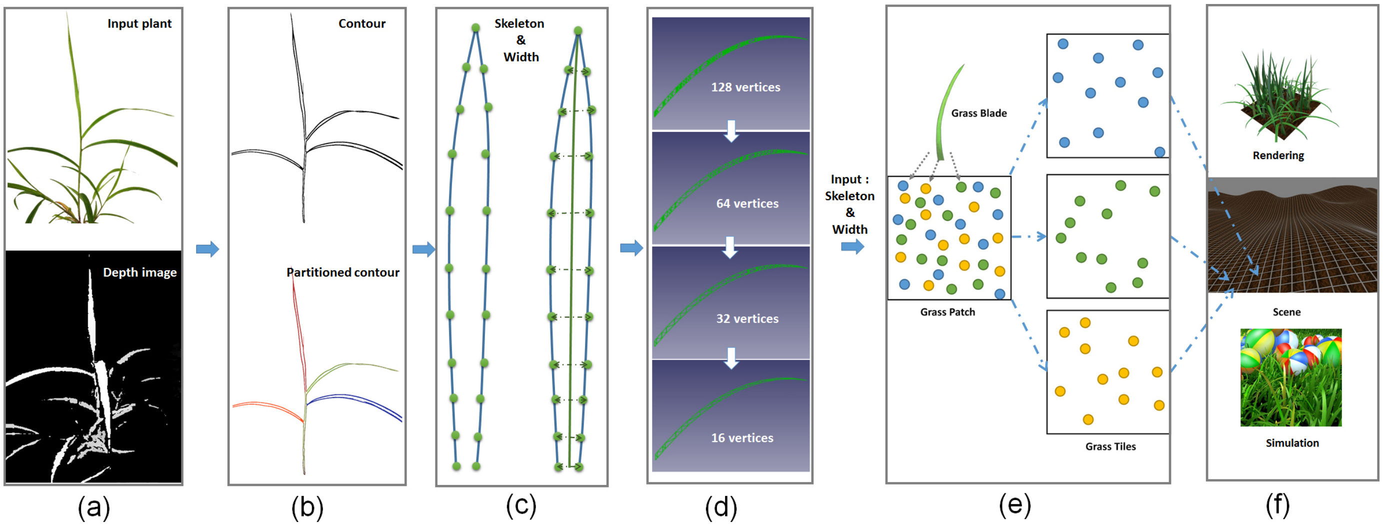

Fig. 2: Overview of system pipeline:(a) a cluster of grass blades and depth depth image captured with camera, those are raw data in our system; (b) extracted contour(top), partitioned contour for single blade marked with different color respectively(bottom);(c) skeleton and expansion width calculation(bottom); (d) skeleton refining process using a idealized blade skeleton with 128 vertices, with different refining levels that reduce skeleton vertices down to 64,32 and 16 vertices; (e) instancing scheme: using refined skeleton to generate a grass patch, every tile refer to a subset of grass blade in the patch, expand skeleton to grass according to expansion width; (f) rendering and simulation.

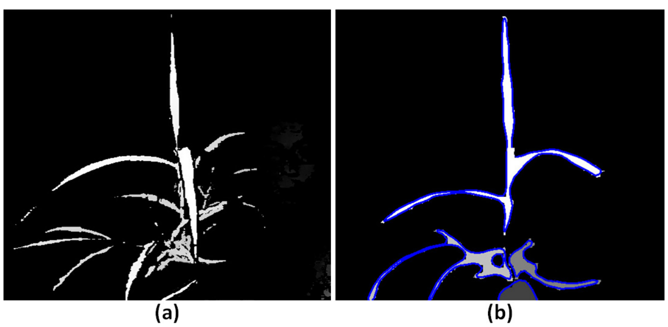

Fig. 3: (a)Depth image and (b)extracted contour

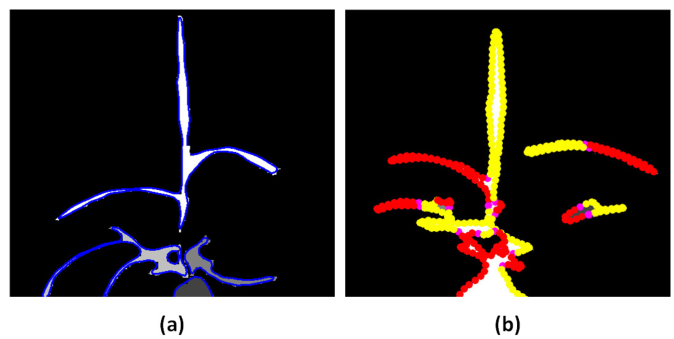

Fig. 4: Single blade partition: Given the (a)contour of a blade cluster,(b)calculate the center point and use particle-flow method to partition each single blade.

system immediately after capturing. Skeleton knots number limits the detail level of our grass blade, however it also requires efficient refining algorithm to convert captured skeleton into appropriate blade model in our system. Motivated by other hair and plant capture methods, we adopt a depth camera to capture grass blade. Depth cameras are deployed in variety of applications and becoming prevailing more than ever, for its advantage to recover shapes and capture movements. With depth camera, we manage to capture grass blade and mask non-grass object on the fly.

Figure3 shows the depth image of a cluster of grass blades and its extracted contour. We just use depth information to mask objects that exceed specified depth to get a blob of blades and then extract edge of the blob. Also canny edge detection can be used to extract contour, however depth value, instead of gray is used to do calculation.

4.2 Single Blade Partition

Individual grass blades are partitioned by a particle-flow method [39]. After getting the contour of grass blades, the center of the contour can be easily calculated. Then the distance between any point on the contour and center point can be measured. Local minimum points of point-distance function imply the intersection points between adjacent blades. Boundary between two intersection points is contour of a single blade. Figure.4 shows the partition result projected in 2D plane.

4.3 Skeleton and Expansion Width Calculation

Given the contour of single blade, we could calculate the blade skeleton. According to our observation on plenty kinds of grass blade ,we suppose that captured blade is symmetrical. Then we could calculate skeleton point by matching symmetrical points on contour. Figure.5 present skeleton calculation according to blade contour. Expansion

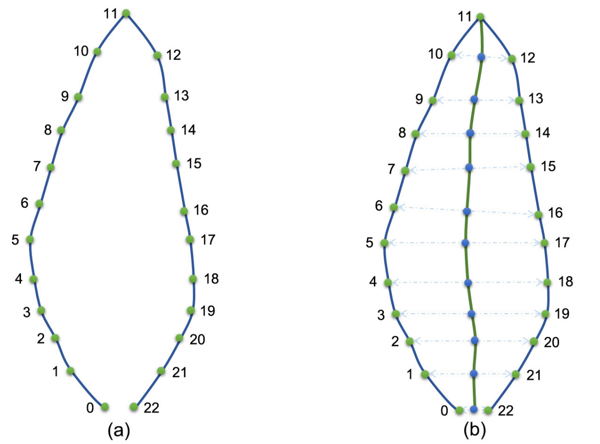

Fig. 5: Skeleton calculation by match symmetrical points on contour.(a) capture single blade contour points, points index increases from bottom left to bottom right;(b) Matching symmetrical points in the contour.

width for each point of skeleton could also be calculated by symmetrical points' distance.

4.4 Skeleton Refining

Having the blade skeleton calculated from contour, it can't be used in our rendering and simulation system for its large number of vertices. Therefore we come up with a skeleton refining method to simplify our skeleton model. This skeleton refining method is similar to some LOD methods. However most LOD methods are designed for triangle meshes or quad meshes. Our skeleton is represented as a curve and more importantly, we need to take geometry fidelity as well as vertices' movement similarity into account simultaneously while doing simplification. Therefore we design this refining scheme that both geometry and simulation similarities of vertices are important factors in skeleton simplification and fidelity evaluation.

In our system, we define any two vertices' movement similarity by the distance between them and their movement trends. For any two vertices, the smaller the distance between them is, the more similar they are. And for any two vertices, we could calculate the distance between them at any time. We could acquire the distance of two vertices every a few millisecond. Then we are able to calculate a series distance data for these two vertices. Variance is usually used to describe data stability. If distance variance is low, it means the distance between two vertices is stable. In another word, this indicate that these two vertices moves synchronously. We tend to merge vertices that moves similarly and keep vertices that have distinct movement features. Assume we acquire n frames every a few milliseconds, v_m^n is mth vertex at nth frame, therefore we introduce movement difference of any two vertices as:

$$D(v_1, v_2) = dis(v_1, v_2) \times \mu + var(v_1, v_2) \times (1 - \mu) \quad (1)$$

$$dis(v_1, v_2) = \frac{\sum_{i=0}^{n} d(v_1^i, v_2^i)}{n} \quad (2)$$

$$var(v_1, v_2) = \left(\frac{\sum_{i=0}^{n} d(v_1^i, v_2^i)^2}{n} - \frac{(\sum_{i=0}^{n} d(v_1^i, v_2^i))^2}{n}\right) \quad (3)$$

dis function represents distance factor and var function represents variance factor. μ is proportion factor for distance and variance, which can be set by user.

We use an iterative algorithm to complete this skeleton refining process. After skeleton calculation, we manage to obtain array of skeleton vertices. We design a greedy algorithm to iteratively merge two adjacent vertices, based on the assumption that if two vertices are the most similar then they should be adjacent vertices in the array. Because distance factor is an very important part in similarity definition, and closer vertices always get lower score for distance factor. This algorithm is illustrated in Algorithm.1.

Algorithm 1 Skeleton Refining

//SkeletonVertices = Calculated from captured result
while SkeletonVertices.size() $> m$ **do**
  DifferenceArr = $\emptyset$
  for v_i in SkeletonVertices **do**
    $d_i = D(v_i, v_{i+1})$
    DifferenceArr.add(d_i)
  end for

  //Search for minimum d in DifferenceArr
  ind = DifferenceArr.minimum()
  $newV = (v_i + v_{i+1})/2$

  //Delete old vertices
  DifferenceArr.delete(ind)
  DifferenceArr.delete($ind + 1$)

  //Insert new vertex at position i
  DifferenceArr.insert($newV$, ind)
end while

Figure.6 illustrates a sequence of grass blade with different refining level. Refining level is set by user according to the performance requirement of application and hardware level.

4.5 Rendering and Simulation

According to our captured grass model and refining result, we pre-generate a list of grass blades with some random scaling.We divide the whole scene into tiles. Those blades in the patch are randomly located in a square which has the same size of grass tile in the scene. Each tile contains a subset of continuous blades in the patch. For each grass tile, its offset in the patch is calculated by tile's position coordinate in the scene. With this instancing scheme, we are able to reduce memory use from about 4 GB of 4 M blades to only 24 MB of for a patch of 16384 blades. With this patch, we manage to implement a grassland with rich variance.

We draw one line segment as two degenerate triangles and expand each knot of this line segment at runtime

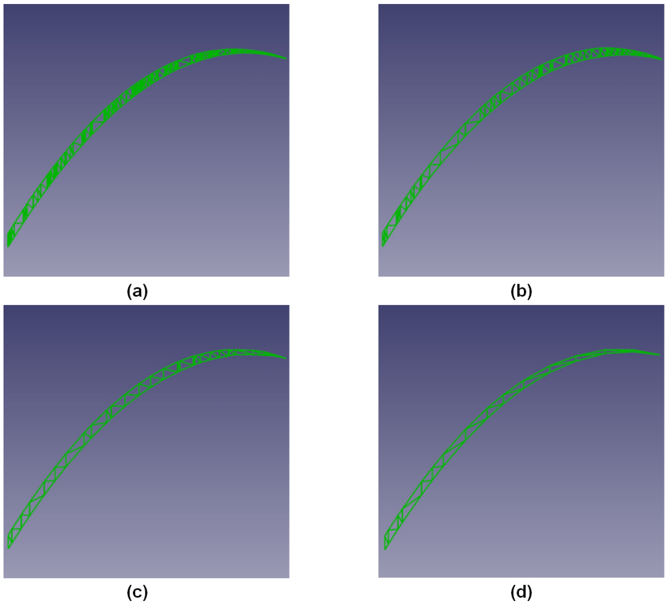

Fig. 6: Skeleton refining process with a sequence of idealized grass blade skeletonin the figure this skeleton is expanded as it is done in vertex shader, at first there is (a) blade skeleton with 128 vertices; after different level of skeleton refining, (b) blade skeleton with 64 vertices; (c) blade skeleton with 32 vertices; (d) blade skeleton with 16 vertices.

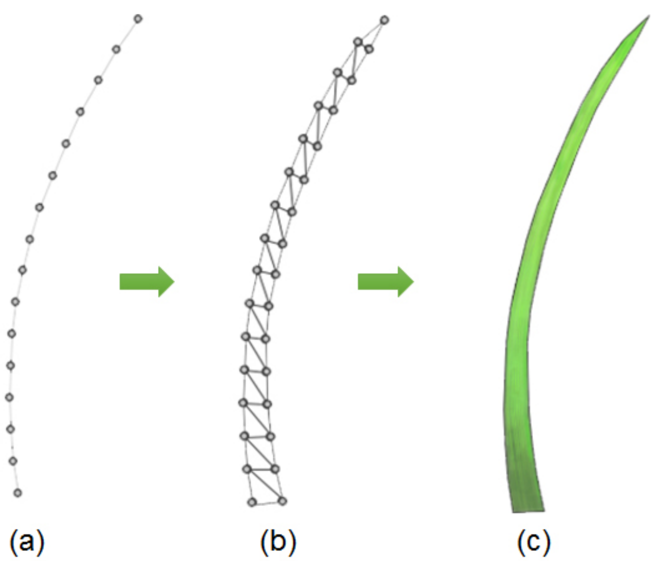

Fig. 7: Blade skeleton expansion:(a) Simulation blade skeleton presented by overlapping vertices; (b) Expanded triangle model from blade skeleton for rendering; (c) shaded result.

according to expansion width captured through depth camera. Figure.7 illustrates this expansion process. We employ Phong shading along with subsurface scattering effect [40] in our system. In order to avoid a monotone we use a density map in our system. Figure.8 illustrated our density map generated with Perlin noise [41].

In our simulation system, $Bullet$ is used to handle collision between ground and other object. We use the simulation method introduced by [1] and tile management scheme introduced by [2]. This simulation scheme is compatible with procedural animation. We are able to simulate millions of grass blades with real-time performance.

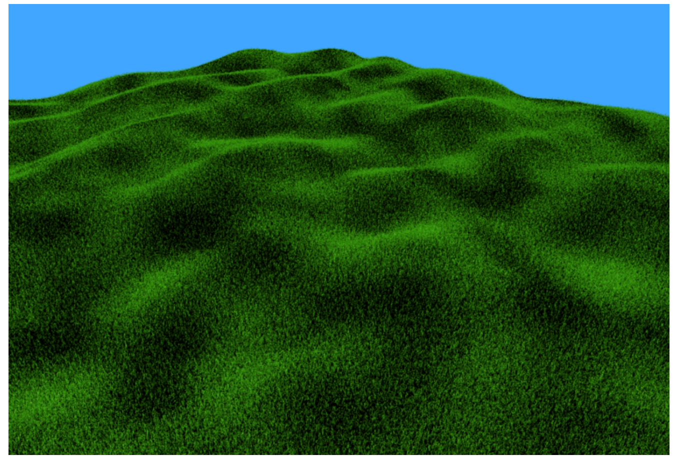

Fig. 8: Density map used in our rendering system.

5 Implementation and Results

We implemented our system on a PC with Windows 8.1, Intel i5-4460 CPU running at 3.2 GHz and AMD Radeon R9-270 graphic card. 4 times multi-sampling anti-aliasing is used in all experiments to guarantee satisfactory effect.

Instancing rendering is used in our implementation in order to reduce draw call overhead for CPU. Instancing parameters are pre-generated and stored in a structure buffer before rendering. Our rendering scheme is also compatible with different LOD algorithms to at far distance.

5.1 Capture, Rendering and Simulation Results

Our system is evaluated with a variety of grass blades and leaves that can be used for large scale grassland. Our capture algorithm can capture grass blade, partition single blade contour, calculate expansion width and finish skeleton refining at interactive performance. This provide the possibility for user to capture any appropriate kind of grass or leaves and use it for large scale grassland rendering and simulation. This is especially useful for virtual-reality or augmented-reality applications where quick reconstruction of grassland with specific grass blade type is needed.

In Figure.10(a) we choose a regular kind of grass which is slim ,thin and common to find everywhere. In (b) we choose a vine and capture its leaf as grass blade. This blade is like a tree leaf, its blade edge can be described as a quadratic curve. In (c) we choose a cluster of leaves from a shrub. This leaf is longer however its blade edge is still regular and quadratic-curve like. In (d) we choose a cluster of bamboo leaves. Bamboo leave is thin and slim, from the figure we could see that its blade edge

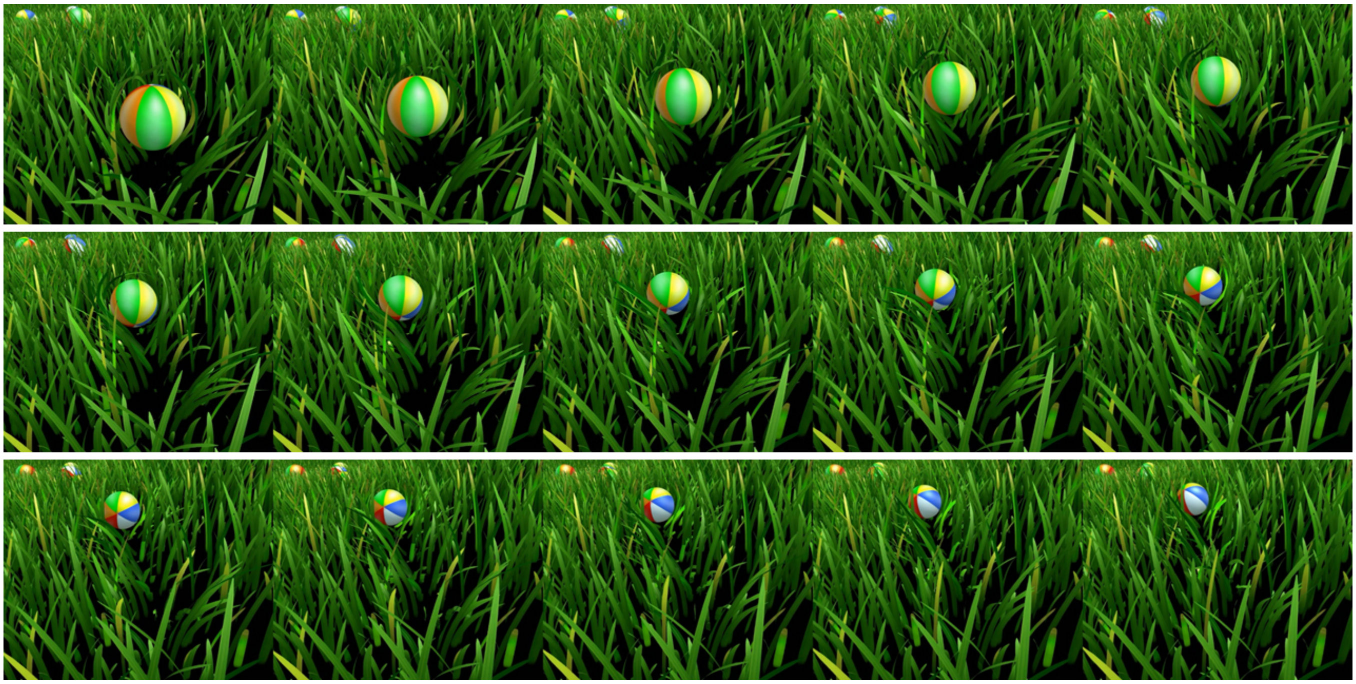

Fig. 9: Sequence of frames obtained through our rendering and simulation method.

is very smooth and flat in the middle and it is hard to be described with simple quadratic curves. In(e) we choose a leaf with variation it the top. It has a very sharp and narrow tip which make it special. Camera captured this feature kept it through our expansion width calculation.

Tiles	6	14	150	709
Blades	384	896	9600	45376
Objects	1	2	20	128
Sim time(ms)	0.176	0.183	0.952	4.67
Sim time/Blade (ns)	0.458	0.204	0.099	0.103

TABLE 1: Simulation time data with different activated tiles, objects.

We have tested our algorithm to evaluate its rendering and simulation performance. In our system, we need only one draw call to rendering the whole scene, which greatly reduce CPU overhead. We are able to render a scene of 14926 tiles in 20 ms, in total there are 955264 blades and over 29 million triangles. In table.1, profiling data for simulation demonstrates the effectiveness of our system. Simulation time for a single blade decreases with the increase of simulated tiles as listed in the table. This gives the evidence that our system is suitable to solve the high simulation cost problem caused by extremely large grassland.

5.2 Skeleton Refining Evaluation

For error measurement of meshes, several methods such as the use error metric of average squared distance were used in previous works [42]. Our grass blade skeleton is represented as a curve, in order to measure the effectiveness of our skeleton refining algorithm, we use a Hausdorff-Fit

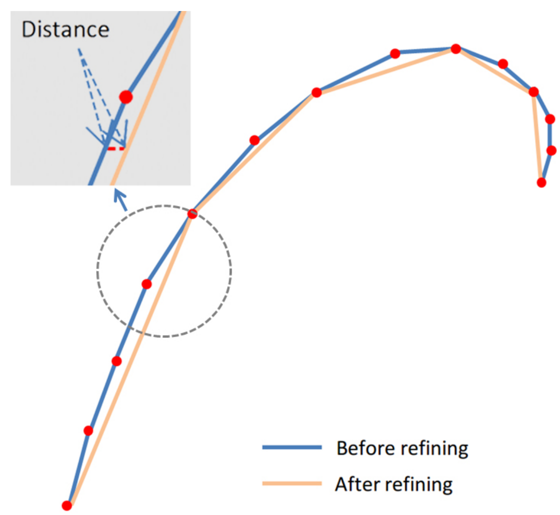

Fig. 11: Distance between refined skeleton and original skeleton calculated from blade contour.

method to evaluate the refining result. We intend to measure the "distance" between refined skeleton and the original one before refining, as it is demonstrated in Figure.11. This is very similar to error metric or energy term used in previous methods. Hausdorff distance is usually used to measure how far two subsets of a metric space are from each other [43]. Hausdorff distance is the maximum distance of a set to the nearest point in the other set.

Hausdorff distance is usually defined as:

$$d_H(X,Y) = \max\{\sup_{x\in X}\inf_{y\in Y} d(x,y), \sup_{y\in Y}\inf_{x\in X} d(x,y)\} \quad (4)$$

where sup represents the supremum of the set and

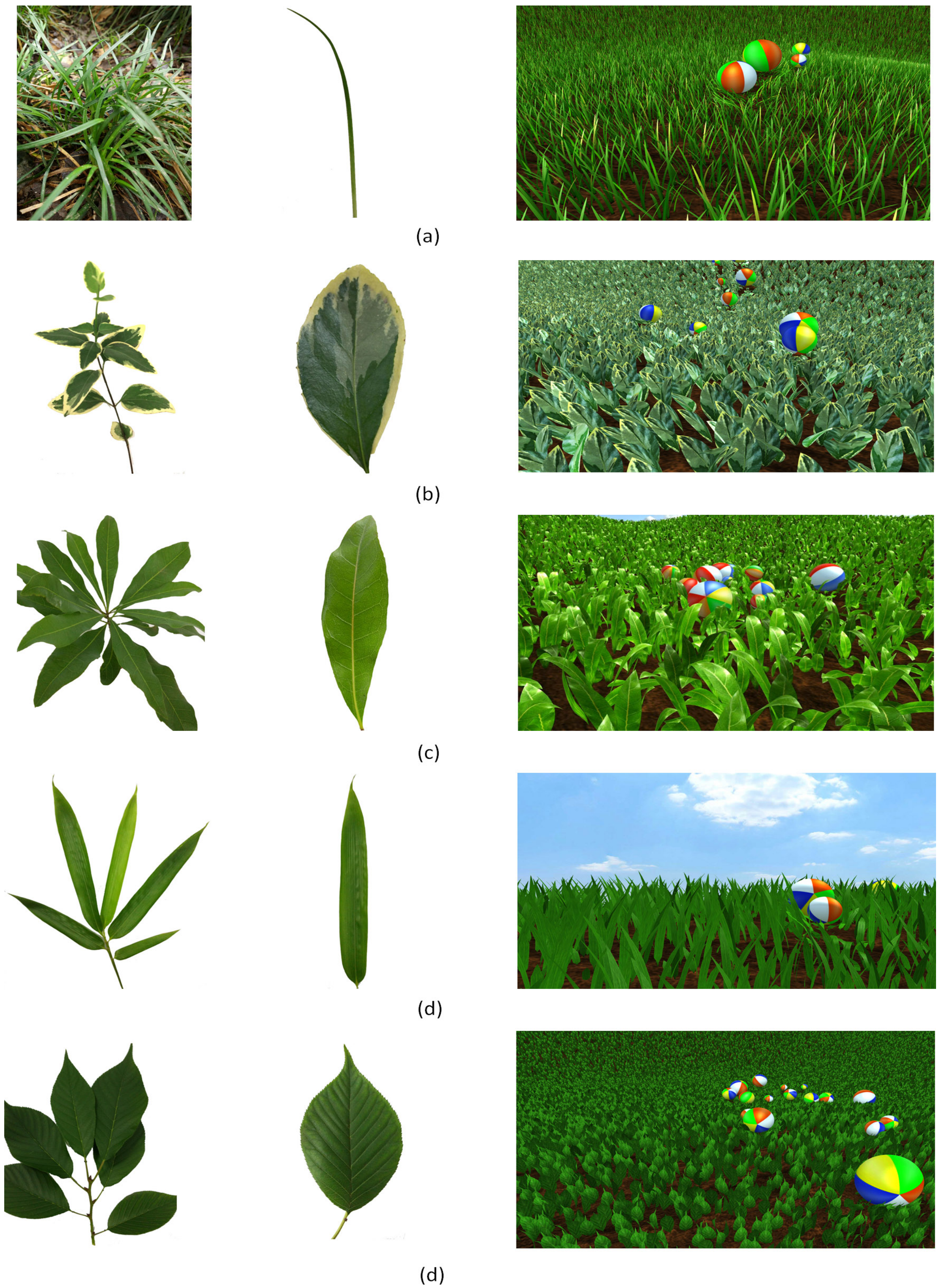

Fig. 10: Experiment results:(a) regular slim and thin grass blade ;(b) blade similar to most leaves, blade edge is quadratic curve like; (c) blade that is slender than the first one, blade edge is still regular (d) bamboo leaf which is thin, its edge is very smooth and flat except for its top and root; (e) leaf with shape variation, it has a very sharp and narrow tip.

inf represents the infimum. X, Y are two sets while x, y are elements that belong to X, Y respectively. In our method, Hausdorff distance is used to measure the similarity of refined skeleton and the original one before refining. However, since the skeleton after refining could be a sub set of the original one, hausdorff distance could become meaningless in that case. Therefore we eliminate the vertices from original skeleton vertex set α if they are also in refined skeleton vertex set β, let $\alpha' = \alpha - \beta$, then we calculate the hausdorff distance from α' to β. In this way we actually measure the distance between the culled vertex set and the remained vertex set after refining. Algorithm for our method is demonstrated in algorithm.2 where d_H is refined hausdorff distance used in our algorithm.

Algorithm 2 Skeleton Refining Evaluating

//SkeletonVertices = Calculated from captured result
α = Original skeleton vertex set
β = Refined skeleton vertex set
for v in α **do**
  if v is β **then**
    Delete v from α
  end if
end for

$d_H = MIN$
for v_1 in α **do**
  $d = MAX$
  for v_2 in β **do**
    $d = min(distance(v_1, v_2), d)$
  end for
  $d_H = max(d, d_H)$
end for

We recorded vertex data from several frames according to our simulation algorithm. Simulation for original skeleton vertices and refined skeleton vertices are conducted at the same time. Then we calculate d_H as in algorithm.2 in each frame for the results of different refining methods. For the needs of comparison, we introduces several other schemes used for skeleton refining. Random method selects vertices randomly from original vertex set. Mean method select vertices evenly along the skeleton from the root to the top. only geometry information is used in geometry-based refining algorithm. It iteratively merge vertices with shortest distance of the skeleton. Since grass blade skeleton is represented as a curve, many LOD methods for meshes could degenerate to it eventually. Refining error is represented as d_H. From Figure.12 we could see that our skeleton refining algorithm always gets the lowest error among the methods listed above. Mean method is a intuitive method used in manual grass blade modeling, however it is not a good choice proved with our experiments. Geometry-based simplification is widely used however, it gets higher refining error since it only considers static mesh structure while for dynamic scenes, simulation similarity helps to get a more accurate grass blade model.

Limitations. Restricted by vertex amount of grass blade, our method has some limitations. First, small vertex amount

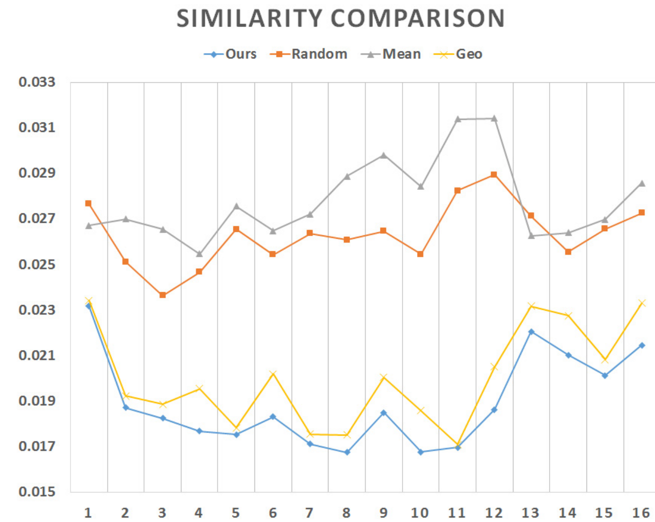

Fig. 12: Skeleton refining evaluation results: our methods, random method, mean method and geometry-based method.

is not enough to model jagged blades or leaves. Sharp variation in shape definitely need more vertices to remain such features. However, vertex amount is severely restricted by performance requirements, we need to balance vertex amount and performance at the same time. Meanwhile, due to the structure of grass blade we use in our system, we are not able to handle horizontally curly blades. Secondly, since we choose to do instancing before rendering and expand grass blade at runtime, moreover our skeleton calculation by matching points will run into difficulties when dealing with asymmetric grass blades or leaves. Thirdly, the center point calculation of a grass blade cluster requires appropriate pose for camera and blade cluster, causing some accuracy problem if the poses are not proper. Fourthly, depth image capture by depth camera only support a maximum resolution of 640×480 pixels, this will cause some problem when we want to capture some small and thin grass blades. Hopefully this problem would be solved with the development of hardware and popularity of depth camera.

6 Conclusion

In this paper, we present a framework for capturing grass blade with depth camera, expansion width calculation and skeleton extraction. We utilize a increasingly popular depth camera and take advantage of depth information provided by depth camera to accomplish efficient capturing. We extract a blob of blades according to objects' depth range and calculate its edge to obtain a contour. A particle-flow method is used to partition single blade's contour from a cluster of blades. Based on a single blade's contour, we calculate a skeleton by matching symmetrical vertices on the contour and expansion width for each vertices of the skeleton. A skeleton refining process according to vertices' movement similarity is introduced to reduce skeleton vertex number so that it can be used in the rendering and simulation system with acceptable performance. As for rendering, we adopt a GPU-instanced scheme reduce memory usage. A tile management method is employed to pay simulation cost for those tiles only when needed.

Our proposed method concentrate on capturing grass blade shape and refining its structure so that it can be used in the rendering and simulation method for large scale grassland. We are able to a variety of grass blades. We would like to extend our method to handle more complex blade shapes or capture other kind of plants. Furthermore, our method does not handle grass fracture and deformation. Skeleton and expansion width calculation may get incorrect result with such cases. We plan to handle grass fracture in future works and add support for structurally different vegetation.

In summary, our work demonstrate how interactive capturing method can be developed to model grass blade and use it in rendering and simulation frameworks. This work can be extended to many fields and inspire other capturing and modeling method for plants and other relative objects.

Acknowledgments

Thank my teammate for helping me with this paper.

References

[1] D. Han and T. Harada, "Real-time hair simulation with efficient hair style preservation," *vriphys12-proc*, pp. 45–51, 2012.
[2] Z. Fan, H. Li, K. Hillesland, and B. Sheng, "Simulation and rendering for millions of grass blades," in *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*. ACM, 2015, pp. 55–60.
[3] W. T. Reeves and R. Blau, "Approximate and probabilistic algorithms for shading and rendering structured particle systems," in *ACM Siggraph Computer Graphics*, vol. 19, no. 3. ACM, 1985, pp. 313–322.
[4] J. T. Kajiya and T. L. Kay, "Rendering fur with three dimensional textures," in *ACM Siggraph Computer Graphics*, vol. 23, no. 3. ACM, 1989, pp. 271–280.
[5] F. Neyret, *Synthesizing verdant landscapes using volumetric textures*. Springer, 1996.
[6] N. Fabrice, "Modeling, animating, and rendering complex scenes using volumetric textures," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 4, no. 1, pp. 55–70, 1998.
[7] A. Meyer and F. Neyret, "Interactive volumetric textures," in *Rendering Techniques 98*. Springer, 1998, pp. 157–168.
[8] B. Bakay, P. Lalonde, and W. Heidrich, "Real-time animated grass," in *Eurographics 2002*, 2002.
[9] M. A. Shah, J. Kontinnen, and S. Pattanaik, "Real-time rendering of realistic-looking grass," in *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. ACM, 2005, pp. 77–82.
[10] K. Boulanger, S. N. Pattanaik, and K. Bouatouch, "Rendering grass in real time with dynamic lighting," *IEEE Computer Graphics and Applications*, no. 1, pp. 32–41, 2009.
[11] S. Guerraz, F. Perbet, D. Raulo, F. Faure, and M.-P. Cani, "A procedural approach to animate interactive natural sceneries," in *Computer Animation and Social Agents, 2003. 16th International Conference on*. IEEE, 2003, pp. 73–78.
[12] K. Chen and H. Johan, "Real-time continuum grass," in *Virtual Reality Conference (VR), 2010 IEEE*. IEEE, 2010, pp. 227–234.
[13] N. Hempe, J. Rossmann, and B. Sondermann, "Generation and rendering of interactive ground vegetation for real-time testing and validation of computer vision algorithms," 2013.
[14] W. Wu, P.-A. Heng, E. Wu *et al.*, "Using time-varying texels to simulate withering grassland," *Computer Graphics and Applications, IEEE*, vol. 32, no. 1, pp. 78–86, 2012.
[15] L. Mündermann, P. MacMurchy, J. Pivovarov, and P. Prusinkiewicz, "Modeling lobed leaves," in *Computer Graphics International, 2003. Proceedings*. IEEE, 2003, pp. 60–65.
[16] M. Okabe, S. Owada, and T. Igarash, "Interactive design of botanical trees using freehand sketches and example-based editing," in *Computer Graphics Forum*, vol. 24, no. 3. Wiley Online Library, 2005, pp. 487–496.
[17] F. Anastacio, M. C. Sousa, F. Samavati, and J. A. Jorge, "Modeling plant structures using concept sketches," in *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*. ACM, 2006, pp. 105–113.
[18] X. Chen, B. Neubert, Y.-Q. Xu, O. Deussen, and S. B. Kang, *Sketch-based tree modeling using Markov random field*. ACM, 2008, vol. 27, no. 5.
[19] S. Pirk, O. Stava, J. Kratt, M. A. Massih Said, B. Neubert, R. Mech, B. Benes, and O. Deussen, "Plastic trees: interactive self-adapting botanical tree models," 2012.
[20] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang, "Image-based plant modeling," in *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3. ACM, 2006, pp. 599–604.
[21] P. Tan, G. Zeng, J. Wang, S. B. Kang, and L. Quan, "Image-based tree modeling," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 87.
[22] P. Tan, T. Fang, J. Xiao, P. Zhao, and L. Quan, "Single image tree modeling," *ACM Transactions on Graphics (TOG)*, vol. 27, no. 5, p. 108, 2008.
[23] F. Yan, M. Gong, D. Cohen-Or, O. Deussen, and B. Chen, "Flower reconstruction from a single photo," in *Computer Graphics Forum*, vol. 33, no. 2. Wiley Online Library, 2014, pp. 439–447.
[24] D. Bradley, D. Nowrouzezahrai, and P. Beardsley, "Image-based reconstruction and synthesis of dense foliage," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 74, 2013.
[25] M. Chai, L. Wang, Y. Weng, Y. Yu, B. Guo, and K. Zhou, "Single-view hair modeling for portrait manipulation," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 116, 2012.
[26] M. Chai, L. Wang, Y. Weng, X. Jin, and K. Zhou, "Dynamic hair manipulation in images and videos," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 75, 2013.
[27] L. Luo, H. Li, and S. Rusinkiewicz, "Structure-aware hair capture," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 76, 2013.
[28] L. Hu, C. Ma, L. Luo, and H. Li, "Robust hair capture using simulated examples," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 126, 2014.
[29] L. Hu, C. Ma, L. Luo, L.-Y. Wei, and H. Li, "Capturing braided hairstyles," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 6, p. 225, 2014.
[30] Z. Xu, H.-T. Wu, L. Wang, C. Zheng, X. Tong, and Y. Qi, "Dynamic hair capture using spacetime optimization," *To appear in ACM TOG*, vol. 33, p. 6, 2014.
[31] T. A. Funkhouser and C. H. Séquin, "Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM, 1993, pp. 247–254.
[32] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin, "Quad-mesh generation and processing: A survey," in *Computer Graphics Forum*, vol. 32, no. 6. Wiley Online Library, 2013, pp. 51–76.
[33] N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez, "Periodic global parameterization," *ACM Transactions on Graphics (TOG)*, vol. 25, no. 4, pp. 1460–1485, 2006.
[34] F. Kälberer, M. Nieser, and K. Polthier, "Quadcover-surface parameterization using branched coverings," in *Computer Graphics Forum*, vol. 26, no. 3. Wiley Online Library, 2007, pp. 375–384.
[35] D. Bommes, H. Zimmer, and L. Kobbelt, "Mixed-integer quadrangulation," in *ACM Transactions On Graphics (TOG)*, vol. 28, no. 3. ACM, 2009, p. 77.
[36] M. Zhang, J. Huang, X. Liu, and H. Bao, "A wave-based anisotropic quadrangulation method," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4. ACM, 2010, p. 118.
[37] D. A. Carlson and J. K. Hodgins, "Simulation levels of detail for real-time animation," in *Proc. of Graphics Interface 1997*, 1997.
[38] D. O'Brien, S. Fisher, and M. C. Lin, "Automatic simplification of particle system dynamics," in *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*. IEEE, 2001, pp. 210–257.
[39] B. Neubert, T. Franken, and O. Deussen, "Approximate image-based tree-modeling using particle flows," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, p. 88, 2007.
[40] T. Sousa, "Vegetation procedural animation and shading in crysis," *GPU Gems*, vol. 3, pp. 373–385, 2007.

[41] K. Perlin, “An image synthesizer,” *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[42] M. Garland and P. S. Heckbert, “Surface simplification using quadric error metrics,” in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216.

[43] Wikipedia, “Hausdorff distance, wikipedia, the free encyclopedia,” 2015.

Zengzhi Fan Biography text here.

Bin Sheng Biography text here.