Optimising the Straylight Raytracer

Daniel Smith

March 20, 2014

1 Solution

The solution used in this optimisation is the utilisation of an R*-Tree. Generally R-Trees organize the objects within a euclidian space into a tree of axis-aligned bounding-boxes (AABB), where the children of each AABB are situated completely within the bounds of their parent's AABB. The leaf nodes then represent the bounds of the actual objects in the space. The R*-Tree, a specific type of R-Tree, tries to make these AABBs overlap each other as little as possible, thus minimising the amount of redundancy when performing spatial queries.

The aim in this case is to use the R-Tree to reduce the number of intersection-tests done when casting rays. R-Trees allow this to be done efficiently by pruning whole branches of the tree that do not match a given query. Each ray cast during the rendering process queries the R-Tree using a Ray-AABB intersection query, instead of performing an intersection test on every object in the scene. The result should be a decrease in the number of intersection tests being performed, and thus a decrease in overall render time.

The implementation used for the R*-Tree can be found at `http://www.virtualroadside.com/blog/index.php/2008/10/04/r-tree-implementation-for-cpp/`

2 Experimental Setup

Experimentation was done by rendering a variety of different scenes and measuring the time taken to complete the rendering of each scene. For the experiment, each scene was rendered twice: once using the original, unoptimised version of the program, and again using the new, optimised version. The time elapsed during each render was measured using the Unix `time` command. The scenes are divided up into "simple" scenes having less than five primitive objects, and "complex" scenes having more than five primitive objects. In order to generate a large enough sample size, the experiment had to be repeated several times.

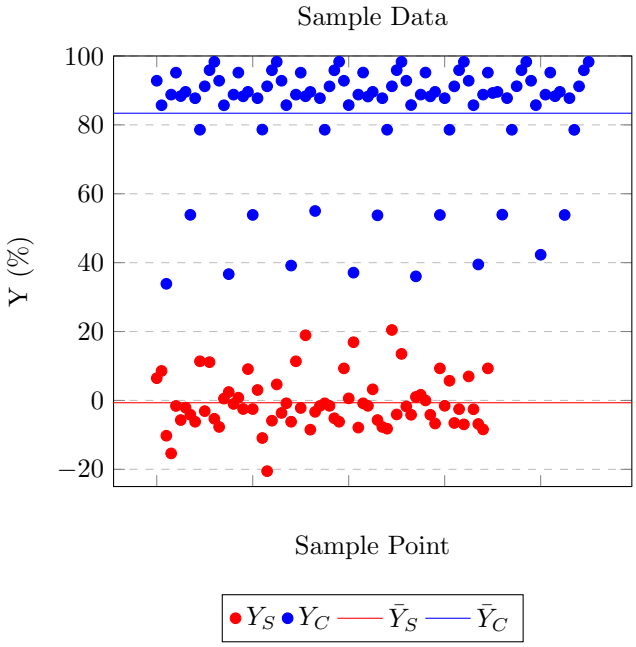
The experiment was carried out on a custom-built desktop with a 3.4GHz Core i5-3570, 8GB RAM and a 256GB Samsung 840 EVO SSD, running Ubuntu 13.10.

3 Analysis

The data was analysed to find out if the optimisation did actually bring about a significant increase in performance. In this case, a 10% increase in speed (i.e. a render time taking at least 10% less time) was considered a significant performance improvement.

The experiment was executed seven times with ten simple scenes and thirteen complex scenes, yielding a total sample size of 161. The simple-scene sample is denoted S, and the complex-scene sample is denoted C. Each sample point contains two random variables, Y_1 and Y_2, denoting the results gathered from the unoptimised and optimised versions of the program respectively. The random variable Y is defined as the percentage increase in speed from Y_1 to Y_2, and is calculated as

$$Y = \frac{Y_1 - Y_2}{Y_1}.$$



The hypothesis being tested is whether or not the mean speed increase due to optimisation is in fact greater than 10%. Thus the null and alternative hypotheses are defined as

$$H_0 : \mu \leq 10\%$$

and

$$H_a : \mu > 10\%.$$

A test level of $\alpha = .01$ was chosen for the rejection region.

4 Results

The results of the hypothesis testing are presented in the following table.

Sample	S	C	$S \cup C$
Z	-11.624	40.960	10.622
H_0	Do not reject	Reject	Reject

5 Conclusion

The results show with almost no doubt that the optimisation resulted in a significant performance increase when rendering complex scenes, while also showing no performance increase when rendering simpler scenes. However, the test results do indicate that overall render performance was somewhat improved. Thus the conclusion is that the utilisation of an R*-Tree resulted in decidedly better performance than the original implementation.