Eurographics Italian Chapter Conference (2008)
V. Scarano, R. De Chiara, and U. Erra (Editors)

On Estimating the Effectiveness of Temporal and Spatial Coherence in Parallel Ray Tracing

Biagio Cosenza[1] and Gennaro Cordasco[1] and Rosario De Chiara[1] and Ugo Erra[2] and Vittorio Scarano[1]

[1]ISISLab, Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno, Salerno, Italy
{cosenza, cordasco, dechiara, vitsca}@dia.unisa.it

[2]Dipartimento di Matematica e Informatica, Università della Basilicata, Potenza, Italy
ugo.erra@unibas.it

Abstract

In this paper we estimate the effectiveness of exploiting coherence in Parallel Ray Tracing. We present a load-balancing technique which divides the original rendering problem in balanced subtasks and distribute them to independent processors through a Prediction Binary Tree (PBT). Furthermore the PBT allows to exploit temporal coherence among successive image frames. At each new frame, it updates the current PBT using a cost function which uses the previous rendering time as cost estimate. We also provide two heuristics which take advantage of data-locality.

We assess the effectiveness of the proposed solution by running two experiments. The £rst one aims to investigate the accurancy of predictions made using the PBT. Results show that such predictions are quite accurate even considering a heavily unbalanced scene and a fast moving camera. The second experiment evaluates the two locality-aware heuristics showing a modest improvement.

Categories and Subject Descriptors (according to ACM CCS): C.1.4 [Processor Architectures]: Parallel Architectures I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

1. Introduction

Ray Tracing. Ray Tracing [Whi80] is a widely used algorithm for rendering images aiming at an high realism. It is the core technique underlying several global illumination algorithms. The input for ray tracing is a scene description that speci£es the geometry of objects together with the de£nition of every object materials, position/orientation of the lights. The output is an image of the scene as seen through a virtual camera.

For sake of clarity we will shortly summarize the ray tracing algorithm. For each pixel (x,y), in the £nal image, a ray is casted from the virtual camera through the scene, it is called *primary ray*. If exists, the £rst object is determinate. Based on the intersection point, the surface properties, the position and the color of lights, the light intensity at the intersection point is computed. In the Whitted-style ray tracing [Whi80] the ray can be re¤ected and/or refracted according to surface properties and the process is repeated recursively with these new rays. Instead, in advanced global illumination algorithms, as for instance in Rendering algorithm based on Monte Carlo approach [Shi96], several child rays have to be traced in order to compute an average of their contributions. At the end, the process adds the light intensities at all intersection points in order to get the £nal color of the pixel. Then, it is quite obvious that ray tracing require a certain computational power which is directly related to the amount of light rays as they bounce around the scene.

Coherence in Ray Tracing. Due to its high computational cost, researchers have been looking for improving the performance of ray tracing [Wal04, RSH05]. Much research has focused on fast acceleration structures and their traversal algorithms, intersection algorithms, shading models, sampling techniques. In particular, the bottleneck of ray tracing algorithm has been located in memory bandwidth and access [WS01].

Adjacent primary rays operate on almost the same data during traversal, intersection and shading. The same is true to a somewhat lesser degree for secondary rays, as shadow rays and re¤ection rays. This property is known as *spatial coherence*. A recent approach to exploit spatial coherence in order to achieve better performance is by grouping related coherent rays into groups of rays and so operate in traversal, intersection and sampling in parallel.

Adding the temporal dimension the spatial coherence can be extended de£ning *temporal coherence* [CCD90]. It can be exploited to reduce the amount of calculations needed for every new frame while rendering a sequence of frames (e.g. animation). Indeed, it is a common place that two successive frames are similar and large part of calculation can be re-used.

Formally, let p be the pixel of generic coordinates (x, y) in frame f_i and let p' be the pixel with the same coordinates (x, y) (i.e. the same pixel) in frame f_{i+1}. Let r be the ray through p and r' the ray through pixel p'. The idea of the *temporal coherence* is based upon a simple consideration: the ray r and the ray r' will follow similar paths across the scene. A common way of exploiting the *temporal coherence* is the *interpolation* [SB88, Che97]: the amount of calculations needed to render pixel p' is reduced re-using (i.e. interpolating) information calculated for pixel p.

Summarizing, several optimization techniques has been proposed to exploit *spatial* and/or *temporal coherence* where it is present, assuring coherent memory accesses and high cache hit ratio. However, in some scenarios coherence properties lack to be effective (as an example, using complex shader [CFLB06] or massive models may affect spatial coherence) and then, coherent ray tracing optimizations become useless.

Parallel Ray Tracing. Ray tracing has been de£ned "embarrassingly parallel" [FWM94] because no particular effort is needed to segment the problem in tasks and there is no strict dependency between parallel tasks. Each tasks can be computed independently from every other task in order to achieve a speed up. There are two different approaches in designing a parallel ray tracer: object-based and screen-based [CR02]. In the objects-based approach the scene is distributed among clients. For each ray casted the clients forward rays between clients. In the screen-based approach the scene is replicated on each client and the rendering of pixels is assigned to different clients. The second approach is the one investigated in this paper by a frame to frame load partitioning schema.

Key requirements of a parallelization scheme are: minimize communication overheads, balance overall load distribution, consider data locality and eventually enable a dynamic load redistribution with minimum overheads.

As common in parallel computing, approach that reach a good balancing between tasks may result in an higher overall performance. In the context of coherent ray tracing, also data locality has a notable impact in performance. Assign a task to a processor with a "similar" cache asset results in a low cache miss ratio.

Related Work. Speeding up parallel ray tracing for interactive use on multi-processor machine has received a big impulse during last years, thanks to an ef£cient implementation designed to £t the capabilities of modern CPUs [BSP06] and the use of commodity PC clusters [WBDS03]. In particular, several techniques are employed to amortize communication costs and manage load balancing.

Slusallek et al. [WBDS03] suggest task prefetching, work stealing and non-synchronous rendering, whereas Parker et al. suggest a distributed load balancer [DGBP05].

In [HA98] a competitive analysis of load balancing strategies for tray tracing has been presented. The analysis show that, for 640x480 resolution images, static load balancing strategies based upon randomization result in unacceptably high level of unbalance and poor scalability. Moreover, the authors argue that tiling strategies may behave even worse and that dynamic models based on diffusion methods have generally better results than randomization. Finally, they proposed an hybrid approach based on a diffusion method and randomization. Unfortunately, the proposed approach neither takes into account task coherence nor considers any extent to dynamic models.

Palmer et al. [PTT97] present techniques to ef£ciently exploit all levels of the deep memory hierarchy of a distributed Power Challenge Array, on which they implement a logical address space for volume blocks with caching. They discuss implications for the design of a parallel architecture suited to solve this class of problems.

Recently, ray tracing had obtained signi£cant research interest on graphics hardware. Purcell et al. [PU02] designed the £rst raytracer on the GPU that utilize an uniform grid structure. Foley et al. [FO05] designed a k-d tree acceleration structure on the GPU and showed that for some scenes this data structure yields far better performance than an uniform grid. Carr et al. [CA06] used a BVH data structure to create a raytracer suited to dynamic geometry. In any case, none of the previous GPU ray tracing outperform signi£cantly the performance of a comparable CPU implementation. In the future some actual limitations will be less strict allowing to exploit new algorithms on the GPU.

Our Result. In this paper we analyze the effectiveness of exploiting coherence in Parallel Ray Tracing. We present a load-balancing technique, based on a Prediction Binary Tree (PBT), which allows to exploit temporal coherence among successive image frames. Furthermore, we also provide two heuristics which take advantage of data-locality.
We verify the productivity of the proposed solution by running two experiments. The £rst one aims to investigate the correctness of predictions made by using the PBT.

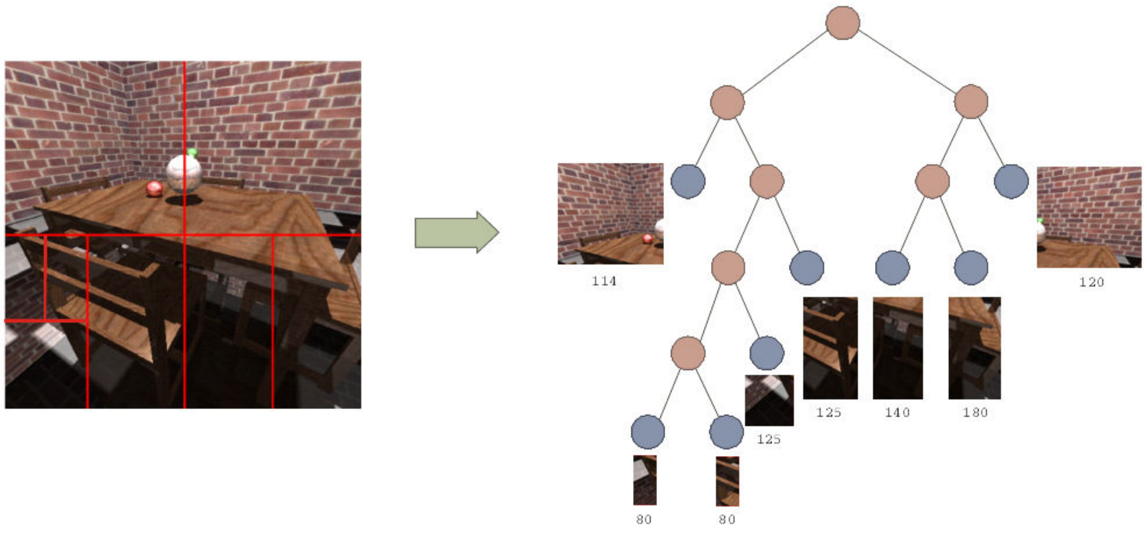


Figure 1: *An example of a PBT tree: the frame on the left has been rendered with the computation times (in ms) for each tile shown on the leaves.*

Results show that such predictions are quite accurate even considering a heavily unbalanced scene, a £ne-grained granularity and a fast moving camera. The second experiment evaluates the two locality-aware heuristics showing only a modest improvement.

Organization of the paper. In the next section we present the parallelization model analyzing in details several opposing issues in design load balancing strategies. Then, in Section 3, we introduce the PBT and describe how it can be used to exploit temporal coherence. Section 4 describes two locality aware heuristics which allow each worker to exploit spatial coherence by an ef£cient usage of its CPU cache. Experiment results are presented in Section 5. Finally, in Section 6, we conclude the paper with comments and further directions of research.

2. Load Balancing vs Data Locality

Our strategy is based on a traditional *demand driven ray tracing* approach. In this approach, the primary rays are the *Principal Data Items* (PDI), which, divided into tasks, are assigned to different workers. The global scene, which contains *Additional Data Item* (ADI) is replicated on all the workers [CR02]. This parallelization approach is particularly suited to the *Master-workers paradigm*.

In this paradigm, the master divides the whole job (the image frame to be rendered) into a set of tasks, usually represented by rectangular areas of pixels (*tiles*). Then, each task is sent to a workers which elaborates the tiles and sends back the rendered partial image. If other tiles are not yet computed, the master sends another task to the worker that has just £nished its own. Finally, the master reassembles and visualizes the rendered image and updates the scene.

Crucial point in this paradigm is the granularity of the subdivision of the image: in fact, the relationship between m, number of tiles, and n, number of workers, strongly in¤uences the performances.

There are two opposite, driving forces that act upon this design choice. The £rst one is concerned about the *load balancing* and requires m to be larger than n. In fact, if a tile corresponds to a zone of the scene whose pixels require large amount of ray-object intersection checking, then, it requires much more time with respect to a simpler tile. Then, a simple strategy to obtain a fair load balancing is to increase the number of tiles, so that the complexity of a zone of the scene is shared among different nodes.

On the opposite side, two considerations would ask for smaller m. In fact, an algorithm that has large m requires more *communication costs* than an algorithm with smaller m, both in latency (more messages) and bandwidth (communication overhead for each message). Another consideration that would require small m is *spatial coherence*. Since two rays will follow similar path if they are close, in order to make an effective usage of the local cache for each node, it is important that the tiles are large enough, so that each worker can exploit spatial coherence of tiles, having a good degree of (local) cache hits.

3. Avoid unbalancing: the PBT

In this section we present how we use a Prediction Binary Tree (PBT) to help balancing the load among the comput-

ing nodes. The PBT is in charge of directing the tiling-based load balancing strategy as follows: each frame is split into a set of m tiles (we assume, here, for sake of simplicity that $m = n$ but the arguments apply to general cases) whose size is adjusted accordingly to (an estimated) tile rendering time that is set as the computational time as measured during the preceding frame. The hypothesis is that the rendering time required by a tile on two consecutive frames are quite similar because of temporal coherence.
We, now, de£ne the Prediction Binary Trees and, then, describe an on-line algorithm which, before each frame, resizes unbalanced tiles in such a way to minimize the frame computation time.

Prediction Binary Trees. A PBT T stores the current tiling being de£ned as a rooted binary tree with exactly m leaves, in which each (internal) node has 2 children. The root of T, called r, represents the complete image frame. The (two) children of an internal node v store the two halves (more details follow on how the image is split) of the image represented by v. Consequently, each level of T represents a partition of the image frame. Moreover, each internal node v represents a tile which is the sum of the tile assigned to the leaves of the tree rooted in v and consequently, the leaves of T (henceforth $L(T)$) represents a partition of the image frame. In order to maintain a good spatial coherence, the children of an internal v node which belongs to an odd (resp. even) level of T are obtained halving the tile in t along the horizontal (resp. vertical) axes. Each leaf $e(\ell)$ also stores two variables: $e(\ell)$ that is the estimate of the time for rendering tile in ℓ and $t(\ell)$ that is time used by a worker to render (in the last frame) the tile in ℓ. Figure 1 gives an example of a PBT, with the corresponding image partition on the left.

Exploit Temporal Coherence: Updating PBT. the PBT stores the subdivision of tiles and each leaf of T is a task to be assigned to a worker. At the end of each frame, the PBT receives (with the image rendered) also the information about the time that each worker has spent on the tile. This time is received as $t(\ell)$ for each leaf, and is used as estimate by copying it into $e(\ell)$. By using the previous frame times as estimate, the PBT is ef£ciently updated for the next frame. Here we describe a provably effective and ef£cient way of changing the PBT structure so that the next frame can be executed (given the temporal coherence) more ef£ciently i.e. equally balancing the load among the processors.

We, £rst, de£ne the variance as a metric to measure the (estimated) computational unbalance that is expected given the tiling provided by the PBT T.

$$\sigma_T^2 = \frac{1}{m} \sum_{\ell \in L(T)} (e(\ell) - \mu_T)^2,$$

where $e(\ell)$ represents the time estimated to render the tile corresponding to the leaf ℓ of T and μ_T is the estimated average computational time, that is, $\mu_T = \frac{1}{m}\sum_{\ell \in L(T)} e(\ell)$. Clearly, the smaller the variance σ_T^2 is, the better is T's balancing of the load to the processors.

Given a PBT T at the end of a frame, the estimated computation time associated to each leaf, $e(\ell)$, is taken by the computation time $t(\ell)$ at the frame just rendered; then, we use a greedy algorithm that £nds the new PBT T^*. The idea of the algorithm PBT-Update (shown as Algorithm 1) is to perform a sequence of simultaneous *split-merge* operations, that consists in splitting a tile whose estimated load was "high", and merge two tiles (stored at sibling nodes) whose (combined) estimated load is "small".

Algorithm 1 PBT-Update

```
 1: $T \leftarrow CurrentPBT$
 2: for all $\ell \in L(T)$ do
 3:     copy computational time $t(\ell)$ in estimated time $e(\ell)$
 4: end for
 5: while true do
 6:     let $\ell_a$ be the leaf in $T$ with max $e(\ell)$, $\forall \ell \in L(T)$
 7:     let $\ell_{b_1}, \ell_{b_2}$ be the two siblings such that $e(\ell_{b_1}) \cdot e(\ell_{b_2})$ is minimized over all the pairs of siblings in $L(T)$
 8:     if $e(\ell_a)^2 \leq 4\, e(\ell_{b_1}) \cdot e(\ell_{b_2})$ then
 9:         return $T$
10:     else
11:         Split $\ell_a$ in $\ell_{a_1}$ and $\ell_{a_2}$        // Now $\ell_a$ is internal
12:         $e(\ell_{a_1}) \leftarrow e(\ell_a)/2$
13:         $e(\ell_{a_2}) \leftarrow e(\ell_a)/2$
14:         Merge $\ell_{b_1}$ and $\ell_{b_2}$ into $\ell_b$     // Now $\ell_b$ is a leaf
15:         $e(\ell_b) \leftarrow e(\ell_{b_1}) + e(\ell_{b_2})$
16:     end if
17: end while
```

By observing that each split-merge operation, operated by the algorithm PBT-Update, reduces the variance of the times on the tree and that the variance is positive, by de£nition, one can easily prove that the PBT-Update algorithm terminates. Our experiments also show that the PBT-Update algorithm terminates, typically, in few steps (around 5 to 10) that is, a small number of split-merge operation is enough to balance the tree. Finally, it is quite easy to show that the improvement on the variance is proportional to $e(\ell_a)^2 - 4e(\ell_{b_1})e(\ell_{b_2})$, then at each step, the greedy algorithm PBT-Update chooses ℓ_a and the siblings pair ℓ_{b_1} and ℓ_{b_2} (in lines 6-7) in order to have the higher (local) improvement in variance.
Figure 2 shows an example of the PBT-Update algorithm, which performs one *split-merge* operation, on the PBT of Figure 1.

4. Exploiting Locality by using PBT

In this Section we investigate how to use PBT in order to better exploit data locality: the idea is to let workers to better use the CPU cache. The rationale behind this investigation is that jobs carried out by two siblings workers in the PBT will follow similar memory access patterns.

Two locality-aware heuristics. To exploit locality we de£ne

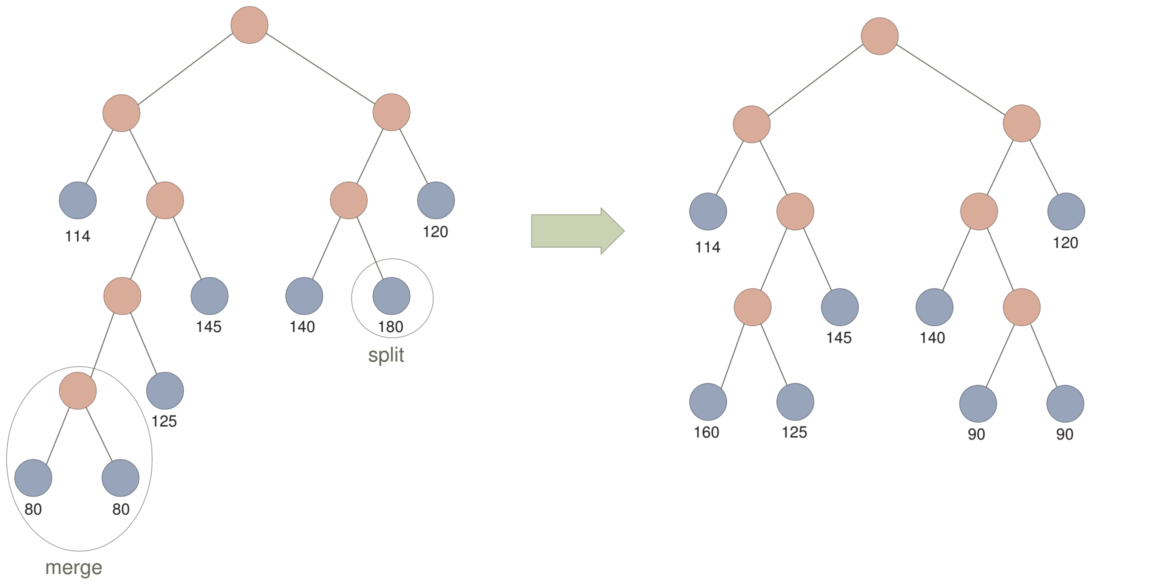


Figure 2: *A merge and split operation on the PBT tree of Figure 1 where the estimation times $e(\ell)$ drive the updates.*

the concept of *af£ne tiles*: between frames, a tile is af£ne to a processor if it has been assigned to that processor in the previous frame; two tiles are af£ne if they are "near" in the framebuffer. When a worker asks for a tile the master node tries to assign an *af£ne tile*. This de£nition clari£es the intent: to leverage af£ne tiles in order to exploit data-locality.

We implemented two heuristics in order to determinate af£ne tiles. A £rst greedy strategy, dubbed *PBT-Greedy*: on every frame, if a tile is not involved in a merge/split operation then it maintains his af£nity with the processor it has been assigned to (during previous frame). In case of tiles involved in a merge/split operation, consider a tile a splits in tiles a_1 and a_2 and tiles b_1 and b_2 merge into tile b: a_1 is assigned to processor that handled a before; b is assigned to processor that handled b_2 before; a_2 is assigned to processor that handled b_1. Just this last assignment, on the total of new 3 assignments, will, probably, not exploit cache and for this reason we say that this heuristic is $2/3$ effective in leveraging locality.

In the second heuristic, named *PBT-Visit* the af£nity is de£ned by a visit on the PBT, tiles are assigned following the in order visit. One can easily check that a subset of af£ne tiles, tiles "near" in the framebuffer, is assigned to the same processor frame-by-frame.

5. Experiments and Results

To verify the effectiveness of exploiting temporal and spatial coherence in load balancing we employed a distributed memory system, a cluster of workstations, and test scenes with remarkable unbalancing between tiles.

Setting of the experiments. Our hardware test platform is a IBM BladeCenter cluster of 33 nodes (1 master node, 32 worker nodes). Each node is a Intel Pentium IV processor running at 3.20 GHz with 1MByte cache, 1 GB of main memory and CentOS 5 Linux as operating system with OpenMPI version 1.1.1 for message passing. All the nodes are interconnected with a Gigabit Ethernet network.

Temporal coherence tests. We tested our scheme on a modi£ed ERW6 test scene (about thousand primitives). We create unbalancing by changing shading properties of the objects, in particular, we used full re¤ective materials, computational intensive (used for table, chairs, ¤oor and other objects) and quite simple diffusive ones (for walls), less computational intensive. The scene has four light sources (see Figure 1 (left)). In both test scenes, we have a £xed walkthrough of the camera throughout the scene, with translations in all directions and rotations too. The image resolution is 512x512 pixels.

In order to explore the performances of PBT in exploiting temporal coherence, we checked it under different conditions. First we tested two different task granularity (we recall that we consider $m = k \cdot n$, where m is the number of tiles and n is the number of worker): we have chosen $k = 1$ (one tiles for each worker) and $k = 4$ (four tiles, on average, for each workers). Moreover, we considered two different camera speed (1x and 2x). In all the tests the number of workers is $n = 32$. To make a comparison, we measured the total amount of tiles which has been estimated correctly using 85^{th}, 90^{th} and 95^{th} percentile (see Table 1). As an example row 2 (Perc. 90^{th}) represents the percentile of estima-

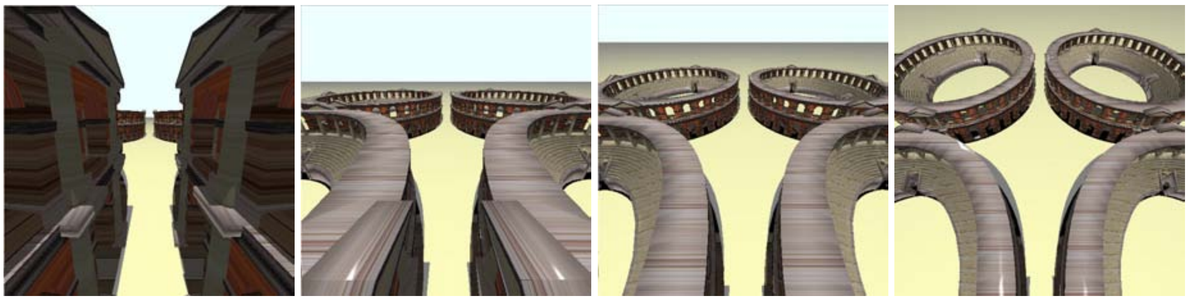


Figure 3: *Images generated during the walk-through of the scene used for spatial coherence tests.*

tions having an error up to 10%. In other words, when $k = 1$ and the camera speed is 1x the 93.2% of estimations have an error smaller than 10%, while when $k = 4$ and the camera speed is 2x the 79.8% of estimations have an error smaller than 10%.

Corr. Perc.	$k = 1$ **x1**	$k = 1$ **x2**	$k = 4$ **x1**	$k = 4$ **x2**
85	96.2%	95.3%	92.1%	89.7%
90	93.2%	92%	86.2%	79.8%
95	92.6%	84%	68%	55%

Table 1: *Results of the predictions in* 85^{th}, 90^{th} *and* 95^{th} *percentile.*

Spatial coherence tests. In this test we investigate the effectiveness of data locality in exploiting the processors' cache. The test consider 4 different scenes with an increasing number of triangles, ranging from less than 30000 to about 950000. In Table 2 are reported the number of triangles and the number of nodes in the Kd-tree. The difference between scenes is an increasing number of amphitheaters (all of them have simple diffusive shaders). Scenes are animated by a prede£ned walk-through of the camera. The frame resolution is 512x512 pixels.

The rationale behind the test is to verify that, whenever the size of the Kd-tree is bigger than the cache size, a drop of the performance can be measured, due to the number of cache misses.

Test scene	Triangles	Nodes in Kd-tree
1 Amphitheater	29759	251017
4 Amphitheater	119026	999237
16 Amphitheater	476098	3970097
32 Amphitheater	952130	7970097

Table 2: *A simple description of the test scenes used for spatial coherence tests. For each scene, we report the corresponding number of primitives and the Kd-tree's size.*

All tests shows that improvements obtained by using the PBT (with both the *locality-aware* heuristics, see Section 4, and a random assignment named *PBT-Random*) with respect a static random assignment. With more details, on small scenes (see Figure 4 (up)), the whole scene £ts into the cache and then the assignment strategy does not matter. When the scene become larger, so that it does not £t into the cache (see Figure 4 (down)), the data locality also provides a modest improvement of the performances of the system (around $1-5$ ms) using both the *locality-aware* heuristics.

6. Conclusions and Future Works

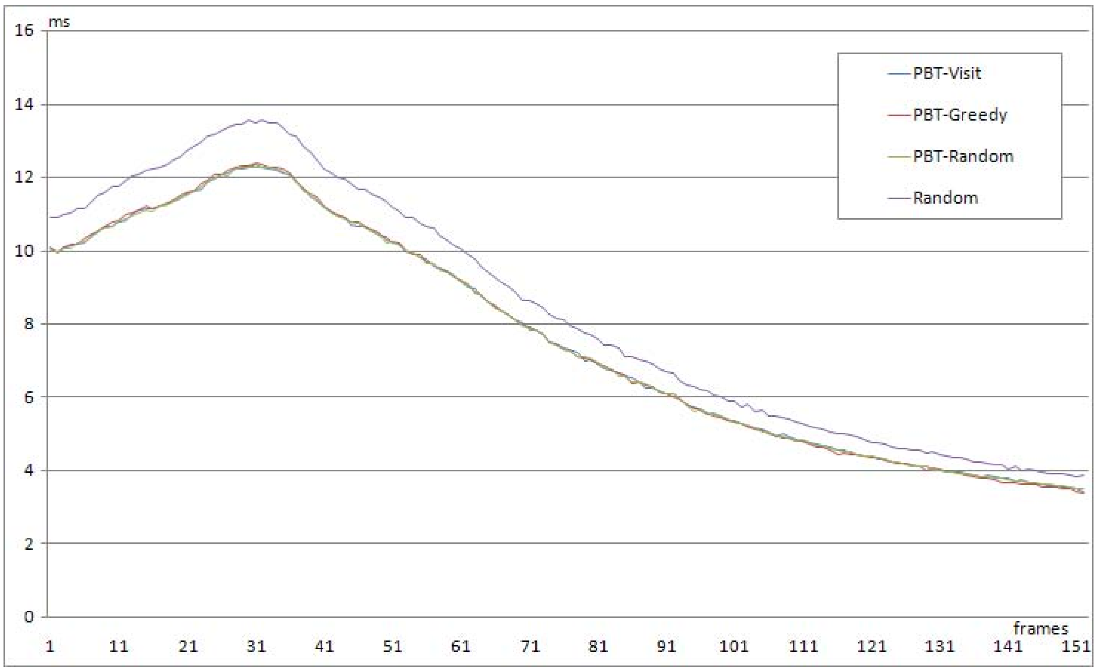
In this paper we have described a load balancing technique that is able to exploit both spatial and temporal coherence in Parallel Ray Tracing. Our strategy divides the original rendering problem in balanced subtasks and distributes them to independent processors through a Prediction Binary Tree (PBT).

The PBT allows to exploit temporal coherence between successive frames. At each new frame, the PBT is updated using a cost function which uses previous rendering time as cost estimate. The PBT-Update performs split-merge operations to let the PBT balancing the load among the nodes and reducing the variance of the computing times.

The PBT is also useful in leveraging data locality. Two locality-aware heuristics have been used in tile scheduling with the purpose of assigning af£ne tiles to the same processors, in two consecutive frames. Heuristics have had different performance improvements.

Temporal coherence showed larger bene£ts in accelerating the PRT. In the case of locality-aware heuristics we have measured smaller improvements. In our experiments we tried to stress the cache performances by using scenes larger and larger in such a way that whole kd-tree would not £t into the cache. This is not the best way to challenge cache performances, as shown in [CFLB06]. In future tests we will increase the unbalancing of the scene by using more computational intensive shaders.

A future work is to integrate the PBT algorithm with other common techniques for manage load balancing. Common



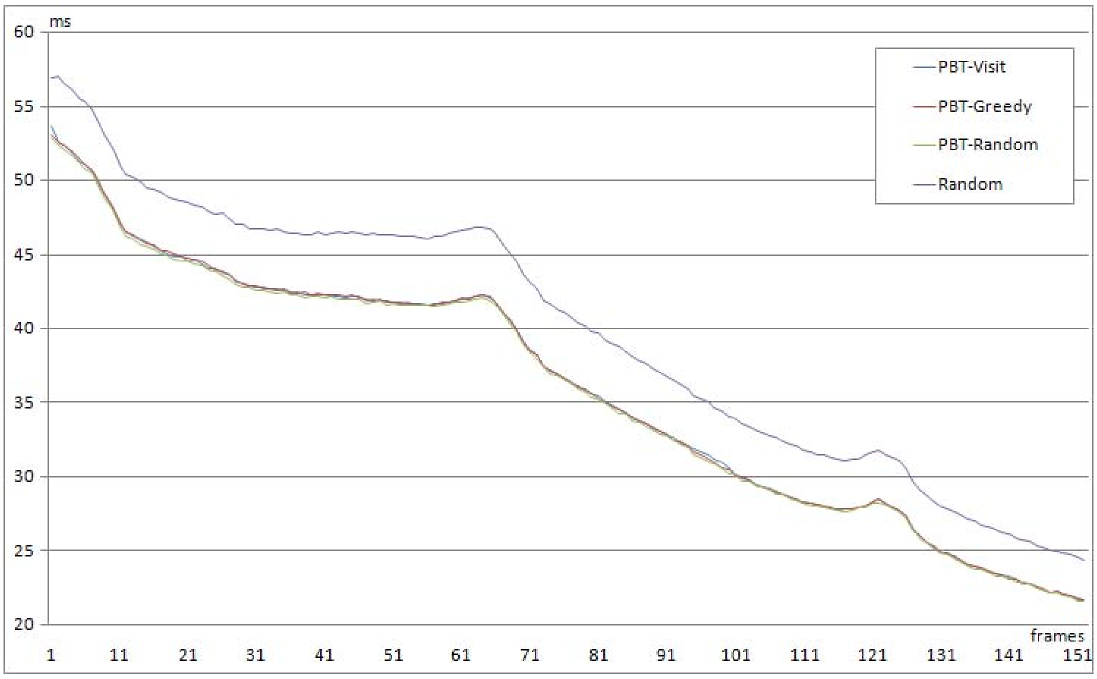


Figure 4: *Rendering time per tile. (up) test scene "1 Amphitheater", (down) test scene "32 Amphitheater".*

techniques used in parallel computing to address load balancing are focused on task subdivision e redistribution; these techniques can be effectively integrated with the PBT. In the particular scenario of work stealing [BL99], we envision that the employ of PBT can signi£cantly reduce the number of work steals with a valuable performance impact. Another appealing scenario of use of the PBT to support distributed load balancing schema.

References

[BL99] BLUMOFE R. D., LEISERSON C. E.: Scheduling multithreaded computations by work stealing. In *Journal of ACM* Vol. 46(5) (1999), pages 720–748.

[BSP06] BIGLER J., STEPHENS A., PARKER S.: Design for parallel interactive ray tracing systems. In *IEEE Symposium on Interactive Ray Tracing* (Los Alamitos, CA, USA, 2006), IEEE Computer Society, pages 187–196.

[CA06] CARR, N. A., HOBEROCK, J., CRANE, K., AND HART, J. C.: Fast gpu ray tracing of dynamic meshes using geometry images. In *Proceedings of Graphics Interface 2006*, Canadian Information Processing Society.

[CCD90] CHAPMAN J., CALVERT T. W., DILL J.: Exploiting temporal coherence in ray tracing. In *Proceedings on Graphics interface '90* (Toronto, Canada, 1990), pages 196–204.

[CFLB06] CHRISTENSEN P., FONG J., LAUR D., BATALI D.: Ray tracing for the movie 'cars'. In *IEEE Symposium on Interactive Ray Tracing 2006,* (Sept. 2006), pages 1–6.

[Che97] CHEVRIER C.: A view interpolation technique taking into account diffuse and specular inter-re¤ections. In *The Visual Computer*, Vol. 13(7) (1997), pages 330–341.

[CR02] CHALMERS A., REINHARD E. (Eds.): In *Practical Parallel Rendering*. A. K. Peters, Ltd., Natick, MA, USA, 2002.

[DGBP05] DEMARLE D. E., GRIBBLE C. P., BOULOS S., PARKER S. G.: Memory sharing for interactive ray tracing on clusters. In *Parallel Computing,* Vol. 31(2) (2005), pages 221–242.

[FWM94] FOX G. C., WILLIAMS R. D., MESSINA P. C.: *Parallel Computing Works!* Morgan Kaufmann, May 1994.

[FO05] FOLEY, T., AND SUGERMAN, J.: Kd-tree acceleration structures for a gpu raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM Press, New York, NY, USA, 15-22.

[HA98] HEIRICH A., ARVO J.: A competitive analysis of load balancing strategies for parallel ray tracing. In *The Journal of Supercomputing,* Vol. 12(1–2) (1998), pages 57–68.

[PTT97] PALMER M. E., TAYLOR S., TOTTY B.: Exploiting Deep Parallel Memory Hierarchies for Ray Casting Volume Rendering. In *IEEE Parallel Rendering Symposium* (1997), Painter J., Stoll G., Kwan-Liu Ma, (Eds.), pages 15–22.

[PU02] PURCELL, T. J., BUCK, I., MARK, W. R., HANRAHAN, P.: Ray tracing on programmable graphics hardware. In *ACM Trans. Graph.* (2002), pages 703–712.

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-level ray tracing algorithm. In *ACM Transaction on Graphics (TOG),* Vol. 24(3) (2005), pages 1176–1185.

[SB88] SIG BADT J.: Two algorithms for taking advantage of temporal coherence in ray tracing. In *The Visual Computer,* Vol. 4(3) (1988), pages 123–132.

[Shi96] SHIRLEY P.: Monte Carlo Methods for Rendering. In *ACM SIGGRAPH '96 Course Notes CD-ROM - Global Illumination in Architecture and Entertainment*, (1996), pages 1–26.

[Wal04] WALD I.: Realtime Ray Tracing and Interactive Global Illumination. *PhD thesis, Saarland University* (2004).

[WBDS03] WALD I., BENTHIN C., DIETRICH A., SLUSALLEK P.: Interactive Distributed Ray Tracing on Commodity PC Clusters – State of the Art and Practical Applications. *In Proceedings of EuroPar '03, Lecture Notes on Computer Science 2790* (2003), pages 499–508.

[Whi80] WHITTED T.: An improved illumination model for shaded display. In *Communications of the ACM*, Vol. 26(6) (1980), pages 313–349.

[WS01] WALD I., SLUSALLEK P.: State of the Art in Interactive Ray Tracing. In *Eurographics 2001 State of the Art Reports* (2001), pages 21–42.