Parallel Rendering with Radiance

Luís Paulo Santos*
Dep. de Informática – Universidade do Minho
Braga – Portugal
e-mail: psantos@di.uminho.pt

Alan Chalmers
Dept. of Computer Science – University of Bristol
Bristol – United Kingdom
e-mail: alan@compsci.bris.ac.uk

July 10, 2003

Abstract

Interactive global illumination at interactive rates is still an unachieved goal for medium to large complexity scenes and high fidelity illumination models, such as the one used by Radiance. In order to achieve this goal several different optimization approaches must be exploited, including parallelism, perceptual issues and low–level optimizations. This work concentrates on developing a parallel message passing prototype of Radiance's renderer targeted towards producing animations. This prototype is used to study scalability issues and perceptual optimizations that will reduce rendering times and will, hopefully, be unnoticed by the viewer. The preliminary results obtained suggest that efforts should concentrate on reducing idle times due to suboptimal load distribution and indirect irradiance calculation times. Also psychophysical experiments and perceptual metrics should be carried on, in order to assess the impact of selective rendering on perceived quality.

1 Introduction

High fidelity interactive rendering is one of the main objectives pursued by the computer graphics community. However, if both specular and diffuse inter reflections are included on the illumination model, the time required to render scenes with medium to high complexity precludes such objective.

In order to sustain the frame rate required for human interaction the rendering tasks and image quality must be adapted such that a perceptually "correct" image is rendered within the time constraints. This adaptation may evolve along several axes:

- include image perception techniques to allow selective resolution and rendering quality for different regions of the image according to their perceptual importance;
- fine tune the code and data structures to the underlying architecture, such that computer resources, such as instruction pipelines, caches, etc., are fully exploited;
- use parallel computers to speed up rendering times by effectively distributing the workload among the available resources.

*Work done as a Visiting Fellow at the Dept. of Computer Science – University of Bristol – United Kingdom, partially supported by grant SFRH / BPD / 11622 / 2002 of the Portuguese Fundação para a Ciência e Tecnologia

The current work focuses on this latter axis. It takes place within the context of a broader "Rendering on Demand" research project, whose goal is to provide high quality global illumination rendering of highly complex scenes through the Web. The goals of the current work are to assess the computational costs and main obstacles associated with parallel global illumination, propose approaches to solve some of these problems and provide an initial code structure to implement a parallel renderer. A parallel version of the Radiance lighting visualization renderer `rpict` has been developed with the aim of:

- provide insight to how Radiance code is structured and how can it be effectively rewrote as a parallel application;
- profiling Radiance rendering algorithm, in order to understand its time costs structure;
- understand how these costs scale with the size of the underlying parallel system;
- provide a faster renderer which can be used transparently in place of the standard `rpict`.

One of Radiance's main contribution to global illumination is the way indirect irradiance is computed and accounted for in the illumination model. Although Radiance strives to reduce this computation requirements by reconstructing the indirect contribution from a sparse set of samples, it still represents a major percentage of the whole rendering time. In strict accordance with Amdahl's Law this work concentrates on this particular cost and suggests some techniques to reduce this cost.

2 Radiance

Radiance is a physically based global illumination image rendering and analysis system. It consists on a large number of different tools, including object modelling, format conversion, rendering, image processing and displaying. The renderer is a eye-forward ray tracer, with extensions to efficiently solve the rendering equation [7] under most conditions.

Radiance emphasis is on lighting visualization, rather than photorealism, although the latter occurs as a consequence of the former. Lighting visualization focuses on rendering images that correspond closely to what would be seen in reality (limited to the accuracy of the geometric and goniometric model [6]), whereas photorealism focuses on rendering images that "look real", but which may be very different from the corresponding real scene.

To ensure physical accuracy computations are performed in physical quantities, radiance being the selected metric since all other lighting metrics can be computed from this one. The illumination model includes the most common mechanisms of light interaction with objects such as specular, diffuse and directional-diffuse reflection and transmission in any combination and up to any level. Participating propagation mediums are also supported. Accurate models of light sources and generic materials are included with Radiance standard distribution. The renderer combines deterministic and stochastic ray tracing techniques in order to achieve the best balance between speed and accuracy [4, 21, 24].

Radiance author, Greg Ward Larson, presents the following main design goals:

Accurate calculation of radiance and luminance – Luminance is a photometric quantity that represents the quantity of visible radiation, as perceived by an average human, passing through a point in a given direction, measured in $candela/m^2$. The radiometric equivalent of luminance is radiance, which represents the radiant energy flowing through a point in a given direction, measured in $watts/(steradian.m^2)$. Radiometric quantities are not weighed according to the average human eye sensitivity as their photometric equivalents. Spectral radiance simply adds a wavelength dependence to radiance [3]. Radiance (the software) produces predictions of the

distribution of these values in modelled environments, therefore enforcing its physically based nature.

Modelling of both electric and day light – In order to model electric light accurately a variety of luminaires' models are included, built from measured or calculated output distribution data from light fixtures. Modelling daylight accurately implies following the initial intense radiation from the sun and redistributing it through its reflections from other surfaces and scattering from the sky – Radiance includes a tool to generate sky light distributions according to the geographic position and day time.

Support for a variety of reflectance models – The accuracy of the global illumination calculation depends critically on the accuracy of the used local illumination model, since this determines how light will be reflected, absorbed or transmitted by each surface. Radiance includes several different surface material types, each based on approximations to the actual physics of light interaction, rather than derived for algorithmic convenience. One of these types allows the user to completely specify the bidirectional reflectance distribution function (BRDF).

Support for complicated geometry – Radiance data structures were carefully designed in order to avoid exponential memory and/or processing complexity growth with the number of surfaces.

Take input from CAD systems – Recognizing that CAD systems are the most adequate and widespread geometric modelling tools, converters are supplied with Radiance to import models from a variety of such systems with minimal user intervention.

2.1 Illumination Model

Radiance uses recursive eye forward ray tracing to evaluate an adapted version of Kajiya's rendering equation [7, 21, 22, 24] at each intersection point, as specified by equation 1.

$$L_r(\theta_r, \phi_r) = L_e(\theta_r, \phi_r) + \int_0^{2\pi} \int_0^{\frac{\pi}{2}} L_i(\theta_i, \phi_i) f(\theta_i, \phi_i, \theta_r, \phi_r) |\cos(\theta_i)| \sin(\theta_i) \partial\theta_i \partial\phi_i \tag{1}$$

where:

θ	is the polar angle measured from the surface normal
ϕ	is the azimuthal angle measured about the surface normal
$L_r(\theta_r, \phi_r)$	is the reflected radiance
$L_e(\theta_r, \phi_r)$	is the emitted radiance
$L_i(\theta_i, \phi_i)$	is the incident radiance
$f(\theta_i, \phi_i, \theta_r, \phi_r)$	is the bidirectional reflectance distribution function ($steradian^{-1}$)

On an ideal world, with infinite resources or time, the most elegant solution to this integral would be to use uniform stochastic sampling (Monte Carlo), by randomly spawning rays across a significant number of directions. However, in the real world, the convergence would be so slow that this solution is impractical. This is because Monte Carlo sampling relies on a stochastic process that requires too many samples to find local maxima in:

incident radiance – arriving both from primary light sources, such as the sun and light fixtures, and from virtual light sources, such as mirrors[1];

[1] To account for light reflected from highly specular surfaces Radiance may designate these as virtual light sources, therefore moving these calculations to the direct component of the integral evaluation.

the BRDF – such as the specular reflection and/or transmission directions.

These local maxima are often localized in small solid angles, compared to the sampling hemispheric domain, therefore they would require an huge number of sampling rays in order to be found and appropriately accounted for.

> *"The key to fast convergence is in deciding what to sample by removing those parts of the integral that can be computed deterministically and gauge the importance of the rest so as to maximize the payback from our ray calculations [21]"*

This is what most ray tracers (Radiance included) do. Shadow rays are spawned explicitly towards light sources and specular rays are spawned near[2] the mirror angle for reflective materials and in the refracted direction for dielectric surfaces, such as glass (see figure 1).

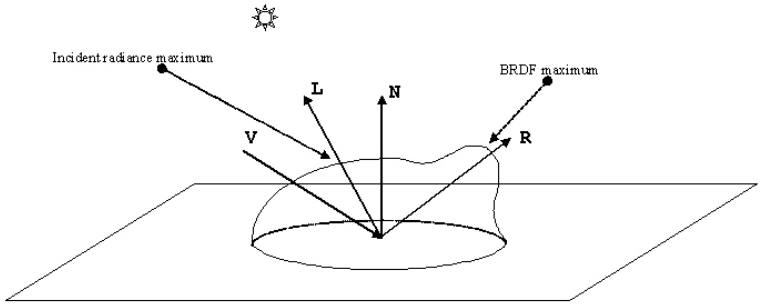


Figure 1: Example of a BRDF and local illumination model maxima

By removing these local maxima from the integral, variance between samples is reduced at a comparatively reduced cost. This approach was introduced at the same time as ray tracing by Whitted [25]. The remaining of the integral accounts for diffuse interreflections, i.e., Lambertian contributions from non self-emitting surfaces. Most ray tracers approximate this component with a constant ambient term, but this results in flat shaded areas inappropriate for lighting visualization (figure 3).

Indirect Irradiance Calculation and Caching

Computing diffuse interreflections amounts to stochastically sample the hemisphere centered on the intersection point, carefully avoiding the direct and specular directions so that the same radiance is not accounted for twice. This is the integral of radiances over the hemisphere (see equation 2), which

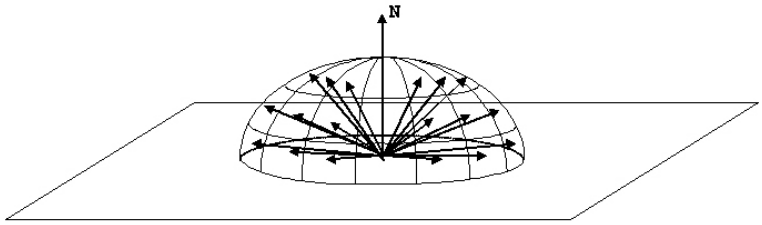


Figure 2: Sampling the hemisphere centered on the intersection point

is not dependent on direction. This physical quantity is referred to as irradiance and is measured in

[2]Radiance stochastically distributes specular rays about the mirror and transmitted directions to increase accuracy [4].

$watts/m^2$. Its photometric equivalent is illuminance, measured in Lux. The important point to note is that, since it is not dependent neither on incident nor on reflected directions, this value is view point independent.

$$E = \int_0^{2\pi} \int_0^{\frac{\pi}{2}} L_{ind}(\theta_i, \phi_i) \cos(\theta_i) \sin(\theta_i) \partial\theta_i \partial\phi_i \tag{2}$$

where:

θ	is the polar angle measured from the surface normal
ϕ	is the azimuthal angle measured about the surface normal
E	is the indirect irradiance
$L_{ind}(\theta_i, \phi_i)$	is the indirect incident radiance

The indirect irradiance is calculated spawning a few hundred rays over the hemisphere in an uniformly weighted, stratified Monte Carlo sampling:

$$E = (\frac{\pi}{M * N}) \sum_{j=0}^{M-1} \sum_{i=0}^{N-1} L(\theta_j, \phi_i) \tag{3}$$

where:

E	is the indirect irradiance
$L(\theta_j, \phi_i)$	is the indirect incident radiance in direction (θ_j, ϕ_i)
(θ_j, ϕ_i)	$= (\arcsin \sqrt{\frac{j+X_j}{M}}, 2\pi \frac{k+Y_k}{N})$
X_j, Y_k	are uniformly distributed random variables in the range (0,1)
$M * N$	is the number of samples, with $N \approx \pi M$

The cost of computing indirect irradiance at each point is prohibitive, since it requires spawning hundreds of rays to reduce variance to acceptable levels. However, since indirect irradiance changes gradually over surfaces and is view independent, it is possible to compute it extensively only at a few points and interpolate its value over neighbouring pixels to obtain a result that is smooth and accurate at a modest cost. Radiance follows this approach, searching for previously computed irradiance values in the neighbourhood of each intersection point and interpolating if successful; if no previous values are cached, then the extensive computation is performed. The domain of validity of each computed irradiance value depends on the user selectable accuracy and on the local geometry. To accurately interpolate among several irradiance values, or to extrapolate whenever only one value is available, information on how irradiance changes as a function of point location or surface orientation is also used in the form of the gradient of the irradiance function. The gradient is available at no cost from the hemisphere sampling process and allows the use of a second order interpolation function, rather than a linear one, resulting in more accurate predictions.

Although irradiance values are view–independent, for each scene only the values required for the current view are computed, since this process is triggered on demand whenever for a new intersection point no valid precomputed neighbours are available.

Radiance uses an octree based caching scheme to store indirect irradiance values. These values may be stored on a file to be used on posterior renderings of the same scene, such as, for instance, in different frames of an animation, dramatically reducing rendering times. It should be noted that irradiance values are independent of the view point, but they are not independent of the scene geometry, therefore can not be reused if this changes. For further details about Radiance's illumination model and particularly about indirect irradiance calculation, caching and interpolation see [21, 22, 23, 24].

2.2 Rendering Parameters

Radiance renderers accept a large number of parameters. Many of these can be set, and even adaptively tuned in runtime, in order to balance the quality of the final image with the rendering time. A comprehensive subset of such parameters is presented in the following tables.

	Comment	**Quality**		
		Low	**Medium**	**High**
-dt	Sets the threshold for selective shadow testing.	0.2	0.1	0.05
-dc	Sets the certainty for selective shadow calculations.	0.25	0.5	0.17

Table 1: Parameters for adaptive light source testing.

	Comment	**Quality**		
		Low	**Medium**	**High**
-ds	Sets the ratio for area light sources subdivision.	0.0	0.3	0.01
-dc	Sets the degree of jittering.	0.0	0.5	0.6

Table 2: Parameters for area light sources subdivision.

	Comment	**Quality**		
		Low	**Medium**	**High**
-dr	Sets the depth of virtual light sources creation.	0	1	3
-dp	Sets the number of pretesting rays.	128	512	4096

Table 3: Parameters for virtual light sources.

3 Parallel Radiance

To achieve the ultimate goal of rendering high fidelity animations at interactive rates using Radiance's global illumination model, the structure of time costs associated with executing this algorithm must be well understood. A prototype parallel version of `rpict` (Radiance renderer) has been developed with the aim of:

- profiling Radiance rendering algorithm, in order to understand its time costs structure;
- understand how these costs scale with the size of the underlying parallel system;
- provide a faster renderer which can be used transparently in place of the standard `rpict`.

3.1 Related Work

The standard Radiance distribution exploits coarse-grained parallelism on both distributed and shared memory architectures [24]. For single frames an image space decomposition is used, which uses different processes to render different subregions of the image. Tasks are assigned to processes on a demand–driven fashion, processes getting more work whenever they get idle. A NFS shared file is used to control task assignments. On shared memory SMP systems several processes are launched using `fork()` UNIX system call and `rpict -PP` command line switch. On distributed memory systems `rpiece` is used to

	Comment	Quality		
		Low	Medium	High
-st	Sets the coefficient threshold for specular sampling.	0.5	0.1	0.01
-sj	Sets the degree of jittering.	0.0	0.7	1.0

Table 4: Parameters for specular sampling.

	Comment	Quality		
		Low	Medium	High
-av	Sets the constant ambient value.	–	–	–
-aw	Sets the ambient weight.	0.0	0.0	0.0
-ab	Sets the number of ambient bounces.	0	1	?
-ad	Sets the number of rays used to sample the hemisphere.	64	512	1024
-as	Sets the number of ambient supersamples when a given area of the hemisphere exhibits high variance.	0	64	512
-aa	Sets the ambient accuracy. (maximum error for indirect irradiance interpolation).	0.4	0.2	0.08
-ar	Sets the ambient resolution.	–	–	–
-af	File name to store indirect irradiance values.	–	–	–

Table 5: Parameters for indirect irradiance.

control several different `rpict` processes. Indirect irradiance values are shared among processes using an "ambient" file (-af command line switch). This file sits on a NFS server and is written and read by every process whenever it has computed a predetermined number of irradiance samples. Global sharing among all processes is thus achieved, drastically reducing computation times. For animations time space decomposition is applied, using multiple `rpict` processes, each rendering complete frames. Indirect irradiance values can still be shared using an ambient file. A general animation control program, `ranimate`, is provided, which distributes frames across many processors. Frame coherence may be exploited by interpolating between frames, using `pinterp`.

The standard Radiance approach does not use any parallel programming special library, relying only on UNIX `fork()` and NFS to launch, coordinate and communicate among coarse grained processes. Although the availability of NFS is not a problem on all main UNIX distributions, some file lock managers are not designed for fast and frequent shared file access, leading to high inefficiencies when the number of parallel processes exceeds a certain system dependent number. Koholka et al. [8] developed a parallel version of Radiance on top of MPI. This also a demand–driven master–slave image space decomposition, although dynamic partitioning is employed to assure a correct load balancing towards the end of the computation – at the beginning of the computation relatively large subregions of the image are sent to the workers, but these subregions become smaller with the progress of the calculation; near the end the tasks are so small they can be executed fast enough to get a good load-balance in most cases. Instead of using a NFS shared ambient file to communicate diffuse interreflection values, blocks of 50 such values are broadcasted to all other slaves. Blocks of values are used instead of single values to reduce communication, although this may result in more extensive indirect irradiance calculations being performed. They report efficiencies above 80% using up to 20 processors, when the scenes' complexity is large enough. Results are presented only for still images.

Robertson et al. [11] developed an MPI–based parallel version of Radiance for real time interactive walkthroughs. An image–space decomposition demand–driven master/slave architecture is used. Ambient values are communicated by each slave processing element (PE) to an ambient master processor, which the returns to the slave PE ambient values accumulated from other processors – broadcasts are

avoided and PEs only communicate new ambient values after a certain number of them has computed. To further reduce communication, PEs compress their subimages prior to sending them to the master. In order to achieve interactive frame rates, these system exploits frame coherence by using a point-cloud method: the 3D positions of primary rays intersections with the scene are retained (together with the respective shading result) in scene coordinates and reprojected into the viewing coordinate systems of subsequent frames. The authors claim that a walkthrough using the point cloud maintains a high degree of accuracy for a large number of frames, since the viewpoint rarely changes by a large extent. A pixel reuse rate of 85% .. 98% is achieved, but the authors goal of 10 frames per second is still far away, even without indirect irradiance computations although a 64–processor Cray T3E was used. Furthermore, geometric reprojection is prone to geometric and specular shading artifacts. There is, however, a trend towards caching of previously computed geometric and shading information. For details see [15, 16, 17, 19, 18, 20].

Reinhard et. al [10] developed a parallel version of `rpict` over PVM. This version is targeted towards very complex scenes, whose description does not fir into a single processor local memory. The scene description must thus be distributed across the system. Each processor holds a resident working set of objects and caches those objects that are required by the work in hand. An hybrid scheduling is used, tasks being subdivided into two different types:

Data–driven tasks are processed by the processors that hold the objects required to intersect and shade rays. The tasks migrate among processors in order to be matched with the required data.

Demand–driven tasks are assigned to a processor whenever it has no work assigned. Data required to intersect and shade rays is remotely fetched from the processors holding it.

Data driven tasks may cause large load imbalances, while demand driven tasks may require a lot of data communication. Therefore, tasks are designated demand driven if they exhibit high ray coherence, such as bundles of primary and shadow rays, and designated as data driven otherwise. Hybrid scheduling provides the opportunity to render large scenes. Data driven tasks assure background workload, while demand driven tasks keep the load balanced. The resulting efficiency is not exceptionally high, but is not strongly dependent on the number of processors resulting in reasonable scalability up to 80 processors.

3.2 System Architecture

The computation of each pixel in ray tracing is completely independent of all other pixels: ray tracing is, therefore, embarrassingly parallel and well suited for image space decomposition. This is the approach taken in the current version of parallel Radiance. Each processing element is a `rpict` process, which reads all scene data and all previously computed ambient values from the file system, therefore replicating all data items at each process address space. A master process subdivides the image plane onto a number of regions and assigns these regions (tasks) to the processing elements on demand[3]. The number of tasks is larger that the number of processing elements to provide for load balancing. Load imbalances may occur due to the different requirements of each image region and different processing power across the processing elements. Tasks get progressively smaller towards the end of each frame computation, in order to achieve better load balance.

The rendering times can be reduced by exploiting coherence among the different pixels. Radiance explores coherence by interpolating indirect irradiance samples (ambient values) among neighboring points, rather than extensively computing this component at each intersection point. The sequential version of Radiance shares ambient data between different frames by storing computed values on an

[3]In the current version these regions are horizontal slabs with the same width as the final image, reducing the amount of available parallelism and locality within each task. Future versions will have tasks of arbitrary width.

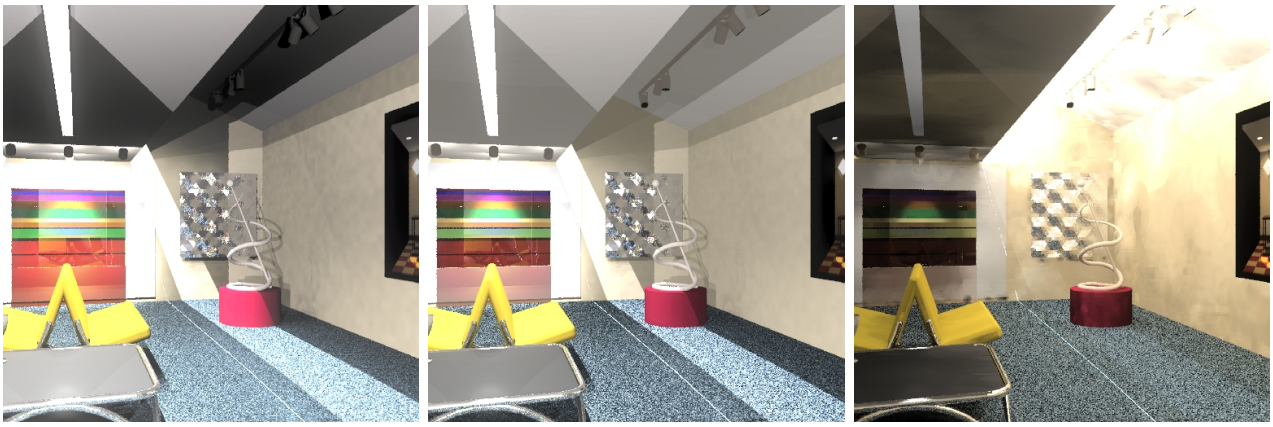
ambient file, which is read in by later executions of Radiance, thus avoid recomputing them. Ambient values are also shared among different processing elements rendering different regions of an image, by sharing the ambient file on a NFS server and regularly writing out locally computed values and reading in remotely computed ones. This NFS–based approach exhibits very poor scalability. An ambient master/slave version was developed, where slave PEs send ambient values to an ambient master process in blocks of N_A values, using MPI message-passing functions. The ambient master stores these new values in the ambient file and returns to the PE ambient values computed by other processors. This approach avoids locks to the ambient file, since only the master accesses it; the master can, however, become a bottleneck, as in all centralised approaches.

A master/slave architecture is used both to allocate tasks in a demand-driven approach and to collect/distribute indirect irradiance values. The same process is used as a master for both these roles. The number of processes to launch is always one more than the number of intended working slaves. On all experiments presented throughout this report only the number of working PEs is reported on tables and graphics and used to compute metrics such as speed–up and efficiency.

The parallel version of Radiance is referred to as `prpict`, standing for "parallel rpict". General changes to Radiance standard distribution are documented in appendix A. mpich 1.2.5 was used and the code compiled using gcc version 2.96 and the -O2 optimization level. Experiments were performed on a network of 21 workstations, each with an AMD Athlon XP 1800+ processor, running at 1533 MHz, with 512 MB of memory. The workstations are connected by a fully switched 100 Mbits/s FastEthernet. These workstations are used by the Department's undergraduate students, but experiments were carried out during the night to minimize interferences.

3.3 Performance Analysis

The experimental data reported throughout this section was obtained using with the Gallery Art scene described in chapter 3 of [24] (figure 3). All indirect irradiance calculations were performed using 2 levels of ambient bounces and the image resolution is 512x512 pixels.



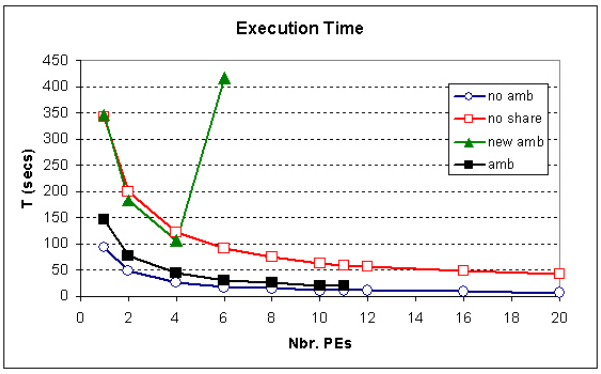
(a) No ambient calculation (b) Constant ambient term (c) 2 ambient bounces

Figure 3: The Art Gallery scene

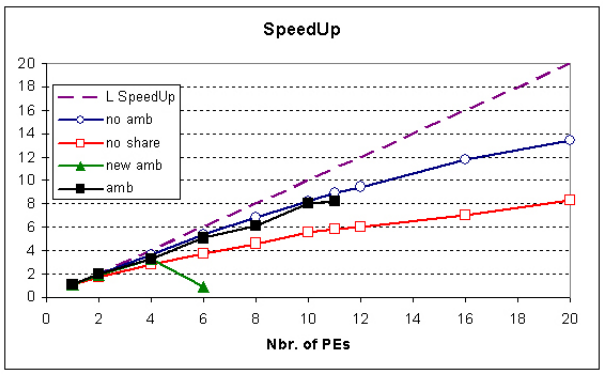
3.3.1 Scalability with NFS ambient values sharing

Figure 4 depicts the execution time, speed–up and efficiency as a function of the number of slave processing elements for the following rendering settings: no ambient calculations (no amb), no ambient calculations sharing (no share), new ambient file (new amb) and existing ambient file (amb). The

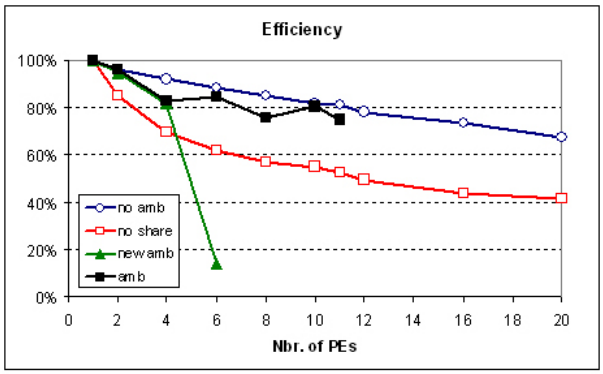
mechanism used to share and store the ambient (indirect irradiance) values is a NFS shared ambient file. The following conclusions can be made from these graphics:



(a) Execution Time



(b) Speed–up



(c) Efficiency

Figure 4: Art Gallery – NFS execution times and speed–up

- If ambient computations are disabled, performance scales quite well until 11 PEs (for this particular scene), but the efficiency drops below 80% if the number of slaves is further increased. This is because ray tracing is an embarrassingly parallel algorithm, but there is not enough work to keep all slaves busy until the end of the computation.
- If ambient computations are performed, but no mechanism for sharing such values among PEs is enabled, then scalability is quite poor and efficiency drops below 70% for 4 PEs and below 45% for 20 PEs. This is because the PEs are performing too much replicated work to compute overlapping indirect irradiance samples.
- If an ambient file is used the NFS system starts thrashing when the number of slaves increases. With a new ambient file this happens with only 4 PEs, since it is being written very often, while if the file already exists 6 PEs may still be used, since the access rate is lower. With larger number of PEs the execution times grow exponentially.

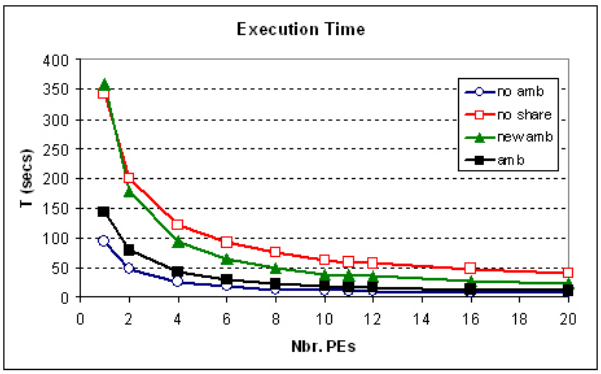
The execution times decrease significantly when an ambient file is used, but the exponential cost of an NFS shared file render this technique ineffective when more than 6 processors are used.

3.3.2 Scalability with message passing ambient values sharing

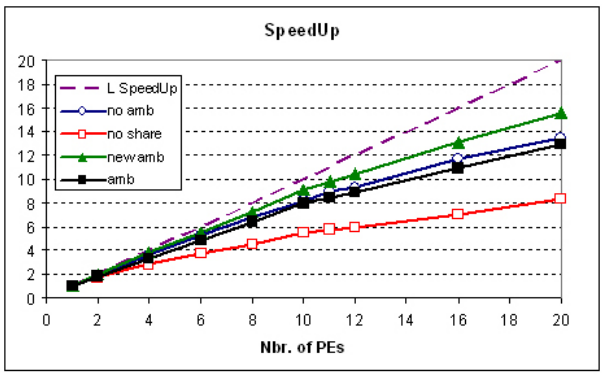
To avoid the NFS bottleneck a different scheme was used to share and store ambient values. The master process, which distributes work on demand to slaves, also acts as an ambient master. The

ambient master initially reads ambient values stored in the ambient file and broadcasts them to all slaves. After computing N_A new ambient values a slave PE will communicate these values to the master and receive whichever new values have been computed by other slaves. Collective communications are thus avoided, although the ambient master might become a bottleneck. This strategy is labelled "CTR" throughout this report, standing for centralized.

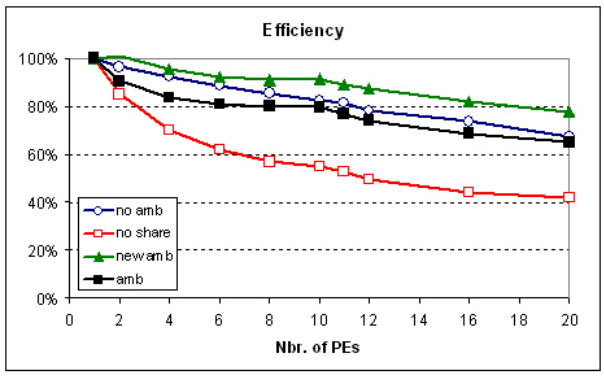
Figure 5 depicts the execution time, speed–up and efficiency as a function of the number of slave PEs. This strategy scales quite well, achieving an efficiency of 82% with 16 PEs and 78% with 20 PEs, when



(a) Execution Time



(b) Speed–up



(c) Efficiency

Figure 5: Art Gallery – CTR execution times and speed–up

new ambient values are being computed (this represents the most common and most time consuming situation).

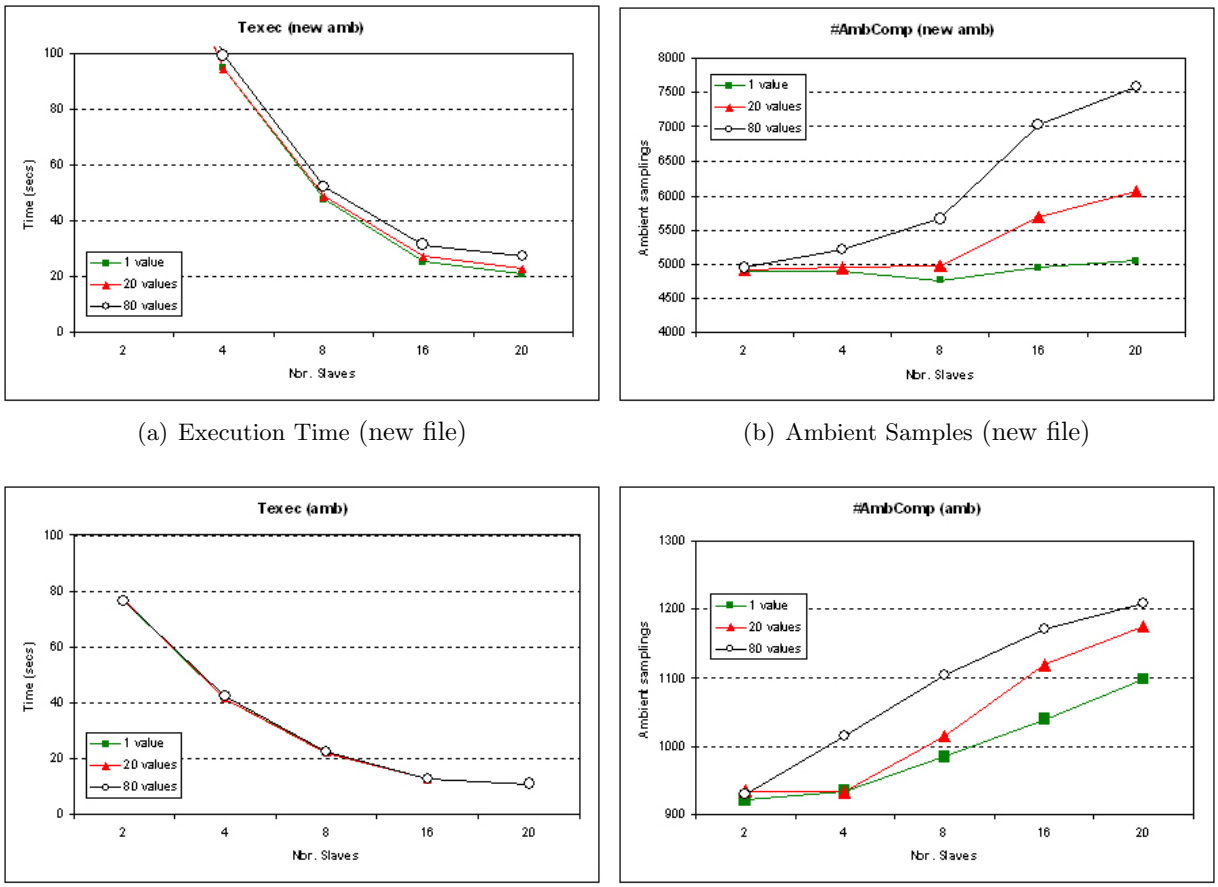
The number of ambient values each PE must compute before synchronizing with the master, N_A, must impact on the overall performance. If N_A is small synchronization occurs frequently, increasing the communication volume and the burden on the centralized server, but the number of computed ambient samples is minimized, since these values are often distributed to all PEs. If N_A is large the number of ambient samples computed by each PE increases, but the burden on the communication medium and ambient server is reduced. The impact on application performance, measured in terms of execution time, depends on many factors:

cost of computing each ambient sample – the higher this cost, the smaller should N_A be. This value depends on the scene being rendered, the computer system used and the rendering parameters;

communication costs – these depend on the latency (t_0) and bandwidth (r_∞), as seen by the application, i.e., including packing and unpacking times, protocol conversion times, network contention, etc.; N_A should increase with both t_0 and r_∞;

contention on the central server – if contention gets too high, then N_A should be increased in order to free the server.

Figure 6 shows the execution times and number of ambient samples calculations for different ambient block sizes with 16 slave PEs. It can be seen that for this particular scenario better results are achieved with $N_A = 1$. It is also evident that execution time is highly correlated with the number of extensive ambient computations. All results presented on this report were obtained with $N_A = 20$, except where explicitly said otherwise.



(a) Execution Time (new file)

(b) Ambient Samples (new file)

(c) Execution Time (existing file)

(d) Ambient Samples (existing file)

Figure 6: Art Gallery – execution times and nbr. of ambient samples as a function of N_A (16 PEs)

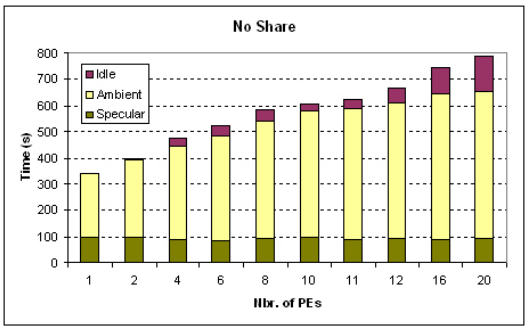
The ideal value of N_A will be different for different computer systems, for different scenes and can even change in time for the same computer system and scene – it is dependent on the view point and system utilization. An adaptive policy that dynamically changes N_A may thus be appropriate.

An optimal parallel algorithm would always achieve linear speed–up. However, in the real world, parallel computing introduces additional costs which prevent the speed–up from growing linearly. These costs are associated with: suboptimal distribution of load across the computing resources, replicated work, communication overheads and parallelism management. In order to better understand why `prpict` is not achieving linear speed–ups, the time it takes to execute different parts of the algorithm was measured:

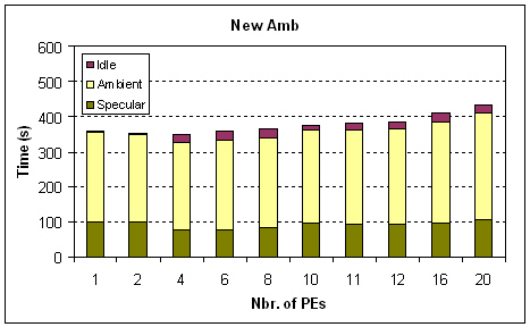
- `Specular` includes all computations required to intersect and shade rays not related with the indirect irradiance calculations; this includes primary rays, specularly reflected and transmitted rays and shadow rays and is denoted by T_{spec};

- `Ambient` includes all computations associated with rays originating in ambient sub-trees (it includes all types of rays, except primary ones); this metric is denoted by T_{amb};
- `Idle`, T_{idle}, includes all idle times, i.e., times when a PE had no work assigned, either because it was waiting for a new task or because no further work was available but some PEs were still busy.

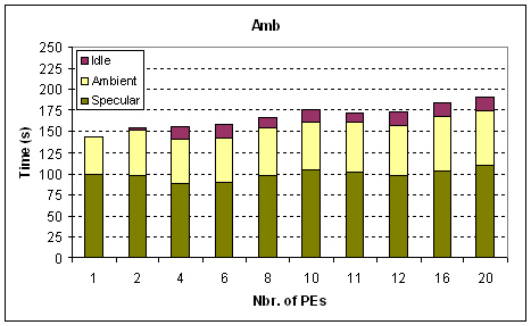
Rather than presenting these measurements for some particular PE, figure 7 presents the sum of such measurements across all slaves – these are referred to as aggregated metrics and are denoted by T^{Agg}_{spec}, T^{Agg}_{amb}, T^{Agg}_{idle}. Ideally the aggregated time should be constant and independent on the number of PEs, therefore resulting in linear speed–up. Any increase in aggregated times represents a penalty, while any reduction represents an added benefit that would allow the application to achieve super linear speed–up. Figure 7(a) presents aggregated times when ambient computations are being performed



(a) No share



(b) New ambient file



(c) Existing ambient file

Figure 7: Art Gallery – Aggregated times

but not shared among PEs, figure 7(b) presents the same measurements when ambient values are shared and figure 7(c) presents these metrics when ambient values computed during a previous render are being used (resulting in more interpolations and less extensive ambient computations). T^{Agg}_{spec} is fairly constant due to the embarrassingly parallel nature of ray tracing. Idle times, however, increase with the number of PEs, making the case for a more efficient load distribution strategy [12, 13]. T^{Agg}_{amb} behavior depends on the sharing strategy used. If ambient computations are not being shared, this metric increases linearly with the number of PEs. This is due to replicated work. If ambient values are shared, then there is still a sublinear increase with the number of PEs, which, together with increasing idle times is responsible for the suboptimal speed–ups depicted in figure 5. Figure 8 illustrates the relationship between T^{Agg}_{exec} and the number of extensive ambient computations performed. As the number of processes increases, the number of extensive ambient computations also increases (due to suboptimal sharing), causing a closely correlated increase on aggregated execution times (correlation factors: 0.9999 for `no share`, 0.9901 for `new amb` and 0.9659 for `amb`).

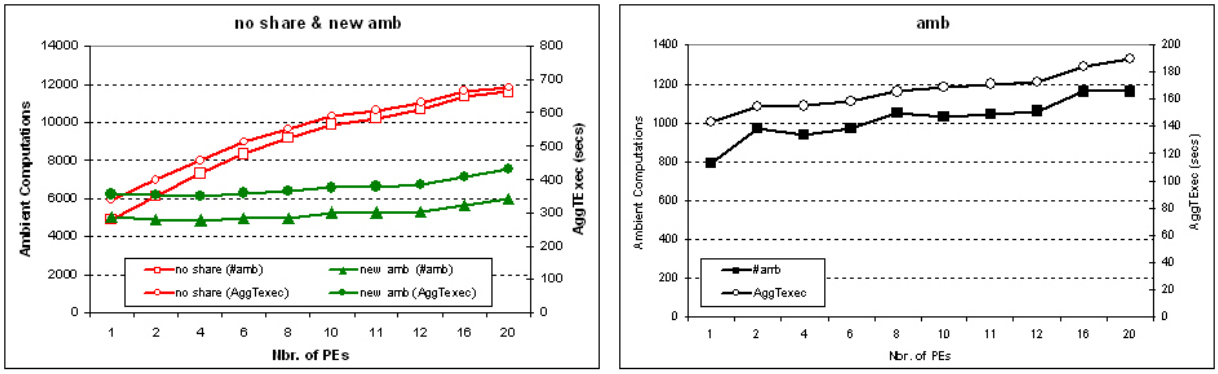


Figure 8: Art Gallery – Number of ambient computations and aggregated execution time

3.4 Highlights

The experiments and results reported suggest the following improvements to `prpict`:

Improve load distribution – $T_{idle} \propto N_{PEs}$, therefore appropriate scheduling strategies are required to increase performance;

Dynamically adapt N_A – to achieve an efficient trade-off between distribution time and ambient values extensive sampling time;

Improve T_{amb}^{Agg} – which is still $\propto N_{PEs}$; this is closely related to the previous item, and should be the subject of further research.

4 Animations and Adaptive Indirect Irradiance

One goal of this research is to render walkthroughs at interactive rates (25 fps), which amounts to render successive frames of the same scene with slightly varying view points. `rpict` (and `prpict`) can render successive frames using options `-o` and `-S` and by providing a list of viewing parameters in the view file.

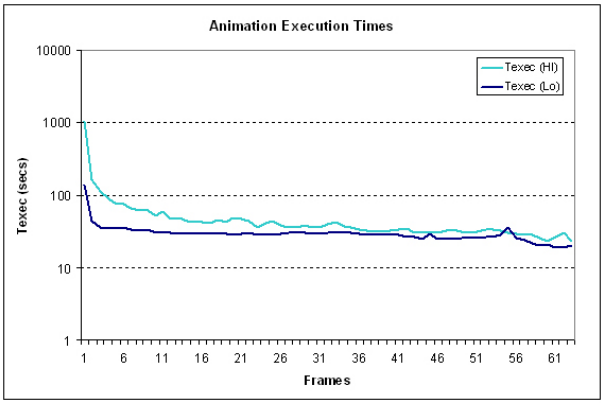
On walkthroughs ambient computations are expected to require a large percentage of the rendering time on the first few frames of the walkthrough and whenever the 3D subregion of the world projected onto the screen changes significantly due to movement of the view point. Since the actual set of extensively computed ambient samples is view–dependent, if the view changes and new objects become visible then new ambient samples have to be calculated.

Figure 10 shows the execution and aggregated ambient times for a walkthrough in the Art Gallery Hall (figure 9)[4]. No major changes in the set of objects projected onto the screen occur during the walkthrough 63 frames. The animation was rendered with both high and low ambient accuracy, `aa`=0.125 and `aa`=0.575, respectively. Independently of the ambient accuracy, the first few frames take an huge amount of time to complete (the time axis have logarithmic scales) compared to the remaining frames due to indirect irradiance computation. Since there are no previously computed ambient values, the number of extensive ambient computations is huge. The same happens if the set of screen projected objects changes significantly.

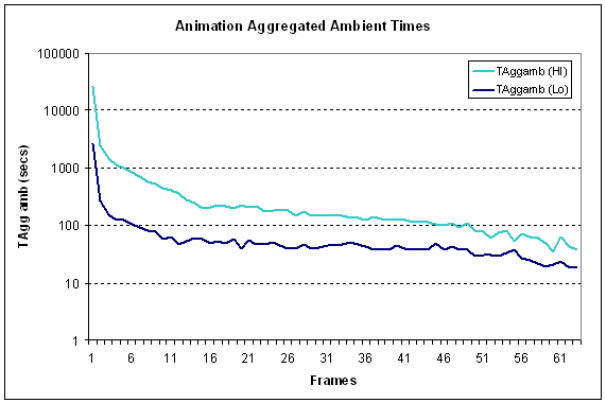
The above described phenomenon prevents the renderer from maintaining a fixed frame rate, since the computational requirements are concentrated on a few frames. Additionally, the computational weight of indirect irradiance calculations constitutes a major obstacle to achieve interactive frame rates.

[4]The animations discussed in this report can be found in `http://www.di.uminho.pt/psantos/Bristol2003`

Figure 9: Art Gallery Hall



(a) Execution Time



(b) Aggregated Ambient Computation Time

Figure 10: Art Gallery Hall Animation Time Measurements

Besides the obvious need to speed up these computations, it may be possible to reduce the frequency with which they are required, i.e., increase the interpolation/sampling ratio while maintaining the same perceptual quality. This ratio can be increased by enlarging the domain of validity of each ambient sample. Radiance's ambient accuracy option, `-aa`, controls the diameter of this domain of validity. This is actually the inverse of accuracy, since the larger its value the larger the tolerance to interpolation errors. By increasing this parameter the domain of validity of each ambient sample is enlarged, resulting in less extensive sampling, but larger errors. Each ambient computation time could also be reduced by varying the number of rays that are spawned to sample the hemisphere. The samples, however, would have varying accuracy, which would prevent their reutilization on posterior frames.

Three complementary criteria are proposed to both distribute ambient calculations across several frames and reduce the time required to perform these computations, while maintaining perceived quality:

time varying accuracy – the computational requirements of ambient calculations can be spread across the first few frames of an animation, by starting with a large tolerance error and progressively reduce it; hopefully, the viewer will not be able to notice the reduced quality of initial frames, since they rapidly converge towards an higher quality solution; the same criterium applies when the set of objects projected onto the image planes varies significantly;

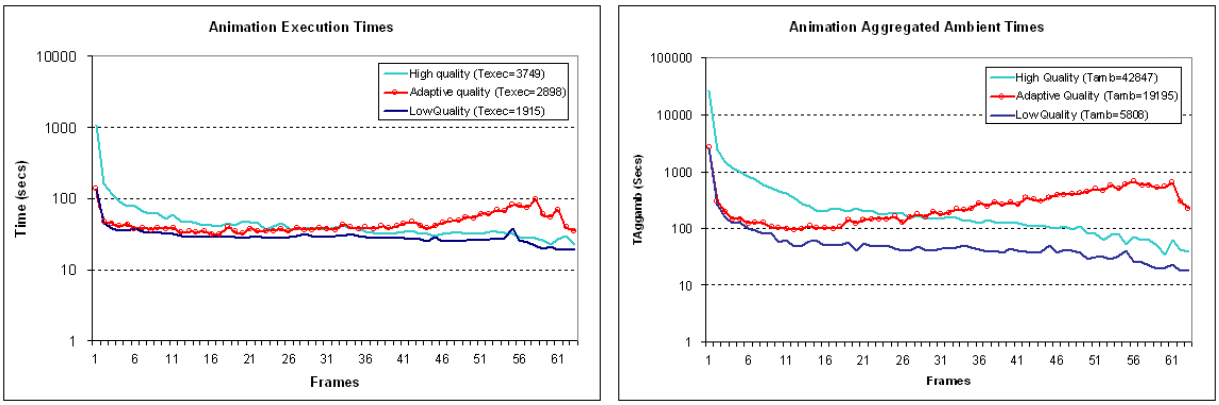
moving direction dependent accuracy – ambient values are cached in order to be reused on the next frames; accuracy should be larger on those regions of the image that will also be seen on the next frames; regions that will not be looked at again can be rendered with less quality; the viewer will not notice this degradation on quality, since these regions will not be displayed long enough;

perceptually guided accuracy – perceptual metrics, such as inattentional blindness [1, 2] and saliency [26], can be used to predict which regions of the image will attract the viewers attention and should, therefore, be rendered with higher quality;

A few experiments were performed to study the first criteria. Work currently being done includes further refinement of this criterium and a more thorough specification and implementation of the others. Assessment of the impact of these techniques on the perceptual quality of the final animation is also required, using both psychophysical experiments [9] and perceptual metrics, such as the Visual Differences Predictor (VDP).

4.1 Time Varying Ambient Accuracy

The `-ax` command line option was added to `prpict` to enable time adaptive ambient accuracy. It takes three arguments: the initial and final error tolerances and a constant step, which is subtracted from the current tolerance at each new frame until the final value is reached. The art gallery animation was rendered with `-ax 0.575 0.125 0.0075`. The initial and final error tolerances are 0.575 and 0.125, the same as the low and high quality renderings of figure 10. The animation has 63 frames, higher quality being reached at frame 61, i.e., 2.4 s after the beginning of visualization at 25 fps. The 3 last frames have the same absolute quality has the high quality animation. Figure 11 presents the execution and aggregate ambient computation times.



(a) Execution Time

(b) Aggregated Ambient Computation Time

Figure 11: Art Gallery Hall Animation Time Measurements with Time Varying Accuracy

The execution times are effectively reduced and more constant: while the high quality animation execution time varies by two orders of magnitude, the adaptive one varies only by one order of magnitude while still converging to the same final solution. However, the initial maximum and the peaks after frame 50 can still prevent the renderer from maintaining a constant frame rate. This criterium alone does not seem able to uniformly distribute indirect irradiance computation costs across a few initial frames.

The ambient accuracy time adaptation is perceivable on the animations: shading and shadows due to indirect irradiance change until they converge to the final solution. Although validation using psy-

chophysics experiments and perceptual metrics is required, observation suggests that the adaptation is too noticeable. This may be due to the time elapsed until final quality level is reached on this particular experiment: 61 frames or 2.4 seconds. This interval can be made shorter by increasing the step in `-ax`; this will certainly increase the initial maximum and smooth the peak visible after frame 50. This last peak suggests that the number of ambient calculations increases non linearly with accuracy. This being true the adaptation step should not be constant: it should be larger for high error tolerances and smaller for high accuracies. Experiments are required to corroborate this hypothesis.

4.2 Moving Direction Dependent Accuracy

4.3 Perceptually Guided Accuracy

4.4 Highlights

Three criteria are proposed to both reduce and spread ambient computation costs across several frames in an animation by dynamically adjusting ambient calculation accuracy. Experiments were performed only for time varying accuracy. Results suggest that although both execution time and frame computation time variance are reduced, this criterium alone can not effectively distribute ambient computation times across several frames. Nevertheless, an ambient accuracy adaptation scheme with a non–constant step should be tried to verify whether frame computation times get more uniform.

The adaptation process is noticeable on the rendered animations. The time interval between initial and final accuracy should be made shorter. Perceptual metrics and psychophysics experiments are required to verify these results.

5 Conclusions

A parallel prototype of Radiance's `rpict` has been developed with the primary aims of getting insight into Radiance's code structure, propose a skeleton for a parallel version and study scalability issues. The ultimate goal of this research is to achieve interactive rendering frame rates, by exploiting parallelism, low-level optimizations and perceptual issues.

The prototype `prpict` is an image space decomposition, message passing, demand driven, totally replicated data parallel application. Although quite good efficiencies are achieved, there is still the need for further improvements. Particularly, the execution time is closely correlated with the time spent computing indirect irradiance values. This time should be minimized to the maximum possible extent, without sacrificing perceived quality. This could be achieved by: reducing the number of such extensive computations, reducing the cost of each computations and effectively sharing these values across space and time. Also processor's idle times due to suboptimal load distribution should be minimized.

Three criteria are proposed to both distribute ambient computation times across several initial frames, rather than just the first one, and to reduce this cost. These criteria are: time varying accuracy, moving direction dependent accuracy and perceptually guided accuracy. Only the time varying ambient accuracy criterium has been implemented and tested. Although it smoothes ambient computation times, they are still far from uniformly distributed and the adaptation process is perceivable on the final animations. Further experiments are required in order to test with smaller adaptation time intervals and with a decreasing accuracy convergence step.

These criteria impact on the final animations and on the viewer's perception of selective and adaptive rendering quality must be assessed using psychophysics experiments and perceptual metrics.

5.1 Future Work

The following issues, closely related to the work presented on this report, must be handled briefly:

1. run experiments with a smaller accuracy adaptation time interval and with a decreasing accuracy convergence step, in order to reduce accuracy adaptation perception;
2. perform psychophysics experiments and use perceptual metrics to quantify the impact of selective rendering on perceived quality;
3. design, implement and evaluate an efficient mechanism to compute and share ambient values, in order to reduce T_{amb} significantly;
4. verify `prpict` robustness and integration with the remaining Radiance tools, particularly `rad` and `ranimate`, and promote its use by the graphics community.

During the course of this work several issues, which can hinder the viability of interactive rendering with Radiance's illumination model, became clear:

- Radiance's code and data structures are not suited for parallel interactive ray tracing; in fact, `rpict` is tailored towards rendering single frames, not animations; an alternative system architecture must be designed, which integrates issues such as frame coherence, perceptual metrics, parallel computing, alternative illumination techniques, asynchronous rendering and visualization, etc.; the "RoD" group at Bristol is working on this;
- perceptual metrics are being studied to predict where the user attention is focused, so that rendering quality is better on those regions; perceptual knowledge can also be used to create artifacts, either visual or audio, to distract the user's attention from very expensive to compute regions; this issue must be further researched;
- a lot of flickering is noticeable in an animation due to the stochastic nature of distributed ray tracing; some solution must be found to reduce this flickering, such as using quasi-random deterministic processes instead of random stochastic ones [5];
- additional artifacts are introduced on the animation by the tone mapping process; since each frame is tone mapped independently, as if the viewer was completely adapted to that frame luminance level, small changes between frames may result in different mappings, causing a sudden change in brightness on the animation; these issue is currently being studied, in order to implement a time dependent tone mapping to reduce such fluctuations in brightness, which are not perceived in real life.

A Parallel Radiance's Code Roadmap

This text presents the code changes done to Radiance 3.5 in order to implement the parallel version of `rpict` (`prpict`) presented and discussed in the previous sections. This text is based on [14].

`prpict` is an image space decomposition, message passing, demand driven, totally replicated data parallel version of `rpict`, as described in chapter 3. It was developed using mpich 1.2.5, although any other distribution of MPI should compile with no problems. To run `prpict` with mpich, using N

processes ($N-1$ processing elements), use the following command:

`mpirun -np` N `<path>/prpict <comand line options>`

MPI might be unable to find Radiance programs on their standard directories. If error messages are displayed, related to the application being unable to find the modelling tools, such as `xform`, `genprism`, etc., create symbolic links to these tools in all subdirectories used by your scene description.

The following files were modified:

prpmain.c
Substitutes rpmain.c Includes MPI, timers and statistics initialization. Calls different routines depending on the process rank. -PP option has been removed. -ax option has been added.
prpict.c
Substitutes rpict.c Contains `rpict()` - function running on the slave nodes. Basically is an adapted version of the sequential `rpict()`, except that the region to be rendered is received as a message from the master.
ambientCTR.c
Substitutes ambient.c Includes timing and statistics. Includes `checkambient()` - create a new ambient file. If running on the master process (rank 0) accesses the ambient file and collects/distributes ambient values from/to all PEs

Table 6: prpict - Changed source code files

master.c
Added - runs on the master node (rank 0) Distributes work the the slaves on demand. Partitions the image on several regions (tasks) Collects the results and saves them onto `stdout` Receives AMBSYNC messages from PEs in order to synchronise the indirect irradiance (ambient) cache

Table 7: prpict - Added source code files

References

[1] K. Cater, A. Chalmers, and P. Ledda. Selective Quality Rendering by Exploiting Human Inattentional Blindness: Looking but not Seeing. In *VRST'02*, Hong–Kong, China, November 2002. ACM Press.

[2] K. Cater, A. Chalmers, and G. Ward. Detail to Attention: Exploiting Visual Tasks for Selective Rendering. In *Eurographics Symposium on Rendering*, Leuven, Belgium, June 2003. The Eurographics Association.

[3] A. Chalmers, S. Daly, A. McNamara, K. Myszkowski, and T. Troscianko. Image Quality Metrics. SIGGRAPH'2000 – Course 44, July 2000.

[4] R. Cook, T. Porter, and L. Carpenter. Distributed Ray Tracing. In *SIGGRAPH*, pages 137–145, January 1984.

[5] K. Dmitriev, S. Brabec, K. Myszkowski, and H. Seidel. Interactive Global Illumination using Selective Photon Tracing. In Debevec and Gibson, editors, *Eurographics Workshop on Rendering*, pages 25–36, Pisa, Italy, June 2002. ACM Press.

[6] D. Greenberg. A Framework for Realistic Image Synthesis. *Communications of the ACM*, 42(8):44–53, August 1999.

[7] J. Kajiya. The Rendering Equation. In *ACM SIGGRAPH*, volume 20, pages 143–150, Dallas, USA, August 1986.

[8] R. Koholka, H. Meyer, and A. Goller. MPI-parallelized Radiance on SGI CoW and SMP. In *Parallel Computation, 4th International ACPC Conference*, number 1557 in LNCS, pages 549–558, Salzburg, Austria, 1999. Springer-Verlag.

[9] R. Kosara, C. Healey, V. Interrante, D. Laidlaw, and C. Ware. User Studies: Why, How and When? *IEEE Computer Graphics and Applications*, pages 20–25, July 2003.

[10] E. Reinhard, A. Chalmers, and F. Jansen. Sampling Diffuse Inter Reflection within a Hybrid Scheduling Ray Tracer. *Journal of Parallel and Distributed Computing Practices*, September 2000. Special Issue on Parallel and Distributed Computer Graphics.

[11] D. Robertson, K. Campbell, S. Lau, and T. Ligocki. Parallelization of Radiance for Real Time Interactive Lighting Visualization Walkthroughs. In *ACM/IEEE Conference on Supercomputing*, Portland, OR, USA, 1999. ACM Press.

[12] L. Santos and A. Proenca. A Bayesian RunTime Load Manager on a Shared Cluster. In *1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'2001)*, pages 674–679, Brisbane, Australia, May 2001. IEEE Computer Society.

[13] L. Santos and A. Proenca. A Systematic Approach to Effective Scheduling in Distributed Systems. In *VECPAR'2002 – 5th Int. Meeeting on High Performance Computing for Computational Science*, pages 813–825, Porto, Portugal, June 2002.

[14] Luis Paulo Santos. *Radiance 3.5 - Source Code & Data Structures Road Map*. Dep. de Informatica - Universidade do Minho, Braga, Portugal, May 2003.

[15] M. Simmons and C. Sequin. Tapestry: A Dynamic Mesh-based Display Representation for Interactive Rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 329–340. Springer-Verlag, 2000.

[16] M. Stamminger, J. Haber, H Schirmacher, and H. Seidel. Walkthroughs with Corrective Texturing. In *Proceeding of the Eurographics Workshop on Rendering Techniques*, 2000.

[17] P. Tole, F. Pellacini, B. Walter, and D. Greenberg. Interactive Global Illumination in Dynamic Scenes. *ACM Transactions on Graphics*, 21(3):537–546, 2002.

[18] B. Walter, G. Drettakis, and D. Greenberg. Enhancing and Optimizing the Render Cache. In *Proceeding of the Eurographics Workshop on Rendering Techniques*, pages 37–42, 2002.

[19] B. Walter, G. Drettakis, and P. Parker. Interactive Rendering Using the Render Cache. In *Proceeding of the Eurographics Workshop on Rendering Techniques*, pages 235–246, 1999.

[20] G. Ward and M. Simmons. The Holodeck Ray Cache: an Interactive Rendering System for Global Illumination in Nondiffuse Environments. *ACM Transactions on Graphics*, 18(4):361–398, October 1999.

[21] Gregory Ward. The RADIANCE Lighting Simulation and Rendering System. In *SIGGRAPH'94 - 21st International Conference on Computer Graphics and Interactive Techniques*, pages 459–472. ACM Press, 1994.

[22] Gregory Ward and Paul Heckbert. Irradiance Gradients. In *3rd Annual Eurographics Workshop on Rendering*, Bristol, UK, 1992.

[23] Gregory Ward and F. Rubinstein. A ray tracing solution for diffuse interreflection. *Computer Graphics*, 22(4), 1988.

[24] Gregory Ward and Rob Shakespeare. *Rendering with Radiance: the art and science of lighting visualization*. Morgan Kaufmann, 1998.

[25] T. Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–349, June 1980.

[26] H. Yee, S. Pattanaik, and D. Greenberg. Spatiotemporal Sensitivity and Visual Attention for Efficient Rendering of Dynamic Environments. *ACM TRansactions on Graphics*, 20(1), January 2001.