Cone Tracing of Human Hair Fibers

Jorge Filipe Branco Romana Martins
jorge.r.martins@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2016

Abstract

Rendering hair presents several challenges. Hair fibers are very thin, requiring very high super-sampling rates in order to render images without aliasing artifacts. Moreover, typical bounding volume hierarchy acceleration structures only use axis aligned bounding boxes, which do not tightly fit the hair primitives, generating a very large number of intersection tests with few actual hits. Both these facts have a negative impact on the rendering performance. In this work we aim at improving the rendering performance with two contributions: a cone based solution and a hybrid bounding volume hierarchy solution. With the cone based solution we are able to produce aliasing free images with just a super-sampling of 2x2 and produce images of comparable quality to the ones produced with a stochastic ray tracer with a 16x16 super-sampling rate while requiring a much lower rate and achieving speedups of up to 4. With the hybrid bounding volume hierarchy, which uses both axis aligned and oriented bounding boxes to bound hair primitives, we are able to achieve an average intersection test reduction of 53%, while only increasing the memory footprint by 11%.

Keywords: Hair rendering, cone tracing, hybrid bounding volume hierarchy, intersection test reduction, speedup

1. Introduction

Producing results at interactive frame rates with ray tracing techniques is still a challenge when rendering hair, since the reduced thickness of hair fibers, requires a high super-sampling rate in order to produce results free of aliasing artifacts, which has a great impact on the overall rendering performance. This is even more noticeable for dense curly hair scenes. Due to the reduced thickness of the hair fibers, a high super-sampling rate of up to 16x16 might be needed to render images without artifacts. Furthermore, assuming we are using a typical bounding volume hierarchy (BVH) where nodes are axis aligned bounding boxes (AABBs) we will have a large number of intersection tests per pixel with few geometry hits. This results from the fact that AABBs cannot tightly bound hair strands in this situation, leading to a high number of positive ray intersections with the bounding box that end up not intersecting the hair strand.

Oriented bounding boxes (OBBs) are an alternative to AABBs, since they allow a tighter bound between the bounding volume and the hair strands, drastically reducing the number of intersection tests to be performed with hair stands. However, intersection with an OBB is computationally heavier, which negates the benefits of reducing the number of intersection tests. OBBs also require more memory, increasing memory requirements.

In this paper we present two approaches for accelerating the rendering process: a cone tracing based GPU algorithm and a hybrid BVH structure that uses both AABBs and OBBs as leaf nodes. In this way, we are able to drastically reduce the super-sampling required to produce artifact free results while minimizing the number of intersection tests to be performed with primitives, achieving speedups of up to 4, depending on the scene.

1.1. Related Work

Our work is most related to techniques for accelerating the rendering process through the use of acceleration structures, techniques for handling hair transparency and simulating the hair’s properties and ray bundle techniques.

Studies for accelerating the rendering process have focused on structures such as grids [1, 2, 3], kd-trees [4] and bounding volume hierarchies [5]. More recently, we have seen an increase in techniques for fast GPU construction of BVHs [6, 7] as well as techniques for reducing the number of intersection tests through the use of a hybrid BVH of AABBs and OBBs [8].

Techniques for handling hair transparency and simulating the hair’s properties have also been developed, falling in one of two categories: on the one
hand, we have pure physical models, such as path tracing [9] and bi-directional path tracing [10] and photon mapping [11, 12, 13, 14]. Several techniques have also been proposed to simulate the hair’s properties, with scattering models for single hair fibers [15, 16, 17], multiple scattering [18] and natural illumination [19]. Recently, we have also witnessed an increase in the studies focusing on the animation of hair scenes in real time, with the AMD TressFx technique [20, 21] and techniques for the animation of hair scenes with data-driven interpolation of guide strands [22] and by bundling hair fibers into hair meshes [23].

On the other hand, we have ad hoc approaches, which do not follow a physical model, in which we can include deep shadow maps [24], opacity shadow maps [25], deep opacity maps [26], adaptive volumetric shadow maps [27] and order independent transparency techniques, such as adaptive transparency [28] and stochastic transparency [29].

Several studies have also been conducted in order to produce artifact free results through the use of super-sampling techniques [30, 31, 32, 33] and ray bundle techniques that use cones [34, 35] beams [36] and hypercubes [37].

1.2. Contributions

Our two main contributions are: an efficient cone-based algorithm for hair rendering and a hybrid BVH, which uses both AABBs and OBBs to bound the hair’s primitives in order achieve a closer fit to the primitives and reduce the number of intersection tests.

Our solution starts by building a BVH following the algorithm described in [6], which builds a fast LBVH on the GPU. We then use the algorithm presented in [7] to perform treelreconstruction on the LBVH, optimizing and transforming it into a TRBVH. Our TRBVH can either be a pure ABB BVH or a hybrid BVH that uses both AABBs and OBBs to bound the hair’s primitives. To handle the transparency of the hair strands we use the adaptive transparency technique presented in [28] to estimate the transmittance function along a primary ray. We use the adaptive transparency structure with eight nodes and during the BVH traversal we always store the eight closest intersections, which will be used during the shading process. For the shading we use a Phong model due to its simplicity as our main aim is to improve the rendering performance rather than producing images comparable to the ones produced by physical shading models.

In our work, hair is represented as a sequence of strands, where each strand has a defined number of segments. Each segment is represented by a generalized cylinder.

2. Cone Tracing Solution

We propose a new cone based solution to improve the rendering performance of hair scenes, producing results of equal quality to the ones produced with a stochastic ray tracing solution, but at faster rendering times and requiring very low super-sampling rates in order to produce aliasing free results. The general idea is that instead of taking several samples inside a pixel with rays, we use a cone. The BVH is then traversed with that cone and the eight nearest intersections with hair segments are saved. However, performing the actual intersection between a cone and a hair segment (cylinder) would generate a great amount of overhead, negating the benefit of reducing the super-sampling rate. Instead, we estimate the intersection area between a cone and a cylinder by shooting rays inside a cone, saving some of the results to be used in the shading process. This effectively reduces the super-sampling needed, as well as the number of BVH traversal steps. In addition we present a solution for reducing the number of intersection tests to be performed when traversing the BVH through the use of a hybrid approach, which uses both AABBs and OBBs.

In our solution, we start by constructing a LBVH as described in [6] for a .HAIR scene, courtesy of Cem Yuksel [38]. We then improve the BVH’s performance through the use of treelet reconstruction, as described in [7], building a TRBVH. The TRBVH can have two different configurations: a typical AABB configuration, where all nodes are axis aligned, and a hybrid configuration, which uses AABBs for the internal nodes and uses both AABBs and OBBs for the leaf nodes. The selection between the two depends on the type of BVH we want to build. In the case of our hybrid BVH, we either create an axis aligned or an object oriented bounding volume, depending on the volume difference between the two. we generate an axis aligned bounding volume if its volume is less or equal to fifteen times the volume obtained with an OBB, and generate an object oriented bounding volume otherwise. We concluded from testing that a factor of fifteen is able to perform closely to using OBBs for all leaves, regarding the number of intersection tests per pixel, while requiring less memory and having a faster data transfer time.

For the intersection with an OBB, we convert the OBB into an equivalent AABB and then perform the intersection with that OBB. To do so, we compute the matrix that allows us to transform the bounding box so that the cylinder that it bounds has the base translated and rotated to the origin and the axis is collinear to the z axis. We then mark the node as an OBB, transfer the matrix to the device and compute the box’s limits as if it were an AABB, which are required to construct the LBVH.
and the TRBVH. After having concluded the construction of the TRBVH we run a separate kernel to update the limits of OBB nodes, as if the hair segment they bound was in the origin and aligned with the z axis. During the intersection we use the precomputed matrix to transform the cone before performing an intersection with the AABB.

To render hair we use primary and shadow rays and Phong shading. We compute the transparency among hair fibers and along a ray/cone with the adaptive transparency technique presented in [28]. We store a structure with eight nodes for each pixel that is updated when we traverse the BVH with primary rays.

Since we use primary and shadow rays, the BVH is traversed in two occasions: tracing and shading. For our traversal algorithm we based the generalized traversal algorithm proposed by Tero Karras [6]. However, this solution was meant for finding the nearest intersection; for that reason we adapted it to our needs, which resulted in three major differences. First, our solution can use two types of BVH: a pure axis aligned approach and a hybrid approach. Secondly, we are interested in finding a set of intersections, rather than only the nearest one. When tracing, we save the eight closest intersections and during shading we traverse the BVH to estimate the amount of light that reaches a point based on the amount of primitives intersected and their transparency. Thirdly, we improved performance by avoiding traversing unnecessary nodes.

In a typical traversal algorithm, even after an intersection with a leaf has been found, no consideration is taken regarding the distance at which the intersection occurred. This leads to unnecessary work, since the first intersected leaf could be the nearest one. Moreover, in hair scenes we can have a very large number of hair strands intersected by a ray, which has a great impact on performance. In our solution we avoid this unnecessary work with two approaches.

For the tracing traversal algorithm we keep a variable with the minimum intersection distance found so far following the reasoning that the intersection distance with a bounding box will always be lower or equal to the intersection distance with a leaf. Therefore, if we test the intersection with a bounding box and the intersection distance is greater than the minimum distance we have already found, we can ignore the node and all its children. However, since we are interested in finding a set of intersections, rather than only the nearest one, we apply a factor when updating the minimum distance but ensure that its value can only decrease or remain the same when updated. During the traversal all the primitive intersections found are added to the adaptive transparency structure, so that the transmittance function can be computed. We also keep a list with the eight closest intersections. Thought the use of this method we were able to achieve an average performance increase of 5%.

For the shading traversal algorithm, the solution is simpler. We are only interested in finding out the amount of light that reaches a point. Therefore, we keep the value of the estimated transmittance, which is updated whenever an intersection with a primitive is found, and stops when there are no more objects to intersect or when the transmittance value reaches a predefined threshold.

Algorithm 1 Tracing algorithm for our cone solution

```plaintext
function TRACE(origin, direction, spread, background) 
    intersectionList[LST_SIZE] = 0 
    finalColor = 0 
    at.initialize(background) 
    results[LST_SIZE][N_SHADOW_POINTS] = 0 
    cone = Cone(origin, direction, spread) 
    traversalBVH(cone, intersectionList, at, results) 

    if intersectionFound then 
        for each fragment in intersectionList do 
            index = fragment.index 
            fragment.color = shade(results[index]) 
        end for 
    end if 

    for each fragment in intersectionList do 
        vis = at.getTransmittance(fragment.distance) 
        finalColor = finalColor + fragment.color * fragment.area * (1 - fragment.transparency) * vis 
    end for 

    return finalColor 
end function
```

Algorithm 1 illustrates how we implemented our cone-based solution. Each pixel or sub-pixel, if we are using super-sampling, executes the `trace` function, which receives the cone data (origin, direction and spread) and then traverses the BVH for the first time, finding the eight closest intersections, which are kept in the list `intersectionList`. Each intersection found is also added to the adaptive transparency structure `at` so that the transmittance function can be estimated and used during shading. We also store some of the results of the intersections (intersection point, surface normal, etc...) that are used during the shading process.

To compute the intersection area between a cone and a cylinder we developed an algorithm that, instead of computing the real intersection, uniformly "shoots" rays inside a cone and estimates the area occupied by the cylinder. To perform these intersections we used a simple ray-cylinder intersection algorithm and shoot five rays inside the cone, since testing revealed that a good approximation for the
intersection area could be achieved with that number of rays. We also store some of the intersection results to be used in the shading process, since unlike in the algorithm proposed by Hao Qin [35] our scenes can have very long hair fibers, which make applying the shading only to the base of the cylinder impossible. Moreover, we tested preforming shading with different amounts of stored points, having concluded that storing three or four points of intersection per cone provides results free of artifacts and without greatly increasing the computational cost. The number of points stored depends on the scene and super-sampling rate. For super-sampling rates lower than 4x4 we store up to four intersection points. For higher super-sampling rates we either store three or four points, choosing the approach that provides the best performance and lowest RMS error.

After having found the set of intersections we need to compute the color at each of them. To do so, we go through each intersection found and we shoot a shadow ray from the intersection point to each of the light sources. In our solution we always have two light sources per scene. Then, we estimate the amount of light that reaches that point and use the Phong model to compute the color, forming "fragments". Finally, we compute the pixel’s color by composing the color at each intersection (fragment) with the transmittance function estimated by the adaptive transparency structure.

3. Results

We evaluated our two contributions by comparing our cone based solution against a stochastic ray tracing solution with a super-sampling of 16x16, and by comparing our hybrid BVH against a typical AABB BVH.

All tests were conducted in a computer with a NVIDIA GeForce GTX 780 Ti GPU, an Intel core i7-4770K CPU and 8GB of RAM, and rendered in a 1024x1024 resolution. We used six test scenes. Table 1 presents the number of hair strands and hair segments for each of the scenes.

Table 1: Number of hair strands and segments for each of the scenes used in the evaluation of our work.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Number of Hair Strands</th>
<th>Number of Hair Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curly</td>
<td>50K</td>
<td>3.39M</td>
</tr>
<tr>
<td>Dark Straight</td>
<td>50K</td>
<td>1.2M</td>
</tr>
<tr>
<td>Natural</td>
<td>10K</td>
<td>1.5M</td>
</tr>
<tr>
<td>Straight</td>
<td>10K</td>
<td>150K</td>
</tr>
<tr>
<td>Wavy</td>
<td>50K</td>
<td>2.4M</td>
</tr>
<tr>
<td>Wavy Thin</td>
<td>10K</td>
<td>863K</td>
</tr>
</tbody>
</table>

Figure 1: Representation of each of the hair test scenes used in the evaluation of our work rendered with the stochastic ray tracing solution at a super-sampling of 16x16.

3.1. Cone Tracing Solution Evaluation

To validate our cone-based solution we rendered all rendering scenes with it at super-sampling rates of 1x1 up to 6x6. We then compared each produced image with the reference, focusing on the rendering times, RMS error and overall image difference. Tables 2 and 3 show the rendering times and RMS error of the cone solution for each scene at different super-sampling rates and compares the rendering times to the stochastic reference. To compute the stochastic times we rendered ten frames and then computed the average rendering time. The lowest super-sampling rate required to produce an image equivalent to the reference is signaled with an asterisk. Figures 2 to 5 present in detail the image results obtained from rendering each of the hair scenes with our cone solution at different super-sampling rates. We also present the absolute image difference to better understand where the differences lay.

Finally we present the lowest super-sampling rate cone tracing result that has equivalent quality to the reference and compare it with the reference.

To better understand the distribution of the rendering times among the different phases of the rendering process we measured the time spent at each
Figure 2: Comparison of the results and differences obtained with our cone solution in relation to the reference for the curly hair scene. Figures (a) to (f) present in detail the results obtained when rendering the scene with a super-sampling from 1x1 to 6x6, comparing them to the reference (g). Figures (h) to (m) present the absolute value difference (multiplied by five) between each of cone results and the reference.

Figure 3: Comparison of the results and differences obtained with our cone solution in relation to the reference for the natural hair scene. Figures (a) to (f) present in detail the results obtained when rendering the scene with a super-sampling from 1x1 to 6x6, comparing them to the reference (g). Figures (h) to (m) present the absolute value difference (multiplied by five) between each of cone results and the reference.

Table 3: Comparison of the rendering times and RMS errors produced by our cone tracing solution in relation to the stochastic 16x16 super-sampling reference for super-sampling rates 5x5 and 6x6 for all the test scenes.

Table 4: Render time distribution in each of the rendering phases (shading, light transmittance estimate and tracing) for both the ray tracing and cone tracing solutions in each of the hair test scenes.
Figure 4: Comparison of the results and differences obtained with our cone solution in relation to the reference for the wavy hair scene. Figures (a) to (f) present in detail the results obtained when rendering the scene with a super-sampling from 1x1 to 6x6, comparing them to the reference (g). Figures (h) to (m) present the absolute value difference (multiplied by five) between each of cone results and the reference.

From the results presented in Tables 2 and 3 we produced three Figures: Figure 6, which presents the variation of the RMS error with different super-sampling rates, Figure 7, which presents the speedup achieved regarding the reference for each super-sampling rate and Figure 8, which presents the speedup achieved by our cone tracing solution for the production of results comparable to the reference.

From the analysis of the presented results we can conclude that without super-sampling our cone solution still has a few aliasing artifacts and is susceptible to noise in bright highlighted areas. With a super-sampling of 2x2 we are able to produce images without artifacts and the noise is reduced but still present in the highlighted areas. With a super-sampling of 3x3 we achieve very pleasing results, without artifacts and almost imperceptible noise in highlighted areas, with an average RMS error of 2.06%. Moreover, the errors result mainly from the errors generated by the area estimate, which produces images that are either darker or brighter than they should be. These errors can be effectively reduced and made unnoticeable by raising the super-sampling to 4x4, 5x5 or 6x6 (depending on the scene). We can also conclude that it is possible to produce images identical to the reference with a super-sampling of 4x4 for the curly and natural scenes, 5x5 for the straight, wavy and wavy thin scenes and 6x6 for the dark straight scene, while achieving speedups ranging from 1.68 up to 4.01. These speedup values are even higher if we con-


Figure 6: RMS errors produced by the cone tracing solution with different rates of super-sampling in relation to the reference for each of the hair test scenes.

Figure 7: Speedup achieved with the cone tracing solution with different rates of super-sampling in relation to the reference for each of the hair test scenes.

Figure 8: Speedup achieved with the cone tracing solution for the production of results comparable to the reference for each of the hair test scenes.

3.2. Hybrid BVH Evaluation

To validate our hybrid BVH we compared the building times, rendering performance, memory consumption and number of intersection tests per pixel to the performance obtained with a typical AABB BVH with all of our test rendering scenes. We tested the performance for both the ray tracing and the cone tracing solutions, having obtained the results presented in Tables 6 and 5. Figures 9 and 10 illustrate the number of intersection tests per pixel for both the AABB BVH and the hybrid BVH for each of our test rendering scenes.

Table 5: Comparison of the memory consumption, building times and rendering performance between a typical AABB BVH and our hybrid BVH for both the ray tracing and cone tracing solutions. The rendering performance was measured as a speedup of the performance obtained with the hybrid BVH in relation to the performance obtained with a typical AABB BVH.

<table>
<thead>
<tr>
<th>Scene</th>
<th>AABB BVH</th>
<th>Hybrid BVH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BVH Size</td>
<td>BVH Data Transfer Time</td>
</tr>
<tr>
<td>Curly</td>
<td>746MB</td>
<td>12.40s</td>
</tr>
<tr>
<td>Dark Straight</td>
<td>264MB</td>
<td>4.37s</td>
</tr>
<tr>
<td>Natural</td>
<td>332MB</td>
<td>5.15s</td>
</tr>
<tr>
<td>Straight</td>
<td>33MB</td>
<td>0.57s</td>
</tr>
<tr>
<td>Wavy</td>
<td>528MB</td>
<td>8.74s</td>
</tr>
<tr>
<td>Wavy Thin</td>
<td>190MB</td>
<td>3.17s</td>
</tr>
</tbody>
</table>

Table 6: Intersection test reduction of the hybrid BVH in relation to a typical AABB BVH for both the ray tracing and cone tracing solutions.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Intersection Test Reduction (Ray Tracing)</th>
<th>Intersection Test Reduction (Cone Tracing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curly</td>
<td>78%</td>
<td>69%</td>
</tr>
<tr>
<td>Dark Straight</td>
<td>71%</td>
<td>59%</td>
</tr>
<tr>
<td>Natural</td>
<td>18%</td>
<td>13%</td>
</tr>
<tr>
<td>Straight</td>
<td>84%</td>
<td>71%</td>
</tr>
<tr>
<td>Wavy</td>
<td>72%</td>
<td>60%</td>
</tr>
<tr>
<td>Wavy Thin</td>
<td>57%</td>
<td>46%</td>
</tr>
</tbody>
</table>

From the values presented in Tables 6 and 5 we can conclude that the hybrid BVH only implies an average memory increase around 11%, an increase in the data transfer time around 45% and achieves an average intersection test reduction of 63% for the ray tracing solution and 53% for the cone tracing solution. It is also possible to conclude that the building times are low and the same for both approaches, since construction algorithm is the same. However, we never achieve speedup for the hybrid
Figure 9: Comparison of the number of intersection tests per pixel achieved with a typical AABB BVH and our hybrid BVH for the curly, dark straight and natural hair test scenes. The intersection tests intervals per color are the following: gray - 0, blue - 1 to 19, green 20 to 39, yellow - 40 to 79, red - 80 to 159 and dark red - higher than 159 intersection tests.

Figure 10: Comparison of the number of intersection tests per pixel achieved with a typical AABB BVH and our hybrid BVH for the straight, wavy and wavy thin hair test scenes. The intersection tests intervals per color are the following: gray - 0, blue - 1 to 19, green 20 to 39, yellow - 40 to 79, red - 80 to 159 and dark red - higher than 159 intersection tests.

approach although, as Figures 9 and 10 illustrate, the hybrid approach largely decreases the number of intersection tests performed per pixel for both the ray tracing and cone tracing solutions. We can also notice that the cone tracing solution generates a few more intersection tests than the ray tracing solution and achieves 10% less intersection reduction. This is expected since a cone can traverse more objects than a single ray, therefore we will experience a few more intersections. Regarding the lack of speedup we remind that the hybrid BVH was first proposed as a CPU acceleration structure. As it is known, CPUs have a much different architecture from GPUs, not being affected by data and code divergence in the same way as GPUs are. We attribute the lack of speedup to the overhead generated by both code and data divergence, which are unavoidable in the implementation of a hybrid BVH. In the hybrid BVH we can have two types of bounding volumes bounding a leaf. Therefore, we always need to test the bounding box’s type, which generates code divergence when we have both AABBs and OBBs. Moreover, to compute the intersection with the OBB we need to access a matrix, which is different for each OBB and does not exist in the case of an AABB, which generates data divergence.

4. Conclusions
We have proposed a fast cone tracing solution, which is capable of producing images of comparable quality to the ones produced with a stochastic algorithm at a 16x16 super-sampling rate, while requiring much lower super-sampling rates. In fact our algorithm never required more than a 6x6 super-sampling rate to produce results identical to the reference and for most scenes using a 4x4 or 5x5 was enough. Besides requiring less super-sampling, our cone algorithm is also capable of achieving speedups going from 1.68 to 4.01 depending on the scene.

Regarding the hybrid BVH, we were able to achieve an average intersection test reduction of 53%, while only increasing the memory footprint by 11%. However, from the conducted tests we never achieved speedup in relation to the AABB BVH. We believe the data and code divergence generated by the hybrid BVH to be the reason behind the absence of speedup. The data divergence is generated
by the OBBs, since each OBB has its own matrix, which has to be accessed during the intersection. The code divergence is generated when we test the type of the bounding box, which for the leaves can either be an AABB or an OBB.

In our work we focused only in the rendering of static hair scenes. However, we also believe that our cone area intersection estimate algorithm can be easily extended to support other primitives. We also believe a shadow map technique, such as adaptive volumetric shadow maps [27], could improve performance, speeding the shading process.

References


