Evaluation of Adaptive Subivision Schemas for Parallel Ray Tracing

Biagio Cosenza[1]
ISISLab, Dipartimento di Informatica ed Applicazioni,
Universitá degli Studi di Salerno, Italy

Abstract

Ray tracing algorithm is widely used for rendering images aiming at an high realism. Speeding up ray tracing for interactive use with parallel architectures has received a big impulse during last years. Despite several techniques are employed in order to amortize communication costs and manage load balancing, they still represents a bottleneck to the scalability. We consider a new way of manage load balancing based on adaptive subdivision, assuring higher scalability and performance, compatible with commonly used balancing techniques such as work stealing and work sharing.

1 Introduction

In computer graphics, rendering is the synthetic production of realistic images from a mathematical description of a scene. Ray Tracing algorithm [1] is a powerful rendering algorithm integrating reflection, refraction, hidden surface removal and shadows into a single model. Ray Tracing, in the simplest form, traces a ray for each pixel, from an eye or view point through the pixel and into the scene. Secondary rays can occur e.g. when rays hit reflective or refractive surfaces, or for shadow, generating a huge computational cost.

Parallel Ray Tracing. Considerable efforts have been spent in order to investigate new ways to overcome the high computational demands of Ray Tracing [12]. Improving performance to interactive rates requires to combine highly optimized ray tracing implementations with massive amounts of computational power. Thanks to the recent advances in both hardware and software, it is now possible to create high quality images at interactive rates on commodity PC clusters (see [13]). Cluster of workstation have been demonstrated successful in the context of Parallel Ray Tracing (see e.g. [10, 9, 11]), but they still require some efforts to manage load balancing and communications hiding.

Parallel Ray Tracing has been defined an "embarrassingly parallel" problem [5] because no particular effort is needed to subdivide problem in tasks and there is no strict dependency between parallel tasks. Each task can be computed independently from every other task in order to achieve a good speed up.

[1] cosenza@dia.unisa.it

There are two different approaches in designing a Parallel Ray Tracer: object-based and screen-based [3]. In the objects-based approach the scene is distributed among clients. For each ray casted, the clients forward rays to the responsible of the scene area. 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 and in this context each tile, a rectangular area of the screen space, represents a task, the Principal Data Item (PDI). Instead the global scene, the Additional Data Item (ADI), is replicated on all worker [3].

A Parallel Ray Tracing system can be synchronous, if after each frame there is visualization barrier (e.g. Bigler et all. in [8])., or asynchronous if rendering is performed asynchronously to the application (e.g. Wald et all. in [13]).

Outline. In this paper we investigate a new approach on affording load balancing in the context of Parallel Ray Tracing. We propose a screen-based synchronous implementation where load balancing is afforded during the subdivision phase instead during the assignment phase. Our approach provides an adaptive subdivision of the whole problem in balanced task and is compatible with well-known load balancing strategies. Indeed our strategy is based on a traditional demand driven ray tracing. This parallelization suits the Master-worker paradigm: a master node subdivides the whole job into a set of tasks (tiles) and then each task is sent to a worker node, which renders the tile and sends back the partial image. Our parallel implementation is tailored for distributed memory architectures such as a cluster of workstations, but our work can be easily extended to shared memory architectures.

2 The Load Balancing Problem

In parallel computing, the key to achieve a good speedup is to keep processors supplied with tasks, so that the resulting scheduling is as greedy as possible. Assuring a perfect load balancing between processors is not easy if tasks have strongly different computing time.

Ray Tracing is a time consuming process and the rendering time of each pixel is influenced by a multitude of factors. The are several sources of unbalancing between pixels computing time, such as shading algorithms, acceleration data structure queries, anti aliasing techniques, texture accesses. We experienced that shading, and in particular the number of secondary rays generated by a surface, can heavily influence the pixel computing time.

We remark that in a screen-based parallelization, unbalancing between pixel cause unbalancing between task.

2.1 Granularity of the decomposition

An important choice in the parallelization is the granularity of the subdivision of the image in tiles. The relationship between m, the number of tiles, and n, the number of processors stongly affects the performance.

There are two opposite, driving forces that act upon this design choice. The first one is concerned about the *load balancing* and requires m to be larger than n. A simple and common 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, several considerations would ask for smaller m. 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 (see [7] for a more detailed discussion on coherence in Parallel Ray Tracing).

A common strategy in parallel ray tracing implementations is to subdivide image in equally sized tiles, then use some load balancing strategy in scheduling (see [8, 10, 9, 11]). If load balancing is still a problem, this approach suggests to use a finer granularity. However a fine granularity introduces an high overhead, especially for high unbalanced scenes. Particularly in a distributed memory system the choice of the granularity is critical due the higher cost of the communication.

3 Adaptive Subdivision Techniques

We introduce a new approach on affording load balancing focused on subdivision.

Our solution (see Figure 1) introduces a new component in the system design, the *predictor*, that is able to do an estimate of the computation time need by a tile. The use of the estimate will be used in order to obtained a more balanced subdivision (i.e. an *adaptive subdivision*).

The idea is that a more balanced subdivision improves both load balancing and performances. Kruskal and Weiss [4] investigated a theoretical model where they show how variance among task compute time, hence load unbalancing, affects performance.

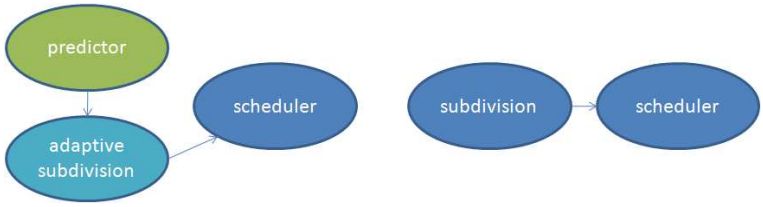

Figure 1: Our system design introduces a predictor in the subdivision (left). A common system usually lack of this component (right).

Predictive Heuristic. The design of the parallel system is independent of how the prediction is obtained. In our system we exploit *temporal coherence* among successive frames in order to obtain a per-tile estimate of the computing time. Given a tile, we suppose that the computing time required by the tile in the frame t is comparable with the time spent in the frame $t+1$.

This estimate works fine on our test scenes. However it presents three main problems: it can be wrong for strongly dynamic scenes; it can occur on discretization problems if tiles are extremely small; and it can not be used for the first frame. These problems suggest the introduction of new predictive heuristics in order to have a more scene-independent system.

A balancing schema: the PBT. In order to help balancing we present a balancing schema based on a Prediction Binary Tree (PBT).

The PBT is in charge of direct the subdivision in balanced tiles. A PBT T stores the current subdivision in tiles in a full binary tree with exactly m leaves, in which each (internal) node has 2 children. The root of T, called r, represents the complete image (i.e., the main tile). The (two) children of an internal node v store the two halves of the tile represented by v. Consequently, each level of T represents a partition of the image. 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. In order to maintain a good spatial coherence and minimize tasks interaction, the children of an internal node v which belongs to an odd (resp. even) level of T are obtained halving the tile in t along the horizontal (resp. vertical) axes. This assure that tiles have an almost-square shape (i.e. one dimension is at most twice the other). Each leaf $\ell \in L(T)$ also stores two variables: $e(\ell)$ that is the estimate of the time for computing tile in ℓ and $t(\ell)$ that is time used by a worker to compute (in the last frame) the tile in ℓ. Figure 2 gives an example of a PBT, with the corresponding image partition on the left.

Updating the PBT. The PBT stores the subdivision in tiles of the whole image. 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 it. 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 computing times as estimate, the PBT is efficiently updated for the next frame. Here we describe a provably effective and efficient way of changing the PBT structure so that the next frame can be executed (given the temporal coherence) more efficiently, i.e. equally balancing the load among the processors.

We, first, define the variance as a metric to measure the (estimated) computational unbalance that is expected given the subdivision 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

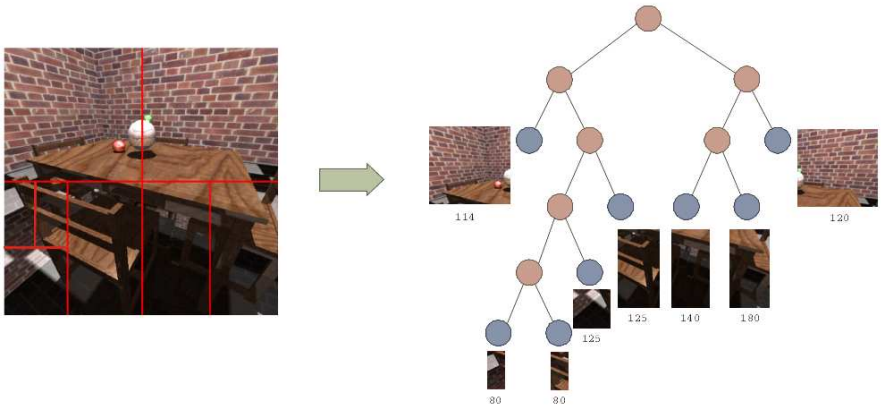

Figure 2: An example of a PBT tree: the frame on the left has been rendered with the computation times (in ms) for each tile shown on the leaves.

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.

Given a PBT T at the end of a frame, the estimated computing time associated to each leaf, $e(\ell)$, is taken by the computing time $t(\ell)$ at the frame just rendered; then, we use a greedy algorithm, the $PBT-Update$, that finds the new PBT T^*. The idea of the algorithm $PBT-Update$ 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".

Pseudo code, correctness and convergence of the $PBT-Update$ algorithm can be found in [6].

The design of the parallel system is also independent from the balancing schema and it can be changed without affect the rest of the system (e.g., using an interpolation-based schema instead of the PBT).

Overhead of the approach. The time spent by the master node in evaluating a new subdivision schema from a prediction is pure overhead introduced by our approach. In our implementation this overhead corresponds to the time spent in subdividing the image in tiles and updating the PBT. Note that this time is serial and affects performance once per frame. However results show that this overhead is negligible (usually few ms) compared with the improvement in balancing and performance (see Section 4 and Figure 7).

Use of adaptive subdivision with known balancing techniques. Many load balancing strategies are used in Parallel Ray Tracing. They mainly focus on scheduling and does not involve the subdivision phase.

If only a single tile is assigned to each client at any time, a client runs idle between sending its results back to the server and receiving the next tile to be computed. With *task prefetching*, it is possible to avoid these latencies, by sending several *prefetched* tiles in advance to each client. *Task prefetching* is suitable also if the subdivision schema is adaptive.

A popular and practical method to handle load balancing in Parallel Ray Tracing is *work stealing* [2], in which processors needing work steal tasks from other processors. Also this method only focus in scheduling and can be easily combined with our schema.

Another interesting scenario is the coupling of a distributed load balancer (e.g. based on work stealing) with our balancing schema.

4 Results

To verify the effectiveness of the adaptive subdivision schema in load balancing we employed a distributed memory system, a cluster of workstation, and test scenes with remarkable unbalancing between tiles.

Our hardware test platform was *cacau*, a NEC Xeon EM64T Cluster available at HLRS High Performance Computing Center at the Stuttgart Universität. For our test we used up to 64 nodes, each equipped with 2 Intel Xeon EM64T processors and 1 GB of main memory, interconnected with a Infiniband 1000 MB/s.

The test scenes were two modified version of the standard ERW6 test scene, with about one thousand primitives (see Figure 3). The test scenes have different shading properties and unbalancing is due to different shading setting. In both test scenes, we have a predefined walk-through of the camera around the scene, with movements in all direction and rotations too. The image resolution is 512x512 pixels. We obtained similar results in the two test, so in the following we refers to the result of the first test scene.

Serial Ray Tracing implementation trace a ray at once. A kd-tree has been used due on acceleration data structure, build with well known Surface Area Heuristic. The kd-tree implementation also provides a fast traversal by using a cache friendly data layout. Our implementation uses Intel SSE extensions for speed up shading, by wrapping a RGB color in a 128 bit word. The parallel version use MPI for node communications, and is based on the master-worker paradigm.

Scalability. We tested our schema based on the PBT with a normal regular subdivision schema with 2, 4, 8, 16, 32 and 64 processors, in order to measure the effective scalability of our strategy (see Figure 6). The tests shows that our schema works always better than the regular, and it presents almost linear scalability up to 32 processors. However the tests also show that with more than 32 processors the use of adaptive subdivision is not enough in order to assure scalability.

Optimal granularity. We also tested several test set in order to obtain the optimal granularity (i.e., the optimal choice of m, the number of tiles) with 2,

4, 8, 16, 32 and 64 processors (see Figure 5). The rationale behind this test is to determinate how our schema affects the tuning of the granularity in the parallel implementation, compared with a simple regular subdivision schema.

Test results show involve several interesting considerations. The optimal granularity for the PBT comes with a lower number of tiles, in respect of the normal regular approach. In particular the PBT work better with coarser tiles, introducing beneficial effects cause the lower overhead of communication. For the same reasons, the PBT become more effective when the test asset need a finer granularity, due to the high number of processors and the strong unbalancing.

Parallel render time analysis. The last test set is focused on how the whole compute time (i.e., the parallel rendering time) is spent. Out purpose is to determinate how unbalancing affects performance, the ratio between time spent by the workers in local rendering and communications, how much time is spent by the master node in serial code. Figure 7 shows the test results for 16 and 32 processors at different granularity. The time spent in updating the PBT is proportional to m, hence to the granularity and the number of processors, but in our test is always lower than 5 ms. Thus the overhead by the adaptive subdivision is small compared with the gain in balancing.

5 Conclusions

In this paper we have described a load balancing technique for Parallel Ray Tracing that is based on an adaptive subdivision of the domain in balanced tasks. Our strategy introduces in the system design a *predictor*, able to do an estimate of the compute time need by a task. In our implementation, we divide the image-space in balanced tasks by using a Prediction Binary Tree, exploiting temporal coherence between frames to obtain the estimate.

We have showed that our strategy provides a better scalability compared with regular subdivision schema. The optimal granularity for different number of processors (see Figure 5) needs a lower number of tiles m, compared with a regular subdivision schema. Moreover the overhead introduced by the adaptive subdivision schema is small compared with the gain in balancing and, hence, assure better performance.

It maybe worth interesting to integrate our adaptive subdivision with other well-known balancing schema. For example, by using work stealing with adaptive subdivision we expect beneficial effect due to the smaller number of work steals.

We suggest to use similar adaptive subdivision also in different problem (see [6]).

6 Acknowledgment

The work has been performed under the Project HPC-EUROPA++ project (project number: 211437), with the support of the European Community -

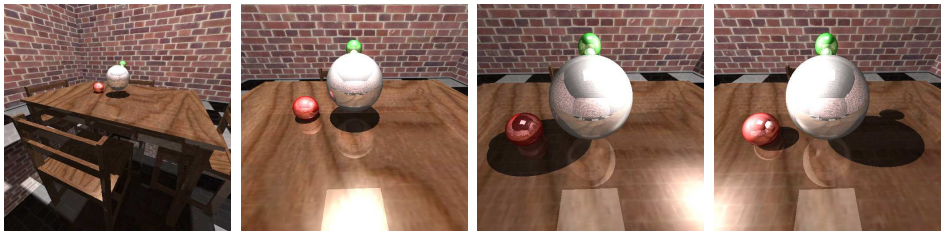

Figure 3: Images generated using our parallel ray tracing.

Research Infrastructure Action of the FP7 "Coordination and support action Programme". Thanks to Thomas Ertl and Carsten Dachsbacher of the Institut für Visualisierung und Interaktive Systeme (VIS) at the Stuttgart Universität for supporting the project and helpful discussions .

References

[1] T. Whitted. An improved illumination model for shaded display. *Communications of ACM*, 1980

[2] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of ACM*, 46(5):720-748, 1999.

[3] Alan Chalmers and Erik Reinhard. *Pratical Parallel Rendering*. A. K. Peters, Ltd., 2002.

[4] Clyde P. Kruskal andAlan Weiss. Allocating Independent Subtasks on Parallel Processor. *IEEE Transactions on Software Engineering*. 1985.

[5] Fox, Geoffrey C. and Williams, Roy D. and Messina, Paul C. *Parallel Computing Works!*. Morgan Kaufmann. 1994.

[6] Biagio Cosenza, Gennaro Cordasco, Rosario De Chiara, Ugo Erra, and Vittorio Scarano. Load Balancing in Mesh-like Computations using Prediction Binary Trees. *Proc. of 7th International Symposium on Parallel and Distributed Computing*, 2008.

[7] Biagio Cosenza, Gennaro Cordasco, Rosario De Chiara, Ugo Erra, and Vittorio Scarano. On Estimating the Effectiveness of Temporal and Spatial Coherence in Parallel Ray Tracing. *In Proc. of 6th Eurographics Italian Chapter Conference*, 2008.

[8] J. Bigler and A. Stephens and S.G. Parker. Design for Parallel Interactive Ray Tracing Systems. *IEEE Symposium on Interactive Ray Tracing*, 2006.

[9] Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive Global Illumination using Fast Ray Tracing. *Rendering Techniques*, 2002.

[10] IngoWald, Philipp Slusallek, and Carsten Benthin. Interactive Distributed Ray Tracing of Highly Complex Models. *Proceedings of the 12th Eurographics Workshop on Rendering*, 2001.

[11] David E. DeMarle and Christiaan P. Gribble and Solomon Boulos and Steven G. Parker. Memory sharing for interactive ray tracing on clusters. *Parallel Computing*, 2005.

[12] Ingo Wald and Philipp Slusallek. State-of-the-Art in Interactive Ray-Tracing. *State of the Art Reports, Eurographics*. 2001.

[13] Ingo Wald, Carsten Benthin, Andreas Dietrich, and Philipp Slusallek. Interactive Ray Tracing on Commodity PC clusters. State of the Art and Practical Applications. *EuroPar*. 2003.

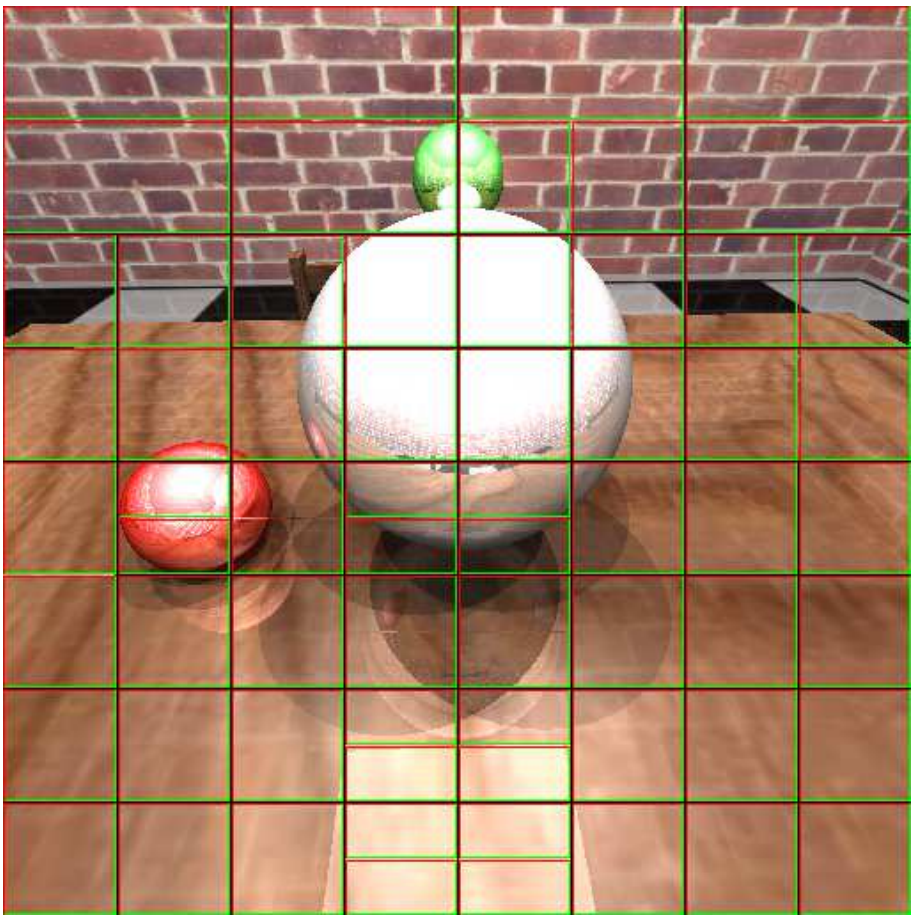

Figure 4: An example of adaptive image space subdivision. The most expensive pixels are located between two reflective surfaces, because rays bounce between them. The cheaper rays simply hit a diffusive surface (e.g., bricks wall). The subdivision adapts to the estimated cost, by further splitting expensive tiles.

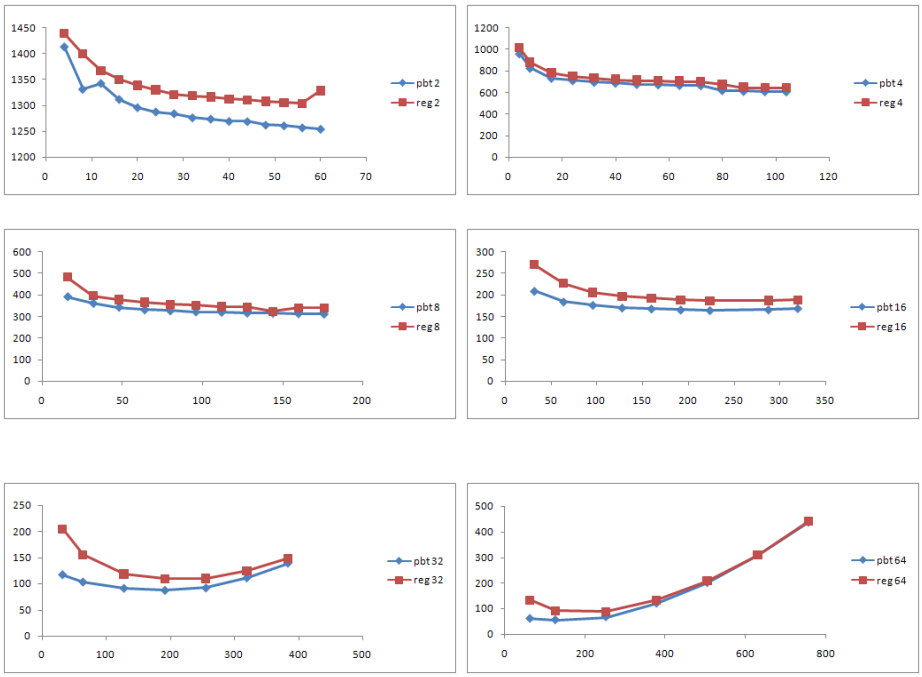

Figure 5: Optimal subdivision granularity with both regular (red) and adaptive subdivision with the PBT (blue). Test done with 2, 4, 8 , 16, 32 and 64 processors. The horizontal axis represents the number of tiles, whereas vertical axis represents rendering time.

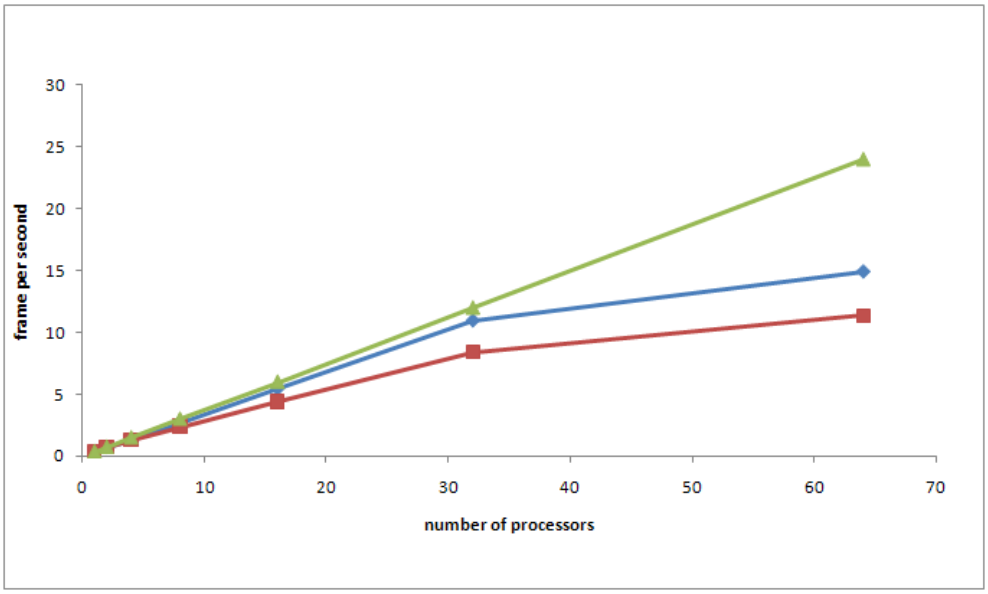

Figure 6: Scalability. Comparing regular subdivision (red), adaptive subdivision with the PBT (blue), linear theoretical speedup (green). Adaptive heuristics shows a close to linear speed-up up to 32 processors.

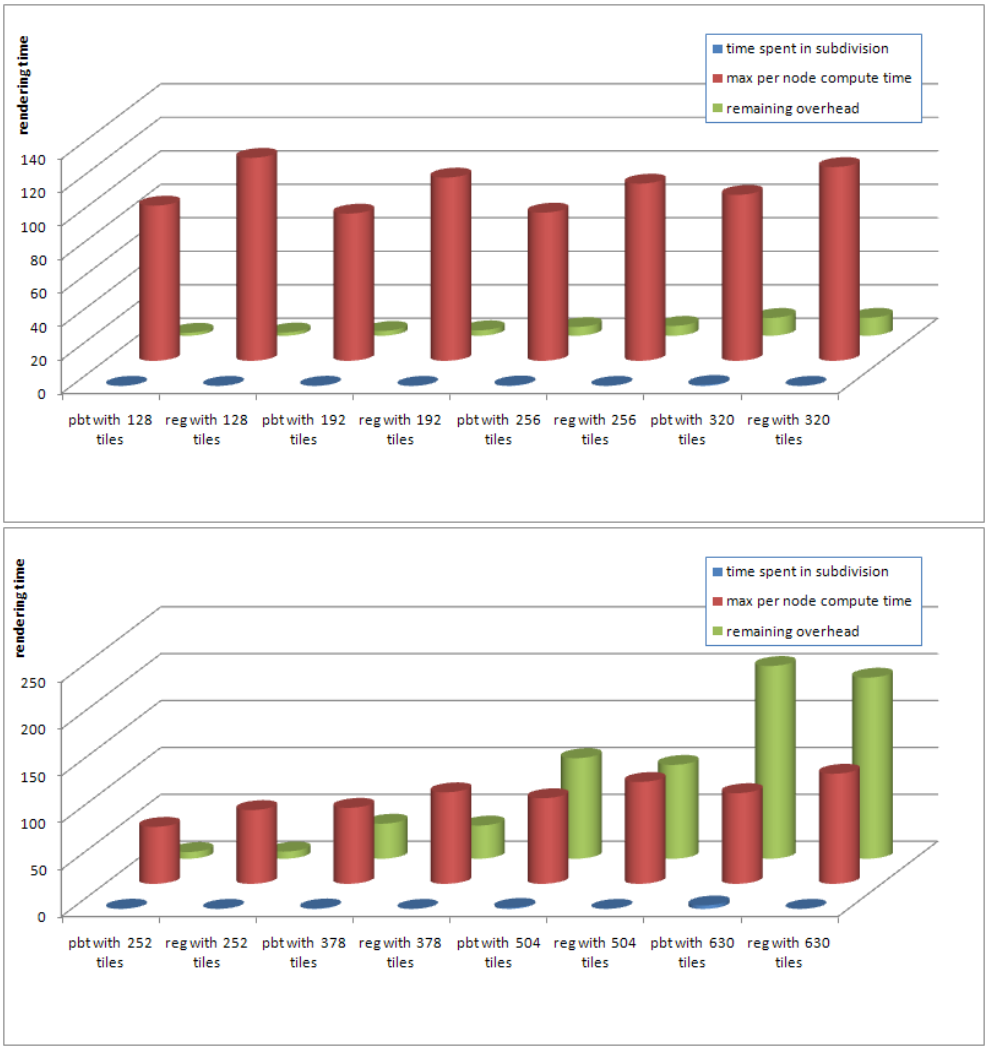

Figure 7: Contributions in rendering time with 32 (up) and 64 (down) processors, at different granularities, with both adaptive and regular subdivision techniques. The total rendering time is split in: the time spent in subdivision (i.e. update the PBT for adaptive subdivision); the maximum per-node compute time, such as an estimate of the load balancing; the remaining time, mostly due by communications. Showed render times are the average between a test scene of 600 frames.