Image Synthesis (INF 584). Final Project

Efficient Divide-And-Conquer Ray Tracing using Ray Sampling

École Polytechnique

Irada Bunyatova

I. Summary of the paper - Efficient Divide-And-Conquer Ray Tracing using Ray Sampling and of the related papers.

Efficient Divide and Conquer ray tracing is an algorithm for significantly reducing the ray tracing time by reducing the number of rays and primitives' intersection tests. According to the authors of the paper the method can accelerate the ray tracing by up to 2 times.

The naïve approach for doing rays (of size r) and primitives (of size p) intersection tests would be to check one by one if the given ray intersects with a given primitive – which would make an overall complexity of O(n * m) tests! This would be computationally very slow, especially for the cases, where there are thousands of primitives and rays. Thus it has been a goal for the researchers to create an algorithm which would reduce significantly this complexity and time.

Before the DACRT many other approaches have been taken for reducing the ray tracing time. The first approaches that were taken were using acceleration data structures – either space subdivision structures (such as grids, various types of trees), or primitive partitioning structures (bounding volume hierarchy). Those methods provide a way to reduce the complexity to O (rays * log(primitives)). The BVH algorithm was implemented as part of the course and I have observed some time reducing for ray tracing compared to the naïve approach. However, the drawback of these methods is that they would require to store those data structures – which is memory costly and this should be done before ray tracing. Another disadvantage is that the memory cost required can't be estimated beforehand.

However, some of the previous papers have introduced the methods that were used in later DACRT methods. As such, in the paper by **Wald, 2007[1]** the binning method of the bounding volume have been introduced for partitioning the bounding box. Along with the binning method, a cost function for selecting one of the partitioning was introduced in the paper:

$$Cost(V \rightarrow \{L,R\}) = K_T + K_I(\frac{SA(V_L)}{SA(V)}N_L + \frac{SA(V_R)}{SA(V)}N_R)$$

where SA indicates the surface area of bounding volumes, Nl and Nr are number of triangles respectively in Vl and Vr. Kt and Ki are constants of cost of traversal step and ray triangle intersection test.

This cost function in its improved version have been used in the EDACRT paper, where instead of the ratio of surface areas, the ratio of rays intersecting with bounding volumes over the number of sampled rays was used. This is a significant improvement that was made in EDACRT, since with the new cost function the distribution of rays has been taken into account.

Later, the methods that don't require storing those data structures and could be done on the fly were introduced. Before the EDACRT some other DACRT approaches have been taken for reducing the ray tracing time.

DACRT – Divide and conquer ray tracing as the name suggests, divides the problem into easier to solve smaller problems. For the case of ray tracing the goal is to reduce the big problem – doing all rays and primitives' test to tests only between a small subset of primitives and subset of rays. The problem is considered to be small when the number of rays or primitives is reduced to a certain number. When the problem is small enough, the naïve ray tracing is performed. I would like to summarize the two papers, which have introduced different approaches of divide and conquer ray tracing.

The main contribution according to the authors of the paper **Mora, 2011 [2]** was introducing a method which would perform ray tracing without the need for storing data structures – which would perform faster for dynamic cases and would perform with same speed for static cases. Since they weren't using a stored data structure, they were using throughout the paper the term Axis Aligned Spatial Subdivision instead of kd-tree.

As their algorithm suggests, for the small number of rays or primitives the brute force approach will be used. Otherwise the recursive approach would be used. The current space in DACRT function is subdivided into K subsets. Then for each of the subsets the rays intersecting the subdivision are stored, as well as the primitives. After this step the DACRT method is called recursively with a new smaller subset of rays and primitives. They also suggest to remove the rays if the space is traversed in visibility order, which however might be complicated for random rays. For primary rays the conic packets of rays are used for optimization reasons (instead of considering a problem with rays and triangles a problem of cones and triangles is solved) according to the lecture introducing his method, given by **Mora [3].**

There is another paper by **Afra, 2012[4]** which have improved the Mora's method that contributes by handling the incoherent rays using SSE and Advanced vector extensions instruction sets, as it is stated in the paper. They have used object partitioning for dividing the list of primitives into disjoint sets. Then the ray filtering is performed for removing the rays that don't intersect the bounding box. After filtering of rays, the triangles are divided into two subsets by two methods- middle partitioning and SAH partitioning (described by **Wald, 2007 [1]**). As in the previous method described by Mora, in paper by Afra when reaching certain number of rays or triangles (8 as suggested in paper), they performed naïve ray intersection.

As we can see, in those papers the subdivision was done only by taking into account the distribution of triangles, the distribution of rays was not taken into account by supposing that rays are distributed uniformly (which is not always the case). In many cases the number of rays is significantly bigger than the number of primitives, because one would trace many rays to remove the aliasing in the image. Thus it is crucial to be able to reduce the rays based on their distribution, which would imply the reduce of all ray triangle intersection tests.

The paper Efficient Divide and Conquer ray tracing by **Nabata, 2013 [5]** improves the previous DACRT methods by considering the ray distribution (which is done by ray sampling). This method provides an acceleration by up to 2 for primary and secondary rays. The main contributions of the paper according to authors are:

- Ray sampling method that reduces the time spent on ray tracing both for coherent and incoherent rays. By using ray sampling information about which partitioning and which order of traversal to choose is obtained.

- A new cost metric was introduced, that allows to decide whether the rays should be filtered or this step can be skipped. This allows to reduce time spent on ray traversal, since there are cases when ray filtering doesn't improve the speed.

The general structure of previously introduced DACRT method is used in this paper by recursive calls and with subsets of rays and primitives. In EDACRT some methods described in previous papers were used as well. Firstly, the binning subdivision method described by **Wald, 2007** [1] was used for dividing the bounding volume. Then the same principle of using the cost function for choosing one of the partitioning was used. However, since now the distribution of rays is taken into account, the formula reflects this difference as well: $C(V \to \{V_{L,j}, V_{R,j}\}) = C_T + C_I(\alpha_{L,j} N_{L,j} + \alpha_{R,j} N_{R,j})$.

After determining the traversal order, according to the value of the new cost metric it is decided on whether the ray filtering performed or not. The details of the algorithm along with implementation of the paper are described in the next section.

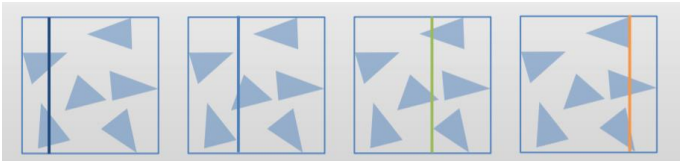
II. Implementation of the paper.

The paper provides a pseudocode of the general algorithm to be implemented, which was followed in my implementation. All the solution was developed using C++ and the code base provided in the course of Image synthesis for basic ray tracing was used.

The EDACRT function in my implementation takes all the rays with the origin at camera, their indices, axis aligned bounding box of all the triangles and several other vectors for storing the values obtained by ray triangle intersection test (in order to do the shading later using these values). The size of all those vectors is equal to the number of rays that are sent initially to the EDACRT function, and in case of the intersection the value at the corresponding index of the vector is filled.

The method itself consists of the 5 main steps:

1. **Subdivision of bounding volume**
   Firstly, there is a check on whether the number of rays or primitives is small enough for performing naïve ray tracing. For this the values of – 50 both for number of primitives and rays were chosen.
   In naïve ray tracing the vectors for storing the intersection information of size equal to the size of the rays are updates. In case if the intersection for a ray is found the vectors storing u,v,d, as well as the vector storing information about whether the intersection was found and index of triangle intersected for that ray index are updated.
   The axis aligned bounding box V is represented by two points – start and end, with lowest and highest coordinates respectively. Then in the box V the axis with the longest distance between end and start points is chosen. After this step, the division of V into K subsets is performed:



- from presentation by **Kosuke Nabata[6]**

   Then according to the calculated bins, the left and right bounding boxes are determined for each subdivision.

For each of the triangles passed to the EDACRT the center is calculated and according to the center point coordinates the triangle is pushed to either left or right vector of triangles. After the left and right vectors of triangles are computed, the bounding volume for those vectors of triangles is calculated for each partition.
After this step we have K (number of possible partitioning) left and right triangles vectors and their left and right bounding volumes. Out of these possible partitioning the partitioning with the least cost will be selected based on results obtained with ray sampling.

2. **Ray sampling**
   Ray sampling is one of the main contributions of the method described in the paper according to the authors. Following the recommendations in the paper, if the number of rays is more than 1000, then the ray sampling is performed and 100 rays are selected out of those randomly. If the number of rays is less than 1000, then ray sampling may lead to misleading information because of random selection among those rays.
   Then these sampled rays are used to get necessary information to later determine the partitioning order and determine the traversal order. For each K possible partitioning the number of rays intersecting with left and right bounding volumes of that partition are calculated. These numbers will be then used to calculate the cost function and determine the partitioning with minimum cost. The check of intersection of a ray with box is done by performing a common ray-box intersection test, which returns as well the distance to the entry point of bounding box.

   Then the distances to the left and right bounding boxes are compared for each ray. We end up having two values (for each partition) - the number of rays that are closer to either left or right bounding boxes, which will be then used for determining traversal order.

3. **Partitioning using cost function**
   Cost function for each partitioning is calculated by:

$$C(V \rightarrow \{V_{L,j}, V_{R,j}\}) = C_T + C_I(\alpha_{L,j} N_{L,j} + \alpha_{R,j} N_{R,j})$$

   Where Nl, Nr – are number of triangles in left and right bounding volumes. Alpha values represent the ratio of number of rays intersected with left and right bounding volume by the number of sampled rays. Ct and Ci – are costs of calculating the ray bounding volume and ray triangle intersection tests, respectively.
   By using this cost function, the distribution of sampled rays is taken into account, which allows to reduce the time spent on ray tracing. After all costs are calculated, the partitioning with the least cost will be selected out of K possible partitions.

4. **Determining traversal order**
   For the selected partitioning the number of sampled rays that are closer to left and right bounding volumes of that partitioning are compared. These numbers of rays were calculated previously with the sample rays – and here we can see the importance of not sampling the rays if their number is not large enough. The bounding volume which locates closer to the rays according to the sampled rays is selected first – and will be traversed first. In order to avoid inefficient tests, it is important to determine which bounding volume is traversed first.

5. **Ray filtering/ skipping ray filtering**
   Another main contribution of the paper is the new cost metric that allows to decide on whether or not the rays need to be filtered. In ray filtering for all the rays the ray and bounding box intersection test is done. Then DACRT method is called only with the rays that intersect the given bounding box and when the distance to the bounding box is less than the distance to a primitive (if there was already an intersection for that ray). If the distance to the bounding box is more than a distance to the primitive that was already found, it means that the bounding box is located beyond the primitive – and it is culled. Those rays are not pushed to the vector of rays over which the DACRT will be performed.
   In case if the majority of the rays intersect the child bounding box, the ray filtering will not be performed, since it is computationally more expensive than just doing DACRT over all the rays. In order to determine if the filtering should be skipped the alpha values (number of intersections with child bounding box / number of sampled rays) are used in the following equation:

$\alpha > 1 - \frac{C_{bv}}{n_{child} C_{child}}$. . Otherwise the ray filtering is performed.

Then after performing EDACRT the color response for the rays for which an intersection was found is calculated (this is done in parallel). The image pixels are then filled with those values.

III. Results

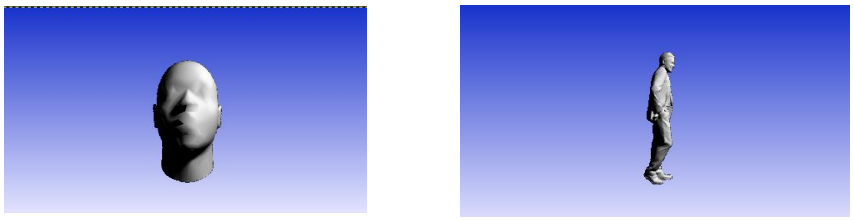


Figure 1 Images ray traced using EDACRT algorithm

In order to evaluate the speed up obtained by the EDACRT, at first the ray tracing using naïve approach was used. Both of the images are of size 480 * 270. The first image with 518.400 rays and 1200 triangles was ray traced in 33 s with naïve ray tracing, whereas with EDACRT it took only 17 s. By increasing the number of rays per pixel from 4 to 8, one would obtain more than a million rays – which would take 57 s with naïve approach and 26 seconds with EDACRT for the same number of primitives. For the second image(with 518400 rays and 29983 primitives) the running time was 172s and 526 s, respectively.

By results obtained with several tests, it is possible to observe that by increasing the number of rays and primitives, the speedup from naïve ray tracing to EDACRT increases. Since in EDACRT the data structure is not stored, the memory usage of the algorithm is not of main interest to make a comparison.

IV. References.

[1] - Wald, 2007. On fast Construction of SAH-based Bounding Volume Hierarchies

[2] - Benjamin Mora, 2011. Naive Ray-Tracing: A Divide-And-Conquer Approach

[3] - Lecture given by Benjamin Mora - https://dl.acm.org/doi/10.1145/2019627.2019636

[4] - Afra, 2012. Incoherent Ray Tracing without Acceleration Structures

[5] - Kosuke Nabata, 2013. Efficient Divide-And-Conquer Ray Tracing using Ray Sampling

[6] – Presentation by Kosuke Nabata on Efficient Divide-And-Conquer Ray Tracing using Ray Sampling
https://www.highperformancegraphics.org/wp-content/uploads/2013/Iwasaki-DaCRT.pdf