國立交通大學

多媒體工程研究所

碩 士 論 文

樹 的 模 擬 與 碰 撞 處 理

Animating Trees with Collision Handling

研 究 生：陳奕辰

指導教授：黃世強 教授

中 華 民 國 一 百 年 十 一 月

樹的模擬與碰撞處理
Animating Trees with Collision Handling

研 究 生：陳奕辰　　Student：Yi-Chen Chen

指導教授：黃世強　　Advisor：Sai-Keung Wong

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

November 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年十一月

樹的模擬與碰撞處理

研究生：陳奕辰　　指導教授：黃世強 教授

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

摘要

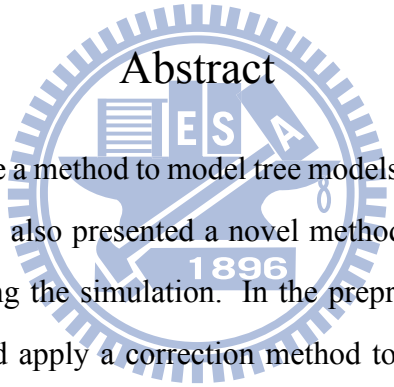
在這篇論文中，我們提出了一個建立樹模型的方法，並且以物理方法模擬樹的運動。我們也提出了一個新方法，處理模擬過程中樹模型自身的碰撞問題。在前置處理的過程中，我們隨機產生樹的模型，並且用矯正的方式避免樹葉跟枝幹彼此穿透。為了模擬樹在風中的運動，我們以物理方法模擬樹葉和樹枝的擺動。在模擬的過程中，我們收集所有在樹模型中發生碰撞的樹葉和樹枝，並對它們施加排斥力以避免它們互相穿透。我們也建立了一個針對樹模型的 k-DOP 階層架構，改善模擬的速度。從實驗結果來看，我們成功地讓樹在模擬過程中一直處在沒有穿透的狀態。

Animating Tree with Collision Handling

Student: Yi-Chen Chen Advisor: Sai-Keung Wong

National Chiao Tung University

Institute of Multimedia Engineering



Abstract

In this thesis, we propose a method to model tree models. A physically-based method is used to animate trees. We also presented a novel method to solve the collision problem of the tree models during the simulation. In the preprocessing phase, we generate the tree model randomly and apply a correction method to prevent the intersection between branches and leaves. To simulate the motion of a tree, we adopt a physically based approach for modeling the motion behavior of tree branches and leaves. At the runtime stage, we collect all the proximal primitive pairs and apply response force so as to avoid the penetration between leaves and branches. Besides, we adopt k-DOP hierarchies of the tree models to accelerate the collision detection process. The experimental results show that the tree models can be simulated in a collision-free state.

Acknowledgements

First, I would like to thank my advisor, Dr. Sai-Keung Wong for his leading in the past two years. He always taught me how to do a good research patiently and gave me many suggestions that help me a lot to complete this thesis. Also thanks that he cared about my health all the time and gave many advices for improving my health. I would also like to thank my thesis committee members, Dr. I-Chen Lin and Dr. Chuan-kai Yang, who evaluated my thesis.

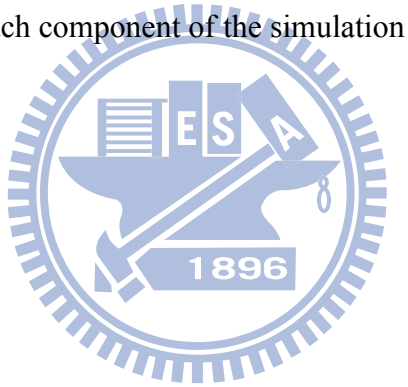
Thanks to all my lab mates for their comments and helps. They are so friendly so that I can have a pleasant learning environment. Finally, I would like to give many thanks to my family and all my friends for the giving of supports and encouragements when I was frustrated.

Yi-Chen Chen

October 2011

Contents

List of Figures



List of Tables

List of Algorithms

Chapter 1

Introduction

In computer animations or games, there are great efforts for producing beautiful scenes by showing or animating the natural trees. Thus, tree simulation is attracted much attention in computer graphics. Many approaches have been proposed to model trees or simulate their motion. Handling the interactions of a tree itself is a challenging problem due to the complex geometry of the tree which consists of thousands of branches and leaves. Many techniques do not handle collisions or simply apply collision detection to a tree with a simple structure.

In order to avoid the penetrations between leaves and branches, we need apply collision detection on the tree model. By handling collisions, we can simulate trees more realistic. Collision detection is an important technology in computer graphics. We briefly describe the collision detection method. In computer graphics, an object is composed by triangles. The position information of the triangles is used to compute whether or not there is any collision between objects. Collision detection is a time consuming job so that many methods were proposed to accelerate the performance of it. One of these methods is using bounding volume hierarchies (BVH). A bounding volume of an object is a closed volume and all triangles of the object are enclosed within the volume. A BVH is a tree structure. A leaf node of BVH is a bounding volume bound the basic element such as a triangle. Each inter node of the BVH is also a bounding volume which bound all the children nodes of it. If two nodes of BVHs do not overlap, the children nodes of these two nodes must

not overlap too. By traversing the BVHs of two objects, we can determine whether or not these two objects collide. Typically, a four-stage process is employed for performing collision detection: (1) construction of bounding volume hierarchies (preprocessing); (2) updating bounding volume hierarchies; (3) traversing pairs of bounding volume hierarchies and collecting potentially colliding pairs; (4) checking whether or not the elements of these pairs collide.

Our choice of bounding volume is discrete orientation polytope (k-DOP) [KHM+98] which is a convex polytope resulting from intersection of the half spaces bounded by k planes. k-DOP can bound a object more tight than other kinds of bounding volumes. We use k-DOP to approximately determine whether or not the leaves or the stems collide rather than check the triangles of the objects. We also construct a k-DOP BVH for the tree model which take leaves and stems as the basic elements to accelerate the performance of simulation.

1.1 Motivation

Simulating natural tree is an important topic in computer graphics. There are many applications of tree simulation in animations and games. For example, *Rio* is a famous animation movie which the main scene is located in the forest of Rio. There are full of trees in many scenes of the movies. We may take a close look of trees in some situations such as the camera following a bird shuttling within the branches or watching a worm climbing on a branch. We can notice the penetrations between the leaves and branches if the collisions are not processed. Due to the complex geometry of the tree, few of researches handle the collisions within the tree. Our focus is on modeling tree without intersection and the collision handling of the tree model. We handle collisions within the tree model and keep the frame rate acceptable.

1.2 Overview

We define the leaves and the stems as the primitives of the tree. Our method is divided into several stages. In the preprocessing stage, the tree model and the k-DOP BVH for it are built. In each time step of the simulation, we update the k-DOP BVH first. Then, the potentially colliding pairs of the tree models are found using k-DOP BVH. We give response force to resolve collisions for the colliding pairs. After that, the force exerting on the stems and leaves are computed. We use the computed force and the response force to compute the future position of primitives. Finally, we update the position of primitives to the next time step. The process is shown in Figure 1.1.

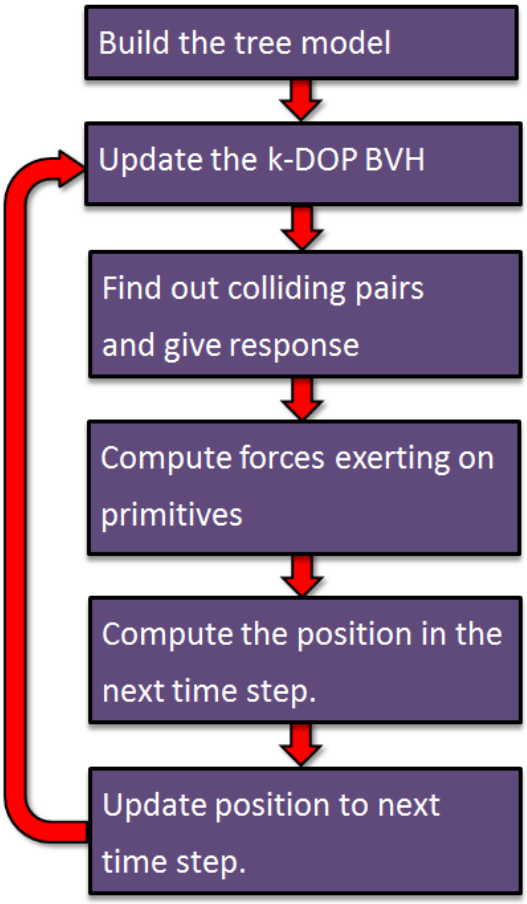


Figure 1.1: The process of our method.

1.3 Contribution

Our major contributions are:

1. We propose a method for tree modeling. We make sure that there is no intersection between leaves and branches.
2. We propose a simple method to avoid the penetrations between leaves and branches of the tree during simulation process.
3. We present a novel k-DOP BVH for the tree model to improve the performance.

1.4 Organization

The remaining chapters of the thesis are organized as follows. Chapter 2 reports the related works about the researches of tree. Later, our tree modeling method is present in Chapter 3. Chapter 4 describes the method animating the tree model. Chapter 5 introduces our approach for handling the collisions during simulation, and we show our experimental results in Chapter 6. Finally, Chapter 7 presents the conclusion and future work.

Chapter 2

Related Work

Many methods have been proposed for modeling trees. Lindenmayer and Prusinkeiwicz applied L-Systems to build tree models [PL91]. Oppenheimer used the geometric notion of the fractal self-similarity to present a fractal model of branching objects and it used bump mapping to simulate the texture of the bark. [Opp86]. After that, some researches took the influences of the environment into account for generating the tree model. In [HBM03], Hart, Baker and Michaelraj calculated the mass of each branch during the process of the growth of the tree to balance the tree model and phototropism was introduced as the parameter in the calculation of the growth process, the shape of the tree model was also affected by the sunlight. In [VHFBVR04], the environment illumination was took into account in the growth simulation by calculating intensity and mean direction of the light sources.

In order to simulate the interactions of plants with the environment, multifarious methods are introduced. Some researches used physically-based methods to simulate the animation, such as the methods introduced by [WWG06], [AK06], [ZST+06], [SO99] and [DCL+09]. However, physically-based methods may take a lot of computations because these methods need to compute the forces and integrate dynamical equations over time.

Weber and Penn simulated the wind sway effect by re-computing the rotation angles of the stem according the wind-speed every frames [WP95]. A hybrid approach was investigated in [GCF01] which a physics-based method was performed if it was needed or a

procedural method was performed. In [SZCZ05], the bending effect of the branches was achieved by the method called vertex weighting and the branch angle could be gotten by the equations they proposed.

Some researches were based on the spectral methods. For example, a tree motion analysis was performed for capturing the motion patterns of trees in [Sta97] and [DRBR09]. In [OFT+03], a technique was presented for generating the leaf and branch motions based on "1/f beta noise" which were derived based on the observation of a variety of natural phenomena. Later in [OTF+04], the spring model was added with the method based on "1/f beta noise" to enhance the reality of the motions. In [CGZ+05] and [HKW09], the motions of the branches were pre-computed in frequency space.

Some other researches were trying to generate the motion data in the preprocessing stage. In [VHDFVR06], the motion of the tree was derived from a small set of motion data got from a physically-based driven tree animation. Later, a data-driven approach that synthesized tree animations from a set of pre-computed motion data was present in [ZZJ+07].

Wong focused on modeling the small plants to present the softness of the foliage by using several segments to construct the foliage. [WD04]. In [HFC09], a pseudo-dynamics model that can handle motion of a curved beam in turbulence and the deformation of the leaves was presented. The computations took the influence of natural frequency and damping ratio into account. In [YSWS09], a frequency decomposition approach was proposed for dynamic animation of tree models. The motion of tree branches was simulated in a low frequency, and the motion of leaves was simulated in a high-frequency in the angular shell space. In [DCL+09], the wind model was modeled by the stastical curves to make the animation more realistic. In [OK10], the Navier-Stokes equations which described the motions of fluid substances were used to model the wind field.

Besides the animation topic, there were researches about the interactions of the tree itself or the interactions with the outer objects. In [LGF+06], a method was proposed for combining spatial decomposition and oriented bounding box to solve the self-collision problem of the broad-leaf plant. By restricting the rotation angle of the stems, many col-

lisions were avoided. In [Web08], a fast method was developed for employing separable projections and streamlined mechanics to simulate the motion of tree, and could handle the collisions with solid obstacles. The interaction between tree branches and raindrops was modeled in [YHYW10].

There were many researches accelerated the performance by implementing on GPU or employing the parallel technology. In [HKW09], the performance was improved by implementing in the vertex shader. In [Web08], the system could be divided into two independent parts so that they used CPU to process these two parts parallelly. In [OK10] and [YSWS09], the process was accelerate by using CUDA which is a parallel computing architecture of GPU.

Chapter 3

Tree Models

In order to simulate the tree in a non-penetration state, we should avoid the situation that the leaves and the branches intersect initially. In this section, we present our approach for building up the tree models which ensure there is no intersection between branches and leaves. We will introduce the structure of a branch model and a leaf model first and then introduce the process of modeling tree. After introducing how to model trees, we will introduce a voxel-based acceleration method which is used to accelerate the process of modeling tree. The detail will be described in the following sections.

3.1 The structures of Branches and Leaves

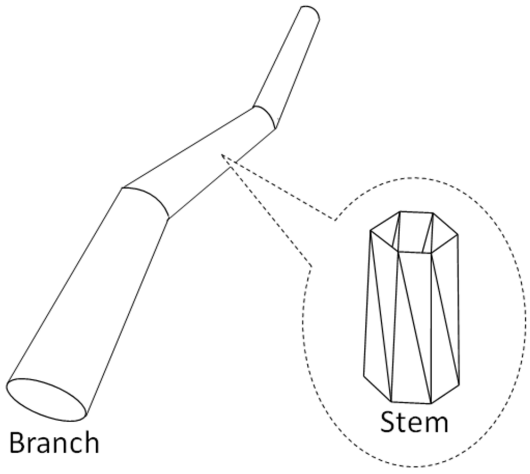


Figure 3.1: The structure of a branch.

We employ the structure of the branch and the leaf pattern proposed in [WWG06]. It is easy to implement and suitable to apply to our method for collision handling. As shown in Figure 3.1, a branch is composed by several stems. Each stem is modeled as a cylinder which is discretized into triangles as shown in Figure 3.2.

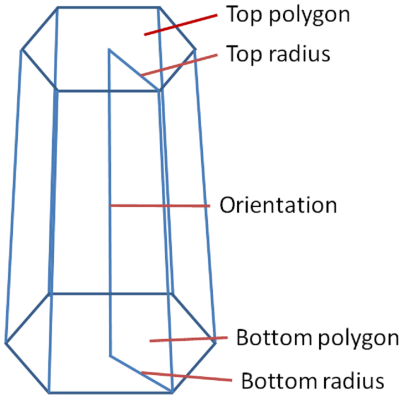


Figure 3.2: The structure of a stem.

We use the width information of the stem to form the points of the top polygon and the bottom polygon as shown in Figure 3.2. These points will be used to form the triangles. We can control the number of edges of the top polygon and the bottom polygon to decide the fineness of the cylinder.

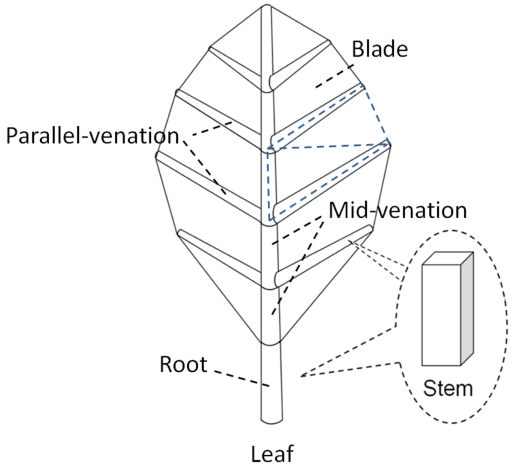


Figure 3.3: The structure of a leaf.

The structure of a leaf model is shown in Figure 3.3. The venations of the leaf are

composed by stems. We model the blade of the leaf by a set of triangles formed by the endpoints of the venations. The stem that connects the branch and the leaf blade is called root of the leaf. By this kind of modeling method, we support with irregular leaf patterns.

3.2 Tree Modeling Process

The overall process is divided into three-stages: (1)building branches; (2)building leaves; (3)removing redundant stems. We adopt a breadth-first method to generate branches. In the beginning, the root of the tree is generated at depth one. Then we use the stems of the current level to generate child stems at the next depth level. The information of the child stem is computed based on the information of parent with certain fuzziness such as width, length, weight and some parameters used in the dynamic system. The magnitudes of parameters of a stem are usually smaller than the ones of its parent. The orientation of the child is determined by rotating the orientation of parent a small angle. In order to prevent the branches growing in an abnormal lightning shaped manner, we limit the angle between a child and its parent smaller than 30 degrees. Each stem will generate one to three children randomly. If the number of the children more than one, we increase the distance between the tips of the children by increasing the length of the generated children to make sure there has enough room to build the next generation of the children. In this way, the chance is decreased for the next generations colliding with each other. Figure 3.4 shows the structure of a portion of the tree.

When a new stem is created, we check whether or not the new stem collides with the existing stems. To do so, we bound each stem with a k-DOP when the stem is built. If the k-DOP of the new stem $stem_{new}$ overlap with the k-DOP of any other stems $stem_{exist}$, the triangles of $stem_{new}$ and $stem_{exist}$ should be further check if there is any intersection between them. If so, $stem_{new}$ is removed from the tree. In the case that the radius of the removed stem is larger than a threshold, there will be an obvious incision at the tip of the branch. To avoid it, we traverse from the removed stem back until a branching stem is encountered and we removed the visited stem as shown in Figure 3.5.

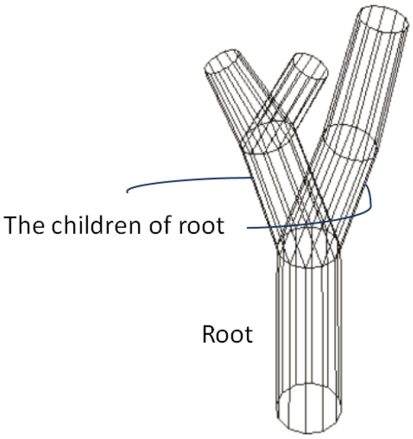


Figure 3.4: The structure of a tree without leaves.

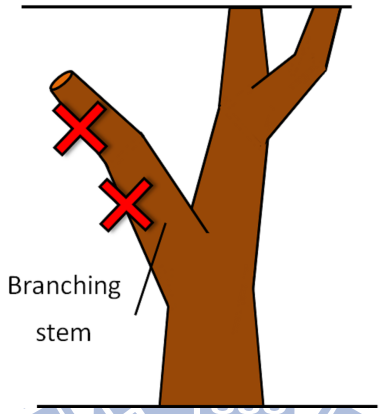


Figure 3.5: The schematic diagram of deleting the useless stem.

Notice that collision detection is not performed between the parent and siblings of $stem_{new}$. This is because they are connected with each other. When the width of the stem is smaller than a threshold, we treat this stem as the start point of the small branches $stem_{sb}$. The width, weight and rigidity of the offspring of $stem_{sb}$ are set much smaller than the ones of $stem_{sb}$. The leaves will be built on the tip of the small branches. we mark the stems which may generate leaves.

After generating the branches, we pick out those stems which have been marked to generate leaves. The orientation and the size of a leaf are also affected by its parent stem. The orientation of mid-venaitons are computed first by rotating the orientation of parent stem a small angle. Then we use the cross product of the orientation of parent stem and the orientation of leaf root to determined the orientation of parallel-venations. After building

the venations of the leaf, the tip of the venations are used to form triangles of the blade. The leaf is divided into two parts: root and remaining portion. In doing so, we can select the parts of a leaf for collision detection. For example, we do not perform collision detection between the root of the leaf and the parent stem of the leaf because they are connected. But the parent stem of the leaf and the blade of the leaf should be applied collision detection. Similarly, we do not perform collision detection between the root of the leaf and the siblings of the root. For this reason, we bound the two parts of the leaf with k-DOP individually. Like stem, when a leaf is created, we check if there is any intersection between the new leaf and the existing primitives. If the new leaf intersects with others, we delete it. If not, we add it to our tree model. Figure 3.6 shows the appearance of our tree models with 8606 stems and 7821 leaves. The algorithm of generating branches and leaves are shown in Algorithm 1 and Algorithm 2.

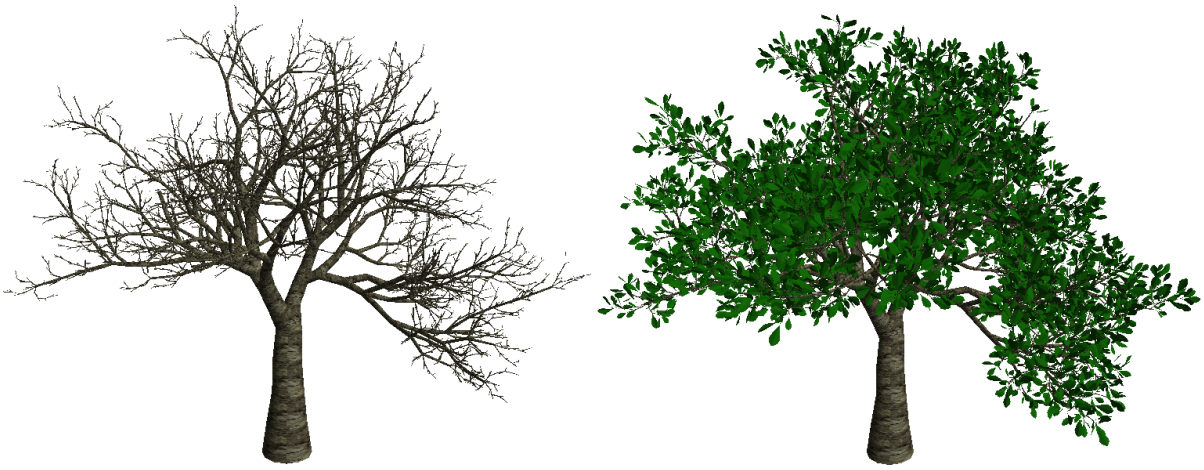


Figure 3.6: The tree with/without leaves.

After building the branches and leaves, we need a follow-up process. Some leaves do not build successfully due to intersection, There are some marked stems do not have any leaves. These stems are small and do not affect the appearance of the tree. For this reason, we removed marked stems to increase the performance of our system. We traverse back from the removed stem until a branching stem is encountered and all the visited stems are removed.

Algorithm 1 The algorithm of generating branches

create two list: $generation_{current}$ and $generation_{next}$
define $width_s$ as the smallest width of stems
define $width_{sb}$ as the threshold of generating small branches.
create a root of the tree and push the root into $generation_{current}$
while The size of $generation_{current} > 0$ **do**
  for i = 0 to the size of $generation_{current}$-1 **do**
    if the width of $generation_{current}$[i] $< width_s$ **then**
      continue
    end if
    decide the number of the children num_{child} of $generation_{current}$[i]
    for j = 0 to num_{child}-1 **do**
      set up the information of child[j]
      build child[j]
      build the k-DOP of child[j]
      if child[j] collide with existing stems **then**
        delete child[j]
      else
        if the width of child[j] $< width_{sb}$ **then**
          set child[j] as the start point of small branches.
        end if
        push child[j] into $generation_{next}$
      end if
      if $generation_{current}$[i] do not have child and is a stem of small branches **then**
        mark the stem to indicate it may generate leaves.
      end if
    end for
  end for
  set $generation_{next}$ as $generation_{current}$.
end while

Algorithm 2 The algorithm of generating leaves

1: $stems_{mark}$: the stems may generate leaves
2: $leaf_{num}$: the max number of leaves that a stem can generate.
3: **for** each stem in $stems_{mark}$ **do**
4: **for** i = 0 to $leaf_{num}$ **do**
5: set the information of the leaf[i]
6: build the skeleton of the leaf[i]
7: build the triangles of the leaf[i]
8: build the k-DOP of the leaf[i]
9: **if** leaf[i] collides with existing primitives of the tree **then**
10: delete leaf[i]
11: **else**
12: push leaf[i] into the leaf list of the tree
13: **end if**
14: **end for**
15: **end for**

3.3 Voxel-based Acceleration Method

In the modeling process, we check collisions between the new generated primitive and each exist primitive. Because the cost of the collision detection, the time of generating tree models can be up to several minutes. We use a voxel-based acceleration method to accelerate the modeling process.

A voxel is a volume element, representing a value on a regular grid in three dimensional space. We partition a space that is big enough to enclose any kind of tree models we generate into a set of regular voxels. All primitives of the tree will be registered in one of these voxels. Each voxel is associated with two lists. One list records the stems that the center points of the centerlines are inside the voxel and the other one records the leaves that the center points of the mid-venations are inside the voxel.

The size of a voxel is decided according to the size of the biggest primitive. In our

implementation, a voxel is a cube and it is big enough to bound the biggest primitive. As shown in Figure 3.7, we use half the length of the centerline of the biggest stem and the bottom radius of the biggest stem to compute a length SR. The length of the edges of a voxel is the double of SR. In this way, we make sure that no matter where the primitive is located in the registered voxel, the portion of the primitive that is beyond the registered voxel can be bounded by half the adjacency voxel. Therefore, if two voxels $voxel_a$ and $voxel_b$ are not adjacency, the primitives which are registered in $voxel_a$ and $voxel_b$ are collision free (Figure 3.7).

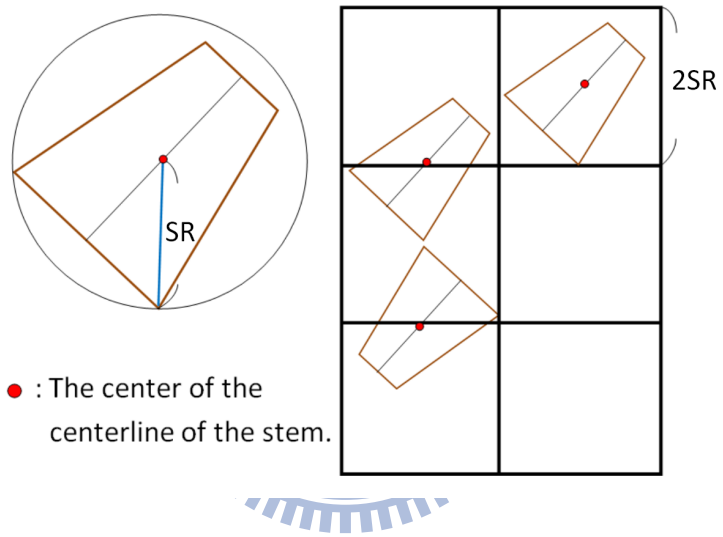


Figure 3.7: The conventional diagram of the voxel structure in 2D. Two primitives must be collision free if the voxels they are registered are not adjacent.

We define the voxel that contain the new created primitive $primivite_{new}$ as $voxel_{reg}$, and define the voxels that are adjacent to $voxel_{reg}$ as $voxel_{adj}$. We can use the position of $primivite_{new}$ to get $voxel_{reg}$ and $voxel_{adj}$ from the voxel array conveniently. If we want to know whether or not $primivite_{new}$ collide with others, we only need to check collisions between $primivite_{new}$ and the primitives that are registered in the voxel $voxel_{reg}$ and the voxels $voxel_{adj}$. By this way, we reduce the cost of the collision detection in the modeling process.

Chapter 4

Animation of Tree Models

The movement of a tree is achieved by animating all stems. The movement of a stem is caused by the rotation of its parent and the force exerting on it. We compute the amount of the force exerting on the stem first. Then we use the force to get the rotation amount of the stem. After that, we update the orientation of the stem from root to the tip of branches. We refer the method introduced in [WWG06] and make some modifications to animate our tree models. In the following sections, we will describe the method in details.

4.1 Force Computation

The movement of the branch is caused by the rotation of stems that composed it. To compute the amount of the rotation of the stem, we need to compute the forces exerting on the stem first. The five different kinds of forces including wind force, gravity force, restoration force, transfer force and response force are considered.

Wind force $\vec{F}_W$ and gravity force $\vec{F}_G$ are the forces come from the environment. We simply define a direction vector as the wind force. The gravity force is calculated using the following equation:

$$\vec{F}_G = m\vec{g}, \tag{4.1}$$

where m is the mass of the stem, and $\vec{g}$ is the gravitational constant.

Restoration force $F_{Restore}$ is an internal force which attempts to recover the orientation

of a stem to its rest position. Our manner for computing restoration force is different with the method introduced in [WWG06]. Here we use the spring force to achieve the restoration effect.

$$\vec{F}_{Restore} = -k_r \frac{P - P_{rest}}{||P - P_{rest}||}, \tag{4.2}$$

where P is the current position of the end point of the stem, K_r is the rigidity of the stem and P_{rest} is the rest position of the end point of the stem. P_{rest} changes as the parent of stem rotates. That is, the stem try to keep the angle between the stem and the parent of the stem the same with the one that the tree just created.

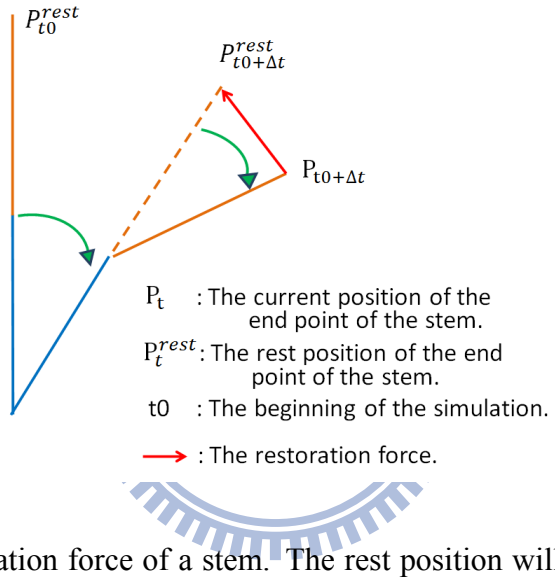


Figure 4.1: The restoration force of a stem. The rest position will change in every time step.

In a nature environment, the movement of the children of the stem should also affect the stem. Therefore, the transfer force $\vec{F}_T$ is computed which is used for the stems affecting the movement of their parents.

$$\vec{F}_T = K_t K_r \sum_{i=1}^{n} \left(\vec{F}_W^i + \vec{F}_G^i + \vec{F}_T^i \right), \tag{4.3}$$

where K_t and K_r are constants for representing transitivity and rigidity, respectively; n is the number of children of the current stem. The two forces $\vec{F}_W^i$ and $\vec{F}_G^i$ are the wind force and the gravity force of their i-th child. The force $\vec{F}_T^i$ is the transfer force of the i-th child caused by its children.

The last kind of force response force $\vec{F}_{Response}$ is used to separate the colliding primitives. The response force will be introduced together with our collision handling method in the later chapter.

Because the skeleton of a leaf is composed by stems like branch, the computation methods of most forces are equal to the ones used for branch. But there is still a little difference due to their different shapes. The transfer force is ignored in a leaf because the influence of the transfer force is negligible in the leaf. The computation of the wind force $\vec{F}_W^{venation}$ exerting on the venations should involves with the normals of the triangles of the leaf because the affected area of a leaf is quite different if winds come from different directions.

$$\vec{F}_W^{venation} = C_W \vec{n}\vec{w} \cdot \vec{n}, \tag{4.4}$$

where C_W is a user defined constant for controlling the magnitude of the wind force, $\vec{n}$ is the normal vector of the end point of a venation and $\vec{w}$ is the wind direction vector.

The total force exerting on the stem (include the venations of leaves) without collisions is the sum of the forces:

$$\vec{F}^{stem} = \vec{F}_W + \vec{F}_G + \vec{F}_T + \vec{F}_{Restore}. \tag{4.5}$$

4.2 Motion Computation

We compute the angular displacement using the force we get in last section. First, the torque $\vec{\tau}$ is computed using the equation:

$$\vec{\tau} = \left(\vec{ori} * len\right) \times \vec{F}^{stem}, \tag{4.6}$$

where $\vec{ori}$ is the unit orientation of the stem and len is the length of the stem.

Next, we get the angular acceleration $\vec{\alpha}$ and the angular velocity $\vec{\omega}$ by the following equations :

$$\vec{\alpha} = \frac{\vec{\tau}}{m * len^2}, \tag{4.7}$$

$$\vec{\omega}_{t+\Delta t} = \vec{\omega}_t + \vec{\alpha} * \Delta t, \tag{4.8}$$

where $\vec{\omega}_{t+\Delta t}$ is the angular velocity of the stem at next time step, and $\vec{\omega}_t$ is the angular velocity at current time step. It should be noted that the angular velocity $\vec{\omega}^{stem}$ decreases at each time step due to damping effect.

$$\vec{\omega}_t = \vec{\omega}_t * (1 - K_d)\,, \tag{4.9}$$

where K_d is the user defined constant representing damping factor.

Finally, the angular displacement $\Delta\vec{R}$ is calculated by the following equation:

$$\Delta\vec{R} = \vec{\omega}_t * \Delta t + \vec{\alpha} * \Delta t^2 + \Delta\vec{R}_{parent}, \tag{4.10}$$

where $\Delta\vec{R}_{parent}$ is the angular displacement of the parent of current stem. The orientation of all stems (including venations of leaves) are updated using $\Delta\vec{R}$.

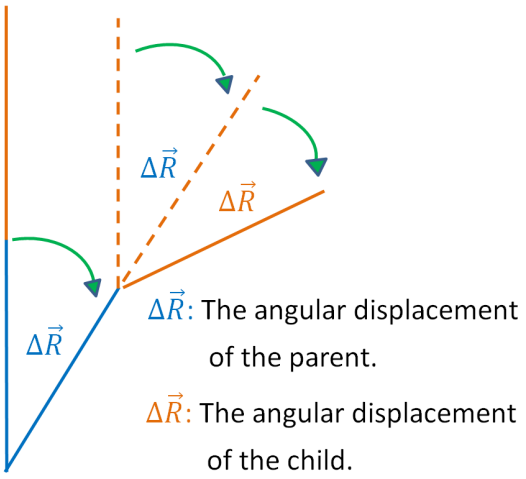


Figure 4.2: The movement of the stems. The total angular displacement of the child is the sum of the angular displacement of the parent and the angular displacement of itself

4.3 Parallel Processing

We employ CPU multi-thread technique to accelerate the computation of the animation. In the thesis, we use four threads to accelerate the simulation.

The tree is divided into two parts: branches and leaves. For branches, there is a property that the stems do not have parent-child relationship are independent. We use this

property to update the information of branches parallel. We choose the stems in the same branching level as the start points of the updating process of threads because the offspring of these stems are independent as shown in Figure 4.3. To make sure the number of the start points of the updating process is more than the number of the threads, we traverse from the root of the tree until reach the branching level which has a sufficient number of stems.

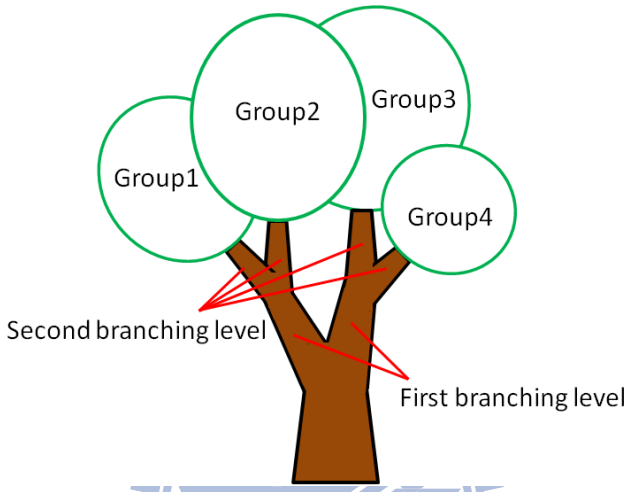


Figure 4.3: The independent relationships of the branches. Group1, group2, group3 and group4 are independent with each other.

For leaves, the information of each leaf is independent to the other leaf. We divide leaves into four groups and distribute them to the four threads.

In each time step, we update the information of the stems from the root to the start points of the traversal using single thread. After that, four threads are used to update the information of the remaining stems. After updating the branches, the leaves are updated parallel.

Chapter 5

Collision Handling

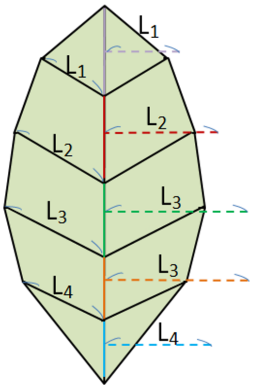
5.1 Collision Detection

In order to keep the tree model in a collision-free state during simulation, collision detections are applied. The basic method to do this job is checking if there is any intersection between the pairs of triangles of two primitives. But it is a time consuming job. Instead of checking the pairs of triangles of two primitives, we use the distance between the collision primitives and the kDOPs of them to approximately determine whether or not they collide.

Two primitives that are collision free in the current time step may penetrate in the next time step. To prevent this situation, we compute safe distance for each primitive to detect collisions earlier. If the shortest distance of two primitives is smaller than the sum of the safe distance of two primitives, then it implies that these two primitives may collide in the next time step. We define the shortest distance between two primitives as the shortest distance between the centerlines of the mid-venations of the leaves (or the centerline of the stems).

The calculation of the safe distance is described as follows. The safe distance of a stem is the sum of the bottom radius of it and the buffer distance computed by us. The buffer distance is the max displacement of the primitive in a time step. In our implementation, we limit the max amount of the angular displacement of the primitive in a time step. We use

this max angular displacement to compute the max displacement of the primitive in a time step. The safe distance of a leaf is computed depending on the different situation because the shape of a leaf is irregular. The stem of mid-venations mid_{sd} that has the shortest distance with the other primitive will be found out first. Each stem of mid-venations is connected with two or four parallel venations. The longest length par_{long} of these parallel venations which is connected with mid_{sd} is recorded. The safe distance of the leaf is the sum of par_{long} and the buffer distance of the leaf (Figure 5.1).



L: The length of the parallel venations,
$L_3 > L_2 > L_4 > L_1$

Figure 5.1: The safe distance of the leaf.

Two primitives are treated as a colliding pair when meet the following conditions: (1) The k-DOPs of two primitives overlap. (2)The shortest distance between two primitives is smaller than the sum of the safe distance. The accuracy of the collision detection is determined by the safe distance and the enclosed range of the k-DOPs. When the k-DOPs of two primitives overlap, we further check if the distance between primitives is smaller than the sum of the safe distance. If so, we treat these two primitives as a colliding pair. It is worth mentioning that even the second condition is satisfied, we do not apply collision detection to the primitives if the k-DOPs do no overlap. In this way, we cannot detect collisions earlier using safe distance. To make safe distance has sense, the k-DOPs are enlarged by changing the minimum and the maximum value of each k-DOPs axis. We increase the maximum value by adding the buffer distance of the primitive to the value, and decrease the minimum value by subtracting the buffer distance of the primitive to the

value.

After collecting the primitives which about to collide, we give these primitives response force to separate them. The computation method of the response forces is described in next section.

5.2 Collision Response

Two colliding primitives may not be separated by the response force in one time step because a primitive may affected by many primitives colliding with it. To stop two primitives keeping moving toward each other, we need a force which is inversely proportional to the distance between two primitives. For this reason, we use a spring force to give response. The amount of the force is decided according to the distance between two primitives if the current distance is smaller than the rest length of the spring.

We set the rest length of the spring as the sum of safe distance of two colliding primitives. We take the shortest distance between the colliding primitives as the current length of spring. The two vectors that formed by two centers of the k-DOPs of the primitives are set to be the directions of the response force (Figure 5.2).

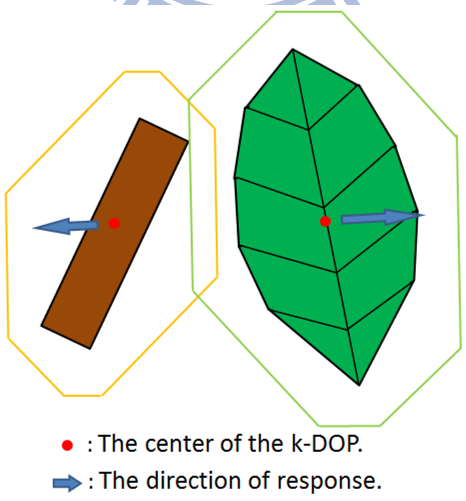


Figure 5.2: The direction of the response force.

Our response force is addressed by the following equation:

$$\vec{F}_{Response} = \vec{Dir}_{Response} * K_s * (Dis_{shortest} - Dis_{safe}), \tag{5.1}$$

Where K_s is the user defined constant to adjust the amount of the force, and $\vec{Dir}_{Response}$ is the direction of the response force. $Dis_{shortest}$ and Dis_{safe} represent the current distance between two primitives and the rest length of the spring respectly.

Even if the direction of the response force is correct, the parents of the primitives may pull the primitives so that the primitives still move in wrong direction. To solve the problem, the response force is also applied to the parents of the primitives so that we can decide at the parent to let the primitives separate. The response force is transferred until reach the common ancestor of two colliding primitives as shown in Figure 5.3.

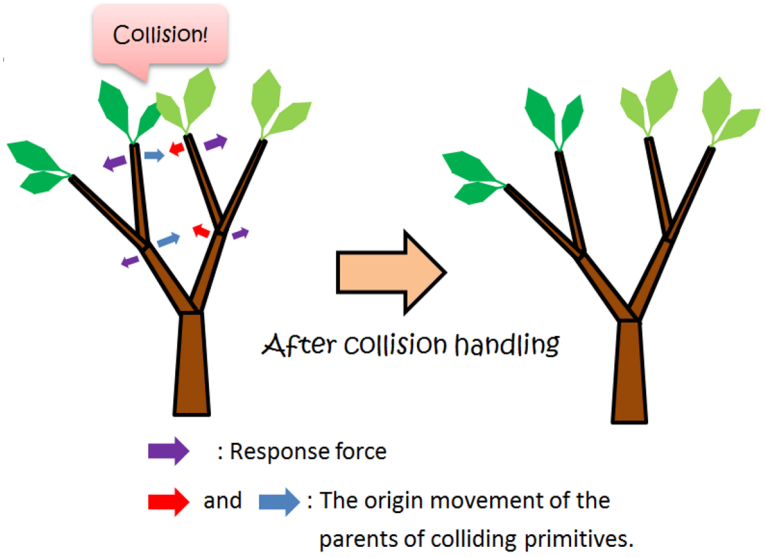


Figure 5.3: The response force is transferred to the parent of the colliding primitives.

If the stem is colliding with the other primitive, the total force of the stem and the ancestors is:

$$\vec{F}^{stem} = \left(\vec{F}_W + \vec{F}_G + \vec{F}_T + \vec{F}_{Restore}\right) * P_1 + \vec{F}_{Response} * P_2, \tag{5.2}$$

where P_1 and P_2 are the weights to adjust the percentage between the original force and the response force. If the stem is far from the colliding primitives, the value of P_1 increases and the value of P_2 decreases. Otherwise, the value of P_1 decreases and the value of P_2 increases. The value of P_1 and P_2 can be set according to the preferences of the users.

5.3 The Acceleration Structure

Even our method only need to check whether or not the k-DOPs overlap, the computation amount is still very large if we check collisions between primitives one by one. So we construct a k-DOP BVH for the tree models to accelerate the process.

In general, there are two methods to consturct BVH: top-down and bottom-up method. Top-down methods proceed by partitioning the input set into two subsets, bounding them in the chosen bounding volume, then keep partitioning (and bounding) recursively until each subset consists of only a single primitive. Bottom-up methods start with the input set as the primitives and then group two of them to form a new internal node, proceed in the same manner until everything has been grouped under a single node. In our method, we design a BVH for the tree models which is different with the BVH generated by general method. We build the BVH based on the structure of the tree rather than based on the distribution of the primitives. For the animated tree, there is a property that the offspring of a stem will move with the stem. To reduce the case that the k-DOP is enlarged due to the relative movement of the primitives, we group the offspring of a stem into a k-DOP. By doing so, we decrease the chance that two connected primitives are bounded by different k-DOPs. We define the stem that all offspring are bounded in a k-DOP as a representative stem as shown in Figure 5.4.

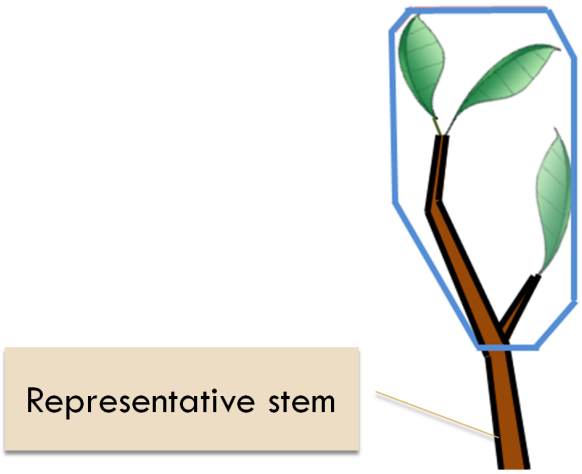


Figure 5.4: Representative stem.

In our approach, only the representative stem and the children of it are the connected primitives that are bounded by different k-DOPs. Therefore, the overlap between the k-DOP of the representative stem and the k-DOP bounds the offspring of this representative stem is ignored. We find all representative stems first and then use these stems to form the BVH from the root of the tree.

To reduce the time of building BVH, some representative stems are already predefined during the modeling process. These predefined stems will be used to find the other representative stems. The stems we pick as representative stems during the modeling process are: (1)the stems $Rstems_{b1}$ that the width are first smaller than the user defined threshold; (2)the stems $Rstems_{b2}$ that treated as the start point of small branches; (3)the stems $Rstems_{b3}$ have leaves. The stems of $Rstems_{b2}$ are the offspring of the ones of $Rstems_{b1}$. Therefore, the k-DOPs bounds the offspring of $Rstems_{b2}$ must be the child nodes of the ones bounds the offspring of $Rstems_{b1}$. For the same reason, the k-DOPs bounds the offspring of $Rstems_{b3}$ must be the child nodes of the ones bounds the offspring of $Rstems_{b2}$. The leaf node of the hierarchy is the k-DOPs of the stems and the leaves. A leaf node that bounds the stem may be included by the k-DOPs at any levels but the one bounds the leaf must be included by the ones bounds the offspring of $Rstems_{b3}$.

We use $Rstems_{b1}$ to find the representative stems of higher levels. We traverse back from the stems of $Rstems_{b1}$ to find the ancestors which are the branching stems. The parents of these branching stems are treated as the candidate $stems_c$ of new representative stem. Because the stem with the biggest width must be the common ancestor of most stems, we choose the stem with biggest width $stem_{bw}$ in $stems_c$ as new representative stem. We mark the stems in $Rstems_{b1}$ which are offspring of the $stem_{bw}$ to indicate they will be bounded by the new k-DOP. Then we keep using the stems in $Rstems_{b1}$ which are not marked to find new $stems_c$ and take the stem with biggest width as new representative stem. The finding process for the representative stems in the new level are done if the stems of $Rstem_{b1}$ are all marked. The algorithm of finding the representative stems is shown in Algorithm 3.

Algorithm 3 The algorithm of finding the representative stems in each levels

1: $Rstems_{cl}$: the representative stems of current level
2: $Rstems_{nl}$: the representative stems of new level
3: $stems_c$: the list contains the candidates of the representative stems of new level
4: num_{hl}: a constant limits the number of the representative stems of highest level.
5: **while** true **do**
6: **while** There is a stem in $Rstems_{cl}$ not be marked **do**
7: **for** each stem in $Rstems_{cl}$ **do**
8: find out the ancestor of the stem which is a branching stem $stem_{br}$
9: push the parent of $stem_{br}$ into $stems_c$
10: **end for**
11: choose the stem with biggest width $stem_{bw}$ in $stems_c$ and push it into $Rstems_{nl}$
12: mark the stems of $Rstems_{cl}$ which are offspring of $stem_{bw}$.
13: **end while**
14: set $Rstems_{nl}$ as $Rstems_{c1}$
15: **if** The size of $Rstems_{cl}$ is smaller than num_{hl} **then**
16: exist the while loop
17: **end if**
18: **end while**

We use these representative stems to build the BVH. To form the k-DOP that bound all the offspring of the representative stem, we traverse from the representative stems of the highest level and we collect the child nodes of the k-DOP in a depth-first manner. We push the k-DOPs of the visited primitives into the child list of the new k-DOP. If the visited primitive is the representative stem of next level, we build the k-DOPs of next level and push it into the child list first. Then, we stop traversing deeper and returning to the most recent node that have not been visited. After the collection of the child nodes for the new k-DOPs is done, we use the information of these child nodes to build the new k-DOPs. The structure of the BVH is shown in Figure 5.5. The algorithm of building the BVH is shown in Algorithm 4. We can use this BVH to find collision pairs efficiently.

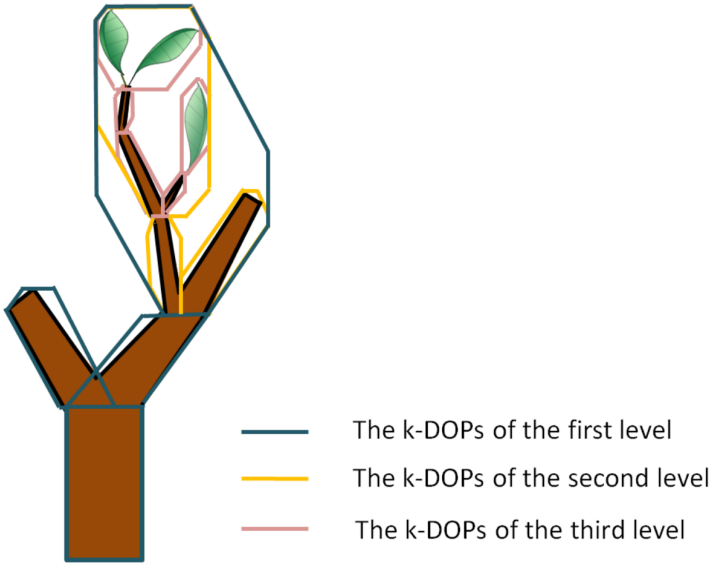


Figure 5.5: The BVH of the tree.

Algorithm 4 The algorithm of building BVH

1: $Rstems_{hl}$: the representative stems of highest level
2: **for** each stem in $Rstems_{hl}$ **do**
3: traverse in depth-first manner.
4: **while** the process of the traversal is not finished **do**
5: push the k-DOPs of the visited primitives into the child list.
6: **if** the visited primitive is the representative stem of next level **then**
7: build the BVH of next level.
8: push the root of the BVH of next level into the child list
9: stop traversing deeper and return to the most recent node that have not been visited
10: **end if**
11: **end while**
12: use the information of the child nodes to build the k-DOP of highest level.
13: **end for**

The manner of traversing BVH is described as follows. For each k-DOP, we maintain an overlap list to record the k-DOPs overlap with it. We choose a k-DOP $kDOP_{hl}$ in the highest level and collect the k-DOPs overlap with it in the same level. If the k-DOP which overlaps with $kDOP_{hl}$ is the one that bounds the representative stem of $kDOP_{hl}$, we do not push it into the overlap list. For each child of $kDOP_{hl}$, we collect the k-DOPs that overlap with it in the same level. These child nodes only need to be checked if there is any overlaps with the sibling of it and the child nodes of the k-DOPs of the overlap list of $kDOP_{hl}$,. After handling the children of $kDOP_{hl}$, we then go to handle next level. The process continues until the collections of the overlap list of all primitives bounded inside $kDOP_{hl}$ are done. These potentially colliding pairs are further checked whether the distance between primitives is smaller than the sum of the safe distance. If so, the response force introduced in the section 5.2 is applied to these colliding primitives. $kDOP_{hl}$ and all children of it will not be used in the following process of the traversal because the collision events that associated with the primitives inside the $kDOP_{hl}$ are solved. When all k-DOPs in the highest level are traversed, all collisions are handled.

The whole process is shown in Figure 5.6.

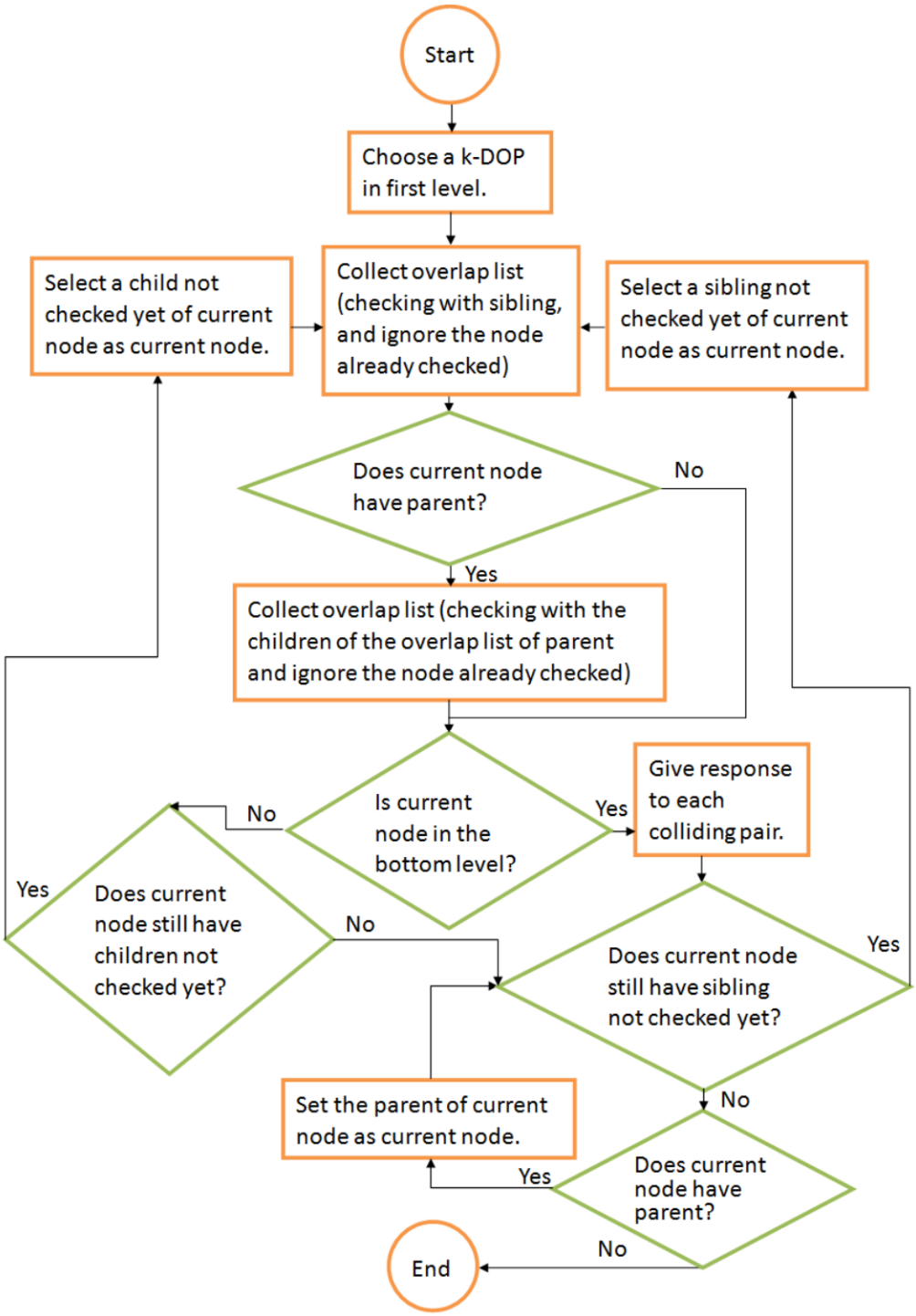


Figure 5.6: The process of collision handling.

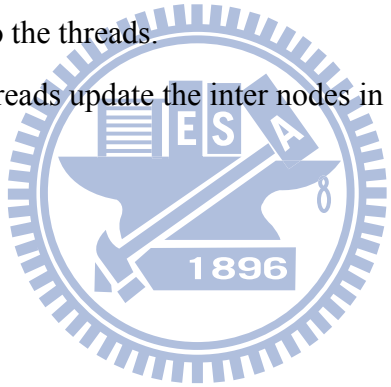
5.4 Parallel Processing

We employ CPU multi-thread technique to accelerate the process of updating BVH. We divide the BVH into two parts: leaf nodes and inter nodes.

The leaf nodes of the BVH are the k-DOPs of stems and leaves. In order to balance the workload of the threads, we update the k-DOPs of stems and leaves separately. We divide the k-DOPs of stems into four groups and distribute them to the threads to update parallel. Then the k-DOPs of the leaves are updated like the ones of stems.

The BVH is a tree structure. If there is no parent-child relationship between two nodes, these two nodes are independent. For this reason, the offspring of the first level k-DOPs are independent to each other. Therefore, we take the k-DOPs in the first level as the start points of the updating process. We divide the k-DOPs in the first level into four groups and distribute them to the threads.

In each time step, the threads update the inter nodes in a bottom-up manner after the leaf nodes are updated.



Chapter 6

Results

In the experimentation, we built and simulated some tree models by a computer with Intel(R) Core(TM)2 Quad Processor Q6600, 4.00GB RAM , NVIDIA Geforce 9600 GT GPU. We rendered the tree models using OpenGL. We compared the cost time of the simulations in detial in this chapter.

6.1 Tree Models

In the thesis, we built random produced tree models for simulations which the stems and the leaves do not intersect with each other. We used five tree models to do our experiment in this chapter. The screen shots of these five tree models from smallest one to the most complex one were shown in Figure 6.1. The other random generated tree models were also shown in Figure 6.2.

We recorded the cost time of each step in the preprocessing stage: (1)The time of building branches; (2)The time of building leaves; (3)The time of reducing redundant stems. (4)The time of building BVH. (5)The time of building voxels. Table 6.1 showed comparisons of these time between different tree models.

In Table 6.1, we compare the time of building primitives of the tree with and without using voxel-based acceleration method. The result showed that we successfully reduce the cost of the collision detection in the modeling process.

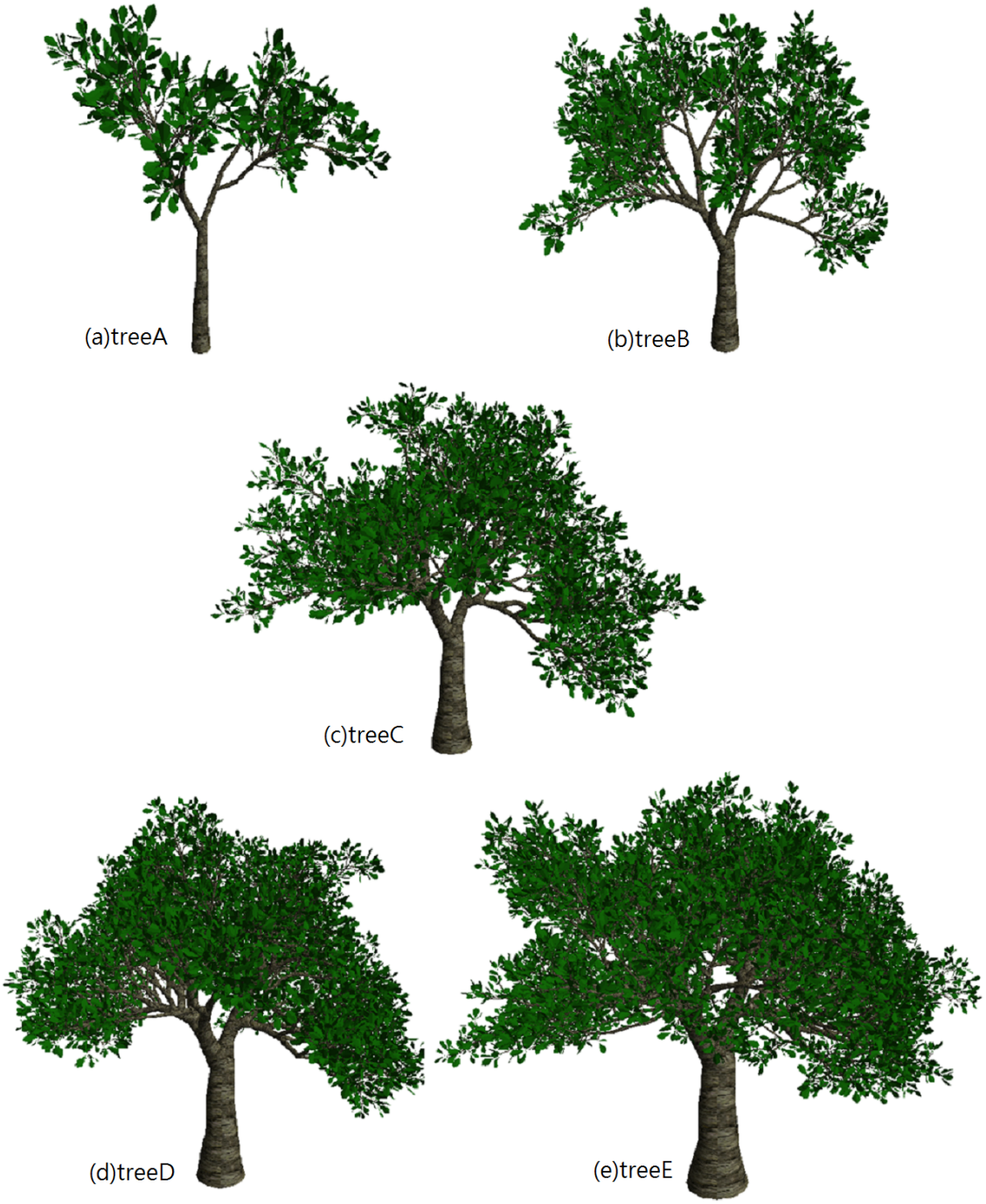


Figure 6.1: The tree models with different size.

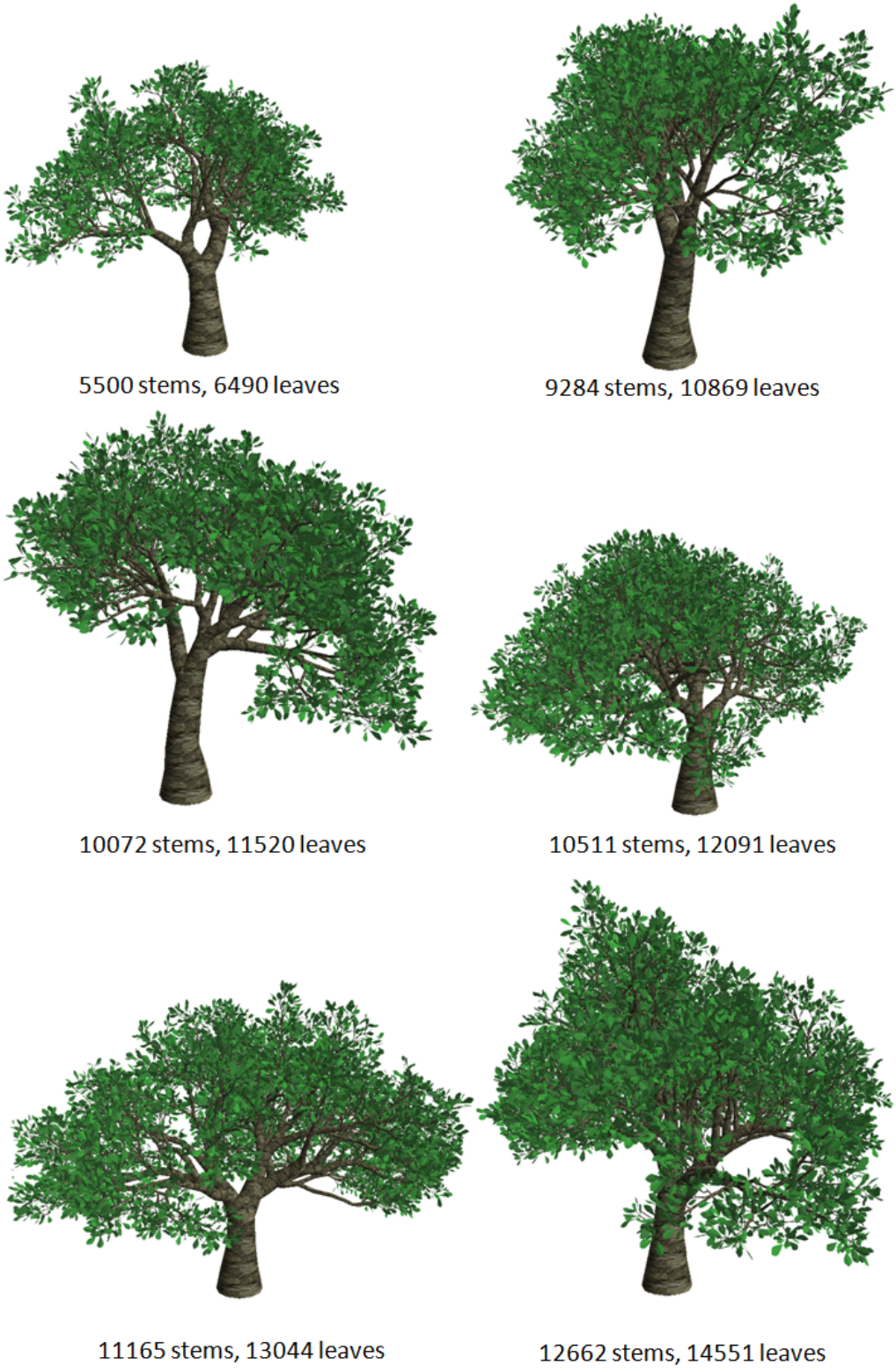


Figure 6.2: The random generated tree models.

		treeA	treeB	treeC	treeD	treeE
The numbers of stems		597	2721	6470	11134	14615
The numbers of leaves		749	3320	7821	12850	16607
BB	with acceleration	0.0194	0.106	0.347	0.1	1.838
	without acceleration	0.0232	0.241	1.541	9.185	17.985
BL	with acceleration	0.2991	1.569	4.625	13.922	29.571
	without acceleration	0.3257	3.498	29.282	91.032	153.309
RS		0.0005	0.003	0.011	0.015	0.021
BK		0.0076	0.032	0.085	0.129	0.171
BV		0.0019	0.0007	0.0004	0.0003	0.0002

Table 6.1: The time (sec) of preprocessing stage. BB: the time of building branches, BL: the time of building leaves, RS: the time of reducing redundant stems, BK: the time of building BVH, BV: the time of building voxels.

Because the space enclosed by voxels was fixed, the time of building voxels was depending on the size of a voxel. As the size of the tree became bigger, the size of a voxel became bigger and the total number of the voxels decreased. Therefore, the time of building voxels decreased as the size of the tree increased.

We observed that the cost of building leaves was the main cost in the preprocessing stage. This was because the k-DOPs of leaves had larger opportunity to overlap with other primitives than the ones of stems. The k-DOPs of these leaves overlap and these leaves are required for further check so as to determine whether or not the triangles of the leaves intersect with the other.

6.2 Tree Animation

The time step of our simulation was defined as 0.03. The uniform wind flow was defined as the external force field. We took some snapshots of the animating tree model under the effects of different winds. To see the swing direction of branches and leaves

clearly, the simulation snapshots of the simple tree model with 597 stems and 749 leaves was shown in Figure 6.3.

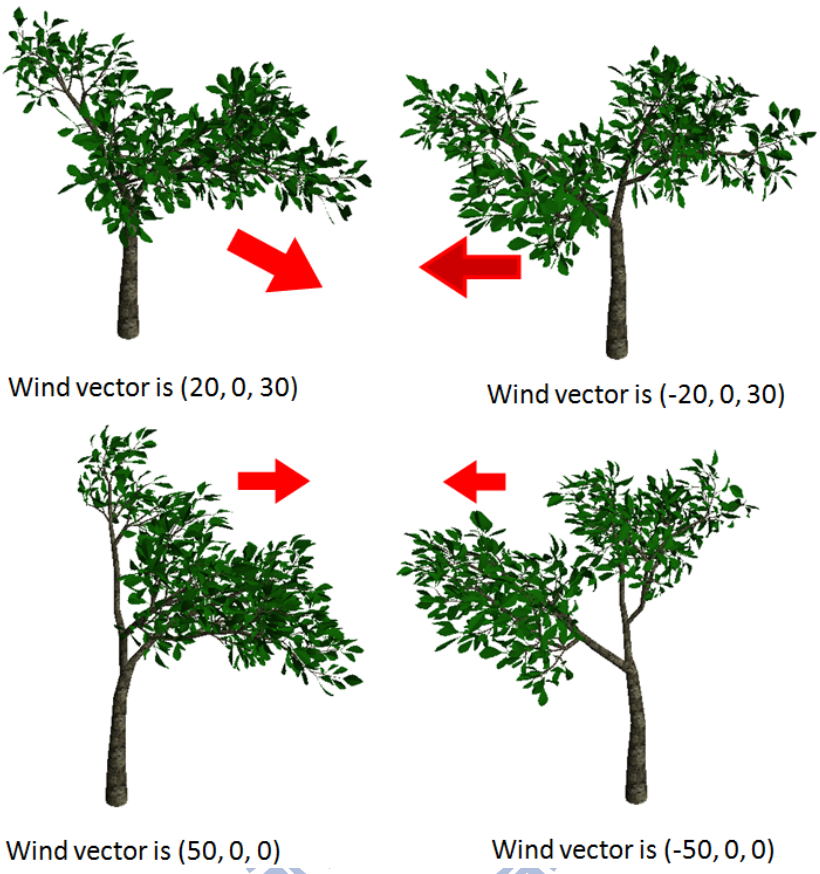


Figure 6.3: The tree models are affected by different winds.

We recorded the cost time of updating primitives' information. The costs of computing forces, updating the orientation of the primitive and updating the information of the triangles were included. The time of animating tree models was compared in Table 6.2. The result showed that the animation time was approximately proportional to the numbers of the primitives of the tree.

We also implemented the defoliation phenomena as shown in Figure 6.4. User can make interaction with the tree model by pressing a key to make some leaves fall down. When a leaf was marked as a fallen leaf, we separated it from the tree model and treated it as a deformable object that moved independently.

	treeA	treeB	treeC	treeD	treeE
The numbers of stems	597	2721	6470	11134	14615
The numbers of leaves	749	3320	7821	12850	16607
UB	0.004	0.018	0.044	0.075	0.099
UL	0.019	0.085	0.201	0.329	0.427

Table 6.2: The animation time (sec) of different tree models. UB: the average time of updating the branches, UL: the average time of updating the leaves.

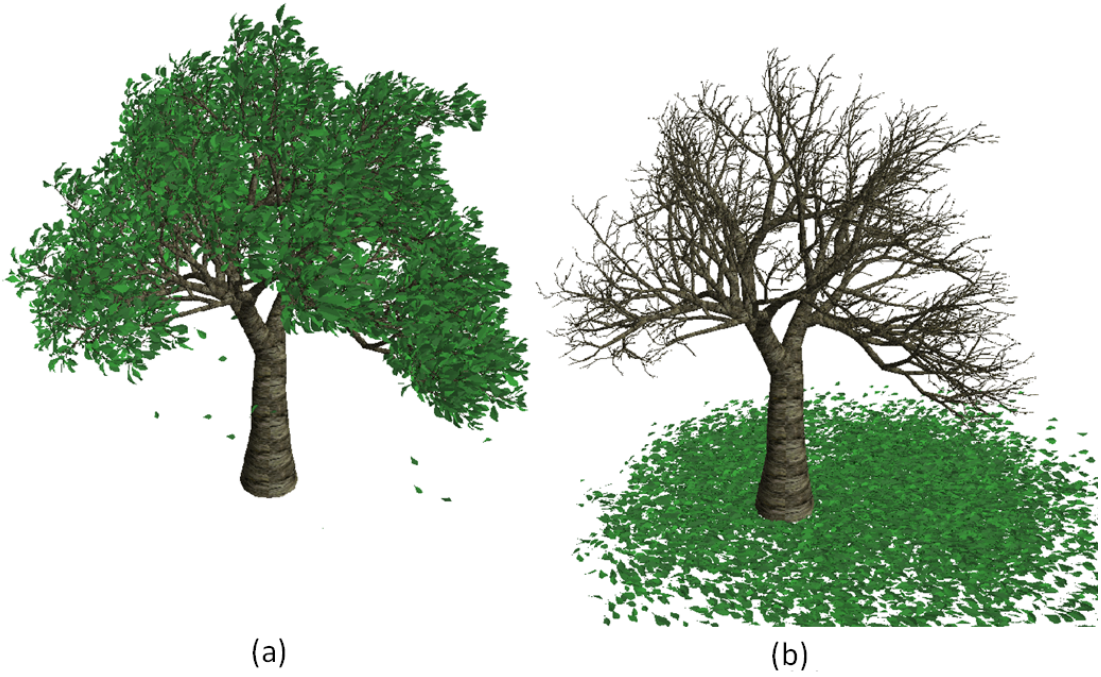


Figure 6.4: The defoliation phenomena.

6.3 Collision Handling

We showed the scenes of the animating tree models with and without collision handling in Figure 6.5. We could see that there was no intersections between the primitives of the tree using our method.

Our method for collision handling could be divided into three parts: (1)updating the k-DOPs BVH; (2)traversing BVH; (3)collision detection and giving response. We defined

Figure 6.5: The snapshots of the animating tree model with and without collisions handling.

the time spent in the first part as UK, the time spent in the second part as TV and the time spent in the third part as CD. These cost time was affected by the orientations of the k-DOP used in the simulation. The culling effect in the process of traversing BVH is better when the orientations of the k-DOP increase; However, the cost time of building the k-DOP and overlap test will increase too. We compared the time spent in these parts using k-DOPs with different k. The comparisons were shown in Table 6.3. In the case using 6-DOPs BVH to handle collisions, TV and CD were larger than the results using higher orientations k-DOPs. This was because that the culling effect of 6-DOP was smaller than higher orientations k-DOP. The number of nodes that needed to be traversed was larger than using higher orientations k-DOP. For the same reason, the number of collision detections was larger than using higher orientations k-DOP too.

We merged UK, TV and CD to compare the difference of using different k-DOPs directly. The comparison was shown in Figure 6.6. The result showed that the 6-DOPs has the best performance in our experiments. We could learn from the histogram that the cost in updating k-DOPs BVH was larger than the benefit brought by the culling effect of the higher orientations k-DOPs in our case.

	k-DOP	treeA	treeB	treeC	treeD	treeE
UK	6-DOP	0.0043	0.0192	0.0452	0.0757	0.0972
	14-DOP	0.0065	0.0292	0.0686	0.1141	0.1475
	18-DOP	0.0076	0.0335	0.0787	0.1314	0.171
	26-DOP	0.0106	0.0469	0.1103	0.1843	0.2389
TV	6-DOP	0.0025	0.013	0.032	0.064	0.085
	14-DOP	0.0019	0.01	0.025	0.048	0.064
	18-DOP	0.0018	0.01	0.024	0.046	0.059
	26-DOP	0.0019	0.01	0.024	0.046	0.059
CD	6-DOP	0.0007	0.003	0.007	0.014	0.019
	14-DOP	0.0003	0.001	0.003	0.006	0.007
	18-DOP	0.0003	0.001	0.003	0.006	0.008
	26-DOP	0.0002	0.001	0.002	0.005	0.006

Table 6.3: The average time (sec) of collision handling. UK: the time of updating BVH, TV: the time of traversing BVH, CD: the time of collision detection and giving response.

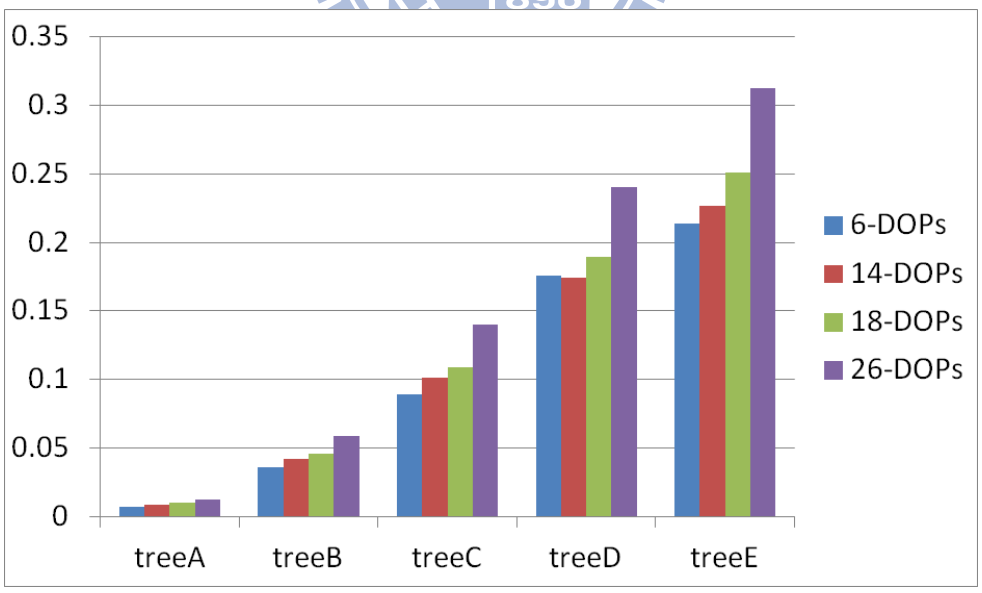


Figure 6.6: The average cost time (sec) of the collision handling using k-DOPs with different orientations.

6.4 Parallel processing

In this section, we showed the comparisons of the time spent in simulations between using single thread and multi-thread. The comparison of the animating time was shown in Table 6.4. Because our tree model was not a balance tree, the workload of each thread may be not the same. Therefore, the acceleration degree of the time of updating branches was unstable.

	tree model	single thread	multi-thread	acceleration rate
UB	treeA	0.004	0.002	2
	treeB	0.018	0.006	3
	treeC	0.044	0.016	2.75
	treeD	0.075	0.0225	3.33
	treeE	0.099	0.03	3.3
UL	treeA	0.019	0.006	3.17
	treeB	0.085	0.023	3.69
	treeC	0.201	0.054	3.72
	treeD	0.329	0.087	3.78
	treeE	0.427	0.111	3.85

Table 6.4: The comparisons of the average time (sec) of updating the information of primitives between using single thread and multi-thread. UB: the time of updating branches, UL: the time of updating leaves.

The comparisons of the time of updating BVH were shown in Table 6.5. As the workload increased, the efficiency of parallel processing increased. Because the workload of each thread that dealt with the part of updating internal nodes of BVH was not the same, the acceleration rate was affected slightly.

	tree model	single thread	multi-thread	acceleration rate
6-DOP	treeA	0.0043	0.0022	1.95
	treeB	0.0192	0.009	2.13
	treeC	0.0452	0.021	2.15
	treeD	0.0757	0.035	2.16
	treeE	0.0972	0.043	2.26
14-DOP	treeA	0.0065	0.003	2.17
	treeB	0.0292	0.012	2.43
	treeC	0.0686	0.028	2.45
	treeD	0.1141	0.043	2.65
	treeE	0.1475	0.053	2.78
18-DOP	treeA	0.0076	0.003	2.53
	treeB	0.0335	0.013	2.58
	treeC	0.0787	0.03	2.62
	treeD	0.1314	0.047	2.8
	treeE	0.171	0.058	2.95
26-DOP	treeA	0.0106	0.004	2.65
	treeB	0.0469	0.017	2.76
	treeC	0.1103	0.04	2.78
	treeD	0.1843	0.06	3.07
	treeE	0.2389	0.073	3.27

Table 6.5: The comparisons of the average time (sec) of updating BVH between using single thread and multi-thread

We further compared the time of the collision handling process after accelerating by parallel processing using different k-DOPs. The results was shown in Figure 6.7.

We learned from the experiment that the 14-DOPs had the best performance than the others. The time saved by the k-DOP BVH with larger k overtook the cost of updating itself after the acceleration of updating BVH. The superiority of the k-DOPs with higher

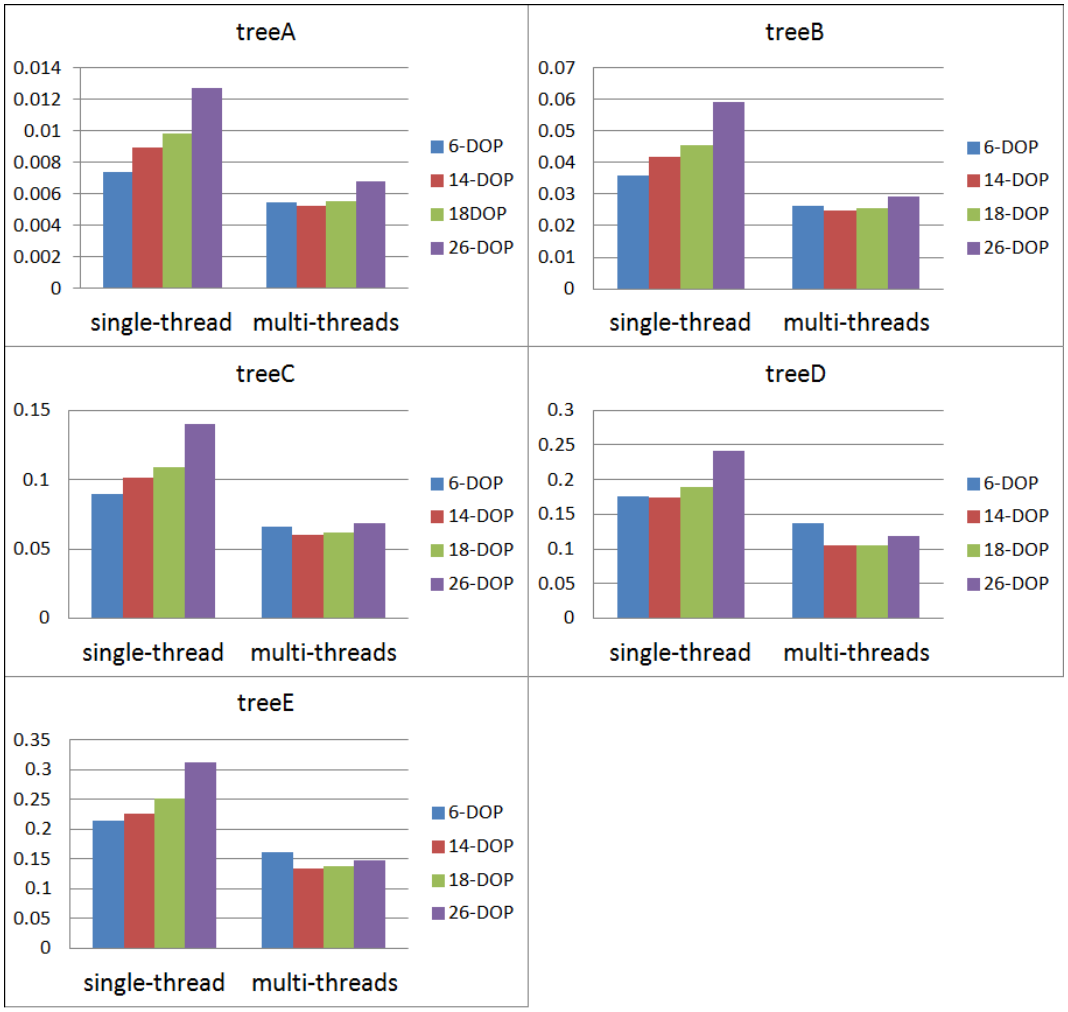
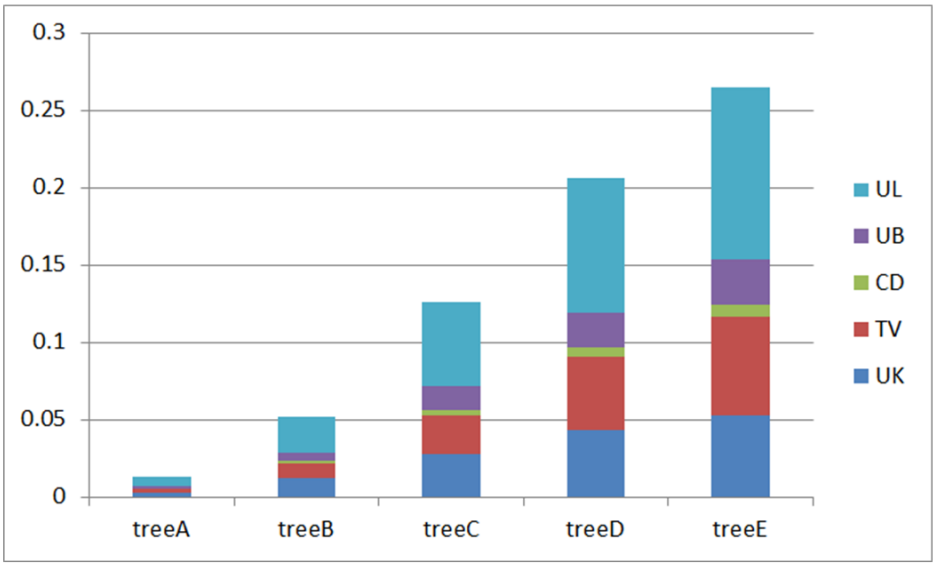


Figure 6.7: The comparison of the time (sec) of collision processing between using single-thread and multi-thread.

orientations became more obvious as the number of the k-DOPs need to be updated increased. When the numbers of primitives exceed about 15000, the performance using 6-DOPs became worst among these different kinds k-DOPs.

We put the cost of each component of the simulation together to see which component had largest cost in Figure 6.8. The detail data was shown in Table 6.6.



UL : The time (sec) of updating leaves.
UB : The time (sec) of updating branches.
CD : The time (sec) of collision detection and giving response.
TV : The time (sec) of traversing BVH.
UK : The time (sec) of updating BVH.

Figure 6.8: The cost (sec) of each component of the simulation using 14-DOP.

We observed that the time of updating the information of leaves still be the main cost of the simulation after accelerating by using multi-thread.

We added the time of runtime stage to get the frame rate of the animations. Table 6.7 shows the frame rate of tree animations.

	UL	UB	CD	TV	UK
treeA	0.006(46%)	0.002(15%)	0.0003(2%)	0.0019(14%)	0.003(23%)
treeB	0.023(44%)	0.006(11%)	0.001(2%)	0.01(20%)	0.012(23%)
treeC	0.054(43%)	0.016(13%)	0.003(2%)	0.025(20%)	0.028(22%)
treeD	0.087(42%)	0.023(11%)	0.006(3%)	0.048(23%)	0.043(21%)
treeE	0.111(42%)	0.03(11%)	0.007(3%)	0.064(24%)	0.053(20%)

Table 6.6: The detail time (sec) of each component of the runtime stage using 14-DOP. UL: the time of updating leaves, UB: the time of updating branches, CD: the time of collision detection and giving response, TV: the time of traversing BVH, UK: the time of updating BVH.

	treeA	treeB	treeC	treeD	treeE
6-DOP	70.349	18.001	7.284	4.246	3.281
14-DOP	72.892	18.971	7.831	4.748	3.709
18-DOP	72.307	18.517	7.806	4.65	3.648
26-DOP	70.043	17.367	7.323	4.463	3.524

Table 6.7: The frame rate (frames per second) of the animations

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have presented a method to handle the interactions of a tree in a windy environment. We build the tree models which are collision-free. We use the k-DOPs to cull-out the primitive pairs those need not to be further checked. We also use a voxel-based acceleration method to accelerate the process of building tree. A physically-based method is applied for animating the trees. During the simulation, we proposed a novel method to handling collisions within the tree. We maintain a k-DOP BVH during the simulation process to collect all the potentially colliding pairs and resolve collisions for the colliding pairs.

The experimental results show that our method animates the tree model naturally and successfully resolve the collision events between the primitives during the simulation process.

7.2 Future Work

We had made an attempt to employ GPU to accelerate our method but the performance was not satisfactory. In our method, the update of a primitive must wait for the results of it's parent so that we cannot update all primitives at a time. The information

used in updating a primitive cannot be reused by another one. We need to transfer all information of the primitives from CPU to GPU. The amount of data for transferring is huge. In addition, the computation of updating the information of a primitive is involved with simple computation. The cost of transferring data between CPU and GPU is much higher than the benefit brought by parallel processing in our case. In the future, we would like to investigate a different strategy to implement our method on GPU.

We also want to improve the realistic of the appearance of our tree models. Our stems and leaves are modeled by using few triangles so that their surfaces can not modeled in a natural way. This can be discovered easily when we take a close look. We want to improve it in the future.

In our collision handling, we simply push away the colliding leaves without computing their deformation. In the future, we want to simulate the detail deformation of the leaf due to collisions. Also, the accurate collision detection between the primitives is still a challenge for us.

Finally, We want to model the various wind model. We would like to simulate the stems being destroyed due to the strong wind loading and handle the collisions of broken branches and the defoliations.

Bibliography

[AK06] Y. Akagi and K. Kitajima. Computer animation of swaying trees based on physical simulation. *Computers & Graphics*, 30(4):529--539, 2006.

[CGZ+05] Y.Y. Chuang, D.B. Goldman, K.C. Zheng, B. Curless, D.H. Salesin, and R. Szeliski. Animating pictures with stochastic motion textures. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 853--860, 2005.

[DCL+09] L. Ding, C. Chongcheng, T. Liyu, W. Qinmin, and X. Wenqiang. Interactive physical based animation of tree swaying in wind. In *2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, pages 623--628, 2009.

[DRBR09] Julien Diener, Mathieu Rodriguez, Lionel Baboud, and Lionel Reveret. Wind projection basis for real-time animation of trees, mar 2009.

[GCF01] T.D. Giacomo, S. Capo, and F. Faure. An interactive forest. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, pages 65--74, 2001.

[HBM03] J.C. Hart, B. Baker, and J. Michaelraj. Structural simulation of tree growth and response. *The Visual Computer*, 19(2):151--163, 2003.

[HFC09] S. Hu, T. Fujimoto, and N. Chiba. Pseudo-dynamics model of a cantilever beam for animating flexible leaves and branches in wind field. *Computer Animation and Virtual Worlds*, 20(2-3):279--287, 2009.

[HKW09] R. Habel, A. Kusternig, and M. Wimmer. Physically guided animation of trees. In *Computer Graphics Forum*, volume 28, pages 523--532, 2009.

[KHM+98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21--36, 1998.

[LGF+06] W. Li, W. Guo, G. Feng, Y. Meng, and M. Li. Broad-leaf Virtual Plant. In *IEEE International Conference on Industrial Technology*, pages 734--738, 2006.

[OFT+03] S. Ota, T. Fujimoto, M. Tamura, K. Muraoka, K. Fujita, and N. Chiba. 1/f\ beta Noise-Based Real-Time Animation of Trees Swaying in Wind Fields. 2003.

[OK10] N.J. Oliapuram and S. Kumar. Realtime forest animation in wind. In *Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing*, pages 197--204, 2010.

[Opp86] P.E. Oppenheimer. Real time design and animation of fractal plants and trees. In *ACM SiGGRAPH Computer Graphics*, volume 20, pages 55--64, 1986.

[OTF+04] S. Ota, M. Tamura, T. Fujimoto, K. Muraoka, and N. Chiba. A hybrid method for real-time animation of trees swaying in wind fields. *The Visual Computer*, 20(10):613--623, 2004.

[PL91] P. Prusinkiewicz and A. Lindenmayer. The algorithmic beauty of plants (The Virtual Laboratory). 1991.

[SO99] T. Sakaguchi and J. Ohya. Modeling and animation of botanical trees for interactive virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 139--146, 1999.

[Sta97] J. Stam. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. In *Computer Graphics Forum*, volume 16, pages C159--C164, 1997.

[SZCZ05] P.A. Singh, N. Zhao, S.C. Chen, and K. Zhang. Tree animation for a 3d interactive visualization system for hurricane impacts. In *IEEE International Conference on Multimedia and Expo*, pages 598--601, 2005.

[VHDFVR06] W. Van Haevre, F. Di Fiore, and F. Van Reeth. Physically-based driven tree animations. In *Eurographics Workshop on Natural Phenomena*, pages 75--82, 2006.

[VHFBVR04] W. Van Haevre, F.D. Fiore, P. Bekaert, and F. Van Reeth. A ray density estimation approach to take into account environment illumination in plant growth simulation. In *Proceedings of the 20th spring conference on Computer graphics*, pages 121--131, 2004.

[WD04] J.C. Wong and A. Datta. Animating real-time realistic movements in small plants. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 182--189, 2004.

[Web08] J.P. Weber. Fast simulation of realistic trees. *Computer Graphics and Applications, IEEE*, 28(3):67--75, 2008.

[WP95] J. Weber and J. Penn. Creation and rendering of realistic trees. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119--128, 1995.

[WWG06] G. Wu, L. Wenhui, and F. Guanghui. Physically based animation of broadleaf plant. *IJCSNS*, 6(2A):198, 2006.

[YHYW10] M. Yang, M.C. Huang, G. Yang, and E.H. Wu. Physically-based animation for realistic interactions between tree branches and raindrops. In *Proceed-*

ings of the 17th ACM Symposium on Virtual Reality Software and Technology, pages 83--86, 2010.

[YSWS09] M. Yang, B. Sheng, E. Wu, and H. Sun. Multi-resolution tree motion synthesis in angular shell space. In *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 47--52, 2009.

[ZST+06] L. Zhang, C. Song, Q. Tan, W. Chen, and Q. Peng. Quasi-physical simulation of large-scale dynamic forest scenes. *Advances in Computer Graphics*, pages 735--742, 2006.

[ZZJ+07] L. Zhang, Y. Zhang, Z. Jiang, L. Li, W. Chen, and Q. Peng. Precomputing data-driven tree animation. *Computer Animation and Virtual Worlds*, 18 (4-5):371--382, 2007.