High-fidelity Interactive Rendering on Desktop Grids

Vibhor Aggarwal, Kurt Debattista, Thomas Bashford-Rogers, Piotr Dubla and Alan Chalmers

The Digital Lab, University of Warwick, UK

Abstract

High-fidelity interactive rendering has been traditionally restricted, to expensive shared memory or dedicated distributed processors, due to the high computational cost. A desktop grid offers a low cost alternative by combining arbitrary computational resources connected to a network such as those in a laboratory or an office. However, prevalent interactive rendering algorithms are currently incapable of seamlessly handling the variable computational power offered by the non-dedicated resources of a desktop grid. In this article, we present a novel fault-tolerant algorithm for rendering high-fidelity images at an interactive rate which is capable of handling variable resources. A conventional approach of rescheduling failed jobs in a volatile environment would inhibit performance while rendering at interactive rates as the time margins are small. Instead, our method uses quasi-random sampling along with image reconstruction techniques to deal with faults. This enables users to experience interactive high-fidelity rendering on their desktop machines.

Keywords: interactive rendering, desktop grids, fault-tolerance, image reconstruction, high-fidelity rendering, computer graphics, parallel computing

1. Introduction

High-fidelity rendering at interactive rates offers the potential for high quality imagery to be used as a regular tool in many visualisation applications including special effects, product appearance, building design and virtual archaeology. This is essential for providing a good feedback to the user for steering any visualisation towards a better solution. In the context of rendering computer generated imagery, especially for areas such as product visualisation, film industry or an architectural walk-through, it would help an animator to adjust lighting conditions and object properties in the scene for the best visual appeal. However, such interactive renderings require large amount of computational power and thus are traditionally rendered using dedicated parallel resources, often referred to as render farms. These dedicated render farms are expensive, limiting the number of users who have access to high-fidelity interactive rendering. In addition, only a few implementations for high-fidelity interactive rendering exist even on these dedicated machines, for example Benthin et al. [BWS03] and Wald et al. [WKB*02].

In this article, we present a novel method for achieving interactive high-fidelity rendering on inexpensive non-dedicated machines such as desktop grids. The challenge of using a non-dedicated platform is that it provides volatile computational power with shared heterogeneous resources and hence, fault-tolerant algorithms are required to achieve the desired results in a timely manner. Our method effectively enables the majority of users that work in small to medium sized companies or universities to experience interactive high-fidelity rendering on their desktop machines when other user's resources are left idling, without the expensive requirements of a dedicated render farm.

With the advent of multi-core desktop machines in recent years, desktop grids have become an attractive option for low cost distributed computing. Some of the cores of a multi-core machine are not always in constant use. Therefore, having such machines readily available on a desktop grid will enable organisations to maximise their usage. In an office or a lab these idle CPUs can be a cheap source of computational power. A desktop grid with a usually stable interconnect between the resources is commonly referred to as a *local* desktop grid. In this article, we use the term desktop grid to refer to such kind of a desktop grid unless specified otherwise. For more information on desktop grids please see Section 2 (Box).

On a volatile computing platform one of the two primary fault-tolerance mechanisms, redundancy or checkpointing-and-restarting (or a hybrid of the two) is typically em-

ployed to deal with faults. Interactive rendering imposes severe time-constraints on the system and these fault-tolerance mechanisms are not well suited under such constraints as they inhibit performance. Instead robust image space sampling techniques such as quasi-random sampling along with image reconstruction methods can be used to deal with faults. If a job fails, image reconstruction algorithms can be used to fill in the missing data.

The idea of combining sparse sampling with reconstruction has been used before (see for example in the render cache [WDP99]) for ray tracing at interactive frame rates. Our approach extends this idea by adapting sparse sampling as a mechanism to enable fault-tolerance. When exploited in the context of interactive rendering of high-fidelity images, it allows the user to change the scene at run-time and receive feedback without employing expensive dedicated resources. This is interesting for any application which can benefit from interactive hypothesis testing and in which creativity may be stifled by the need to wait for significant periods of time before seeing any rendering results from a small change to a model. To the best of our knowledge, our method is the first desktop grid method to handle interactive rates and the first within the context of high-fidelity rendering.

Our system would be useful for an architect or an archaeologist to visualise a building under different lighting conditions. For a product designer it could provide a preview tool for testing different object materials and an animator can use it to have a quick look at characters in a fully rendered environment before submitting the final frames to a render farm. A user may not be able to try out all these different settings in an off-line rendering environment as it is a time consuming process and could use our system instead without any additional expense by just connecting all the available machines into a desktop grid.

---

2. Desktop Grids (Box)

2.1. Desktop Grids

A computational grid can be defined as a distributively-owned multiprogrammed large pool of heterogeneous computing resources interconnected by telecommunication channels. It is a multi-user, massively parallel shared resource environment with interconnected clusters, databases and equipment spanning administrative and geographic boundaries. The grid is viewed by Foster as a dependable, consistent, pervasive and inexpensive access to high-end computational resources [FK99].

A desktop grid is an example of a computational grid which combines computational power from vast number of desktop machines. Large computational power at a low cost makes desktop grids an attractive option for many applications. As an example, Berkeley Open Infrastructure for Network Computing (BOINC) projects which run on the world's largest desktop grids, utilise an average of 3.2 PetaFLOPS from about 500,000 active hosts worldwide [BOI10]. This is comparable to the 1.7 PetaFLOPS currently provided by the world's fastest supercomputer [TOP10].

A desktop grid is sometimes termed as volunteer computing to reflect that the resource owners donate idle CPU cycles without any guarantee of service. The idea of using "cycle stealing" from desktop machines originated with the PARC worm [SH82] which scanned a list of resource addresses and replicated itself on the idle hosts. Since then it has been widely used for solving complex scientific problems, the most well known being the SETI@Home project [ACK*02]. SETI@Home analyses radio signals to search for extra-terrestrial intelligence using idle CPU cycles donated by volunteers all around the world.

The computational power provided by a desktop grid is volatile because of shared resources, machine hardware failure and network congestion or failures. This poses a significant challenge when computing on a desktop grid and the algorithms for utilising it need to cater for any failures. Applications that can be subdivided into independent jobs, with high computation to communication ratio such that the number of jobs is much more than the available resources, are potentially suitable for a desktop grid [CBS*03]. A wide array of scientific applications from various fields such as astronomy [ACK*02], climate modelling [CAS05] and computational biology [LSS*03] have been successful in harnessing the power of desktop grids.

A local desktop grid contained within an institution is beneficial for running applications which need greater control. Such a desktop grid offers more reliable and faster connectivity of resources through a local area network. Application deployment and development is easier on them because the resources do not span many administrative boundaries and geographic locations and consist of a limited number of software platforms [Kon05]. A local desktop grid is suitable for scheduling parallel applications requiring a rapid turnaround.

References

[ACK*02] ANDERSON D. P., COBB J., KORPELA E., LEBOFSKY M., WERTHIMER D.: Seti@home: an experiment in public-resource computing. *Communications of the ACM 45*, 11 (2002), 56–61.

[BOI10] BOINC Project Statistics. http://boincstats.com/stats/project_graph.php?pr=bo , January 2010.

[CAS05] CHRISTENSEN C., AINA T., STAINFORTH D.: The challenge of volunteer computing with lengthy climate model simulations. In *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 8–15.

[CBS*03] CIRNE W., BRASILEIRO F., SAUVE J., ANDRADE N., PARANHOS D., SANTOS-NETO E., MEDEIROS R.: Grid computing for bag of tasks applications. In *Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment* (2003).

[FK99] FOSTER I., KESSELMAN C. (Eds.): *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 1999.

[Kon05] KONDO D.: *Scheduling Task Parallel Applications For Rapid Turnaround on Desktop Grids*. PhD Thesis, University of California, San Diego, 2005.

[LSS*03] LARSON S. M., SNOW C. D., SHIRTS M., P V. S., PANDE V. S.: Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. In *Computational Genomics*, Grant R., (Ed.). Horizon Press, 2003.

[SH82] SHOCH J. F., HUPP J. A.: The "worm" programs—early experience with a distributed computation. *Communications of the ACM 25*, 3 (1982), 172–180.

[TOP10] TOP500 Project. http://www.top500.org/lists/2009/11 , January 2010.

Desktop Grids Box

3. High-fidelity Rendering

High-fidelity rendering is the process of synthesising images by computing the light transport in a scene using physically-based quantities. This calculation is performed by solving the rendering equation [Kaj86]. The outgoing radiance at a point p in direction Θ is given by:

$$L(p \rightarrow \Theta) = L_e(p \rightarrow \Theta) + \int_{\Omega_p} f_r(p, \Theta \leftrightarrow \Psi) \cos(N_p, \Psi) L_i(p \leftarrow \Psi) \delta_{\omega_\Psi}$$

where L_e is the emitted radiance, L_i is the incoming radiance in the direction Ψ and f_r represents the Bidirectional Reflectance Distribution Function (BRDF). There are two broad categories of solving this equation: finite-element methods and point-sampling solutions. Our method for interactive rendering on desktop grids is suitable for the point-sampling solutions.

Monte-carlo integration is frequently used to estimate the rendering equation. Many samples are required to reduce the Monte-carlo noise, making the process computationally expensive. This takes a long time to complete on a single machine for even moderately complex scenes. Parallel computing offers the potential to obtain results in a reasonable time. Many parallel rendering solutions [CDR02] have been proposed but most of them are focused towards expensive closely-coupled machines. The prevalent parallel rendering techniques are not suited for computing on non-dedicated massively parallel systems which provide a cheaper alternative but with variable resources. The heterogeneity of resources raises issues of load balancing and cross platform computing. In addition, fault-tolerant rendering algorithms are needed to cope with volatility of resources.

Previous algorithms [ACD08, CSL06] for off-line rendering on a computational grid relied on the fault-tolerance provided by the grid middleware to tackle any faults. A time-critical visualisation method for grid computing presented by Gao et al. [GLH*08] used redundancy for fault-tolerance. A time-constrained algorithm for rendering on a desktop grid was presented by Aggarwal et al. [ADD*09] which used quasi-random sampling with reconstruction to cope with faults. But this could only handle static images due to its dependence on the grid middleware for job management and could not achieve interactivity. The method presented here describes a novel system that enables interactive rates, which would not have been possible by relying on traditional grid middlewares for job management and fault-tolerance.

4. Interactive Rendering on Desktop Grids

In this section, we discuss the design and functioning of our method. It follows a master-worker paradigm using a job pull mechanism [CDR02]. In this approach, workers ask the master for work, process the input and then send the results back to the master for composition. The workers do not communicate with each other. We schedule these workers on the available resources of a desktop grid and they connect to the master to fetch a rendering job. Workers can connect and disconnect at their own will and the master has no control over this. This enables the method to employ variable resources for parallel rendering. Each frame which needs to be rendered is divided into quasi-random sets of pixels and each set is queued up as a different job by the master. The number of jobs per frame is much greater than the maximum number of available resources for load balancing. If some of these jobs are not completed, reconstruction is used for completing the frame.

The overview of our method is presented in Figure 1. First, the user interacts with the scene which defines the rendering parameters for the frame. This frame is then divided into sets of quasi-randomly chosen pixels which are queued up as jobs. When an idle worker connects to our system, a job from the queue is sent to it to process. Once the results are received, the partial frame is sent to the Graphics Processing Unit (GPU) for reconstruction and display. Figure 1 also illustrates the four possible states of any machine in the desktop grid: receiving a job, rendering, sending results and unavailable for computation. The following subsections provide the details about communication, scheduling, display and reconstruction, and workers used in our method.

4.1. Communication

The master uses one thread per worker architecture for communicating with them for maximum throughput. We have designed these threads such that they do not need to synchronise with each other. Also, they use non-blocking data structures while synchronising with the scheduler and display and reconstruction threads allowing the master to scale efficiently with the increase in number of workers.

The master sends out a job packet from the front of the job queue and then waits for the worker to process the job. When the worker sends the results back, these are stored for later compositing. The job packet consists of the following:

Frame Number identifies the frame to which this job belongs.

Set Number is used by the worker to identify which set of quasi-random pixels it needs to render.

Scene State includes all the information which the worker

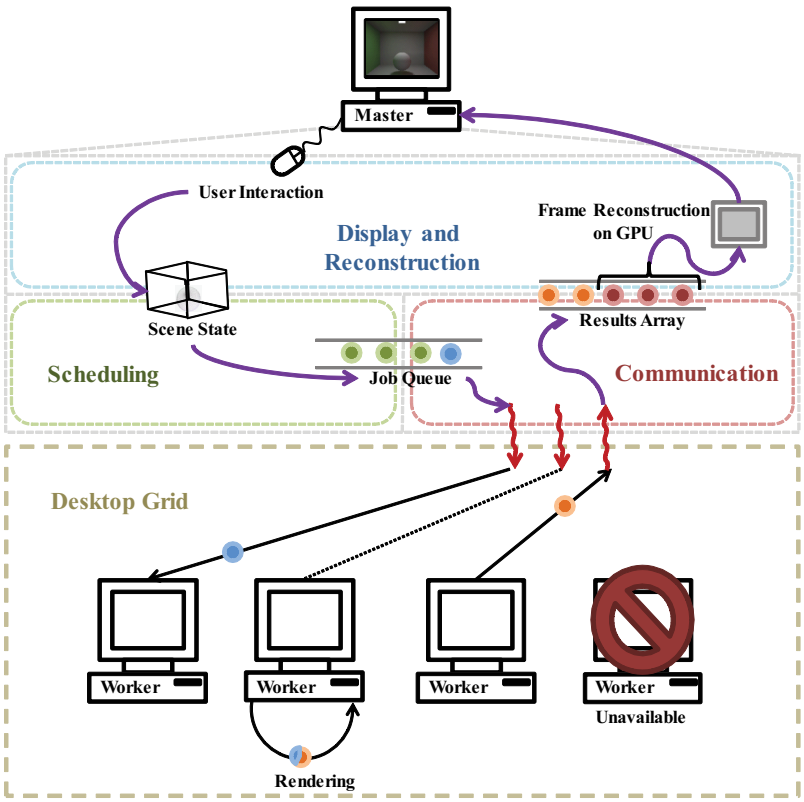


Figure 1: The overview of our method showing the interactions inside the master and between the master and the desktop grid.

needs to render the correct view of the scene such as camera position, object materials, object positions etc.

The job packet contains the absolute scene state rather than having an update scene message, because a worker may or may not receive all the update messages depending on which frames it processes. This helps in synchronising the scene state on all the workers processing a given frame. The result packet contains the luminance values of the computed pixels along with the frame number and the set number according to which job was processed by the worker. As multiple frames may be scheduled together, the master can receive results for multiple frames at the same time. Therefore, an array of results is used, and the frame and set numbers correctly identify the results. It is possible that a worker takes too long a time to send back the results for a frame which, by then has been displayed, in such a case these results are discarded and a new job is sent to the worker.

4.2. Scheduling

The computing power of a desktop grid is time-variant and hence, it needs to be closely observed to prevent either an excess or lack of jobs in the queue. The job scheduler monitors the number of jobs in the job queue. If this number falls below the job queue threshold, it adds jobs for the current frame, using the latest scene state, to the job queue. The scheduler works asynchronously to match the rate at which jobs are added to the rate at which they are being processed by the workers in a desktop grid to avoid under- or overflow of the queue.

Traditionally, an image is subdivided into tiles for parallel rendering such that each tile can be calculated in parallel [CDR02]. The image is composited when all the tiles are rendered. For desktop grids, redundancy could be employed to ensure all the tiles are computed. This would entail that if one of the tiles fails to come back (within a specified time) due to a delay or fault, a duplicate copy of the same job would be computed. However, redundancy is not ideal when rendering towards a deadline as it can add substantially to the rendering time needed for a frame, thus hindering performance. Instead, in our approach, an image is subdivided into group of pixels chosen quasi-randomly [KK02] over the complete image space instead of using tiles (see Figure 2). This enables a fair coverage of the whole image per job. If

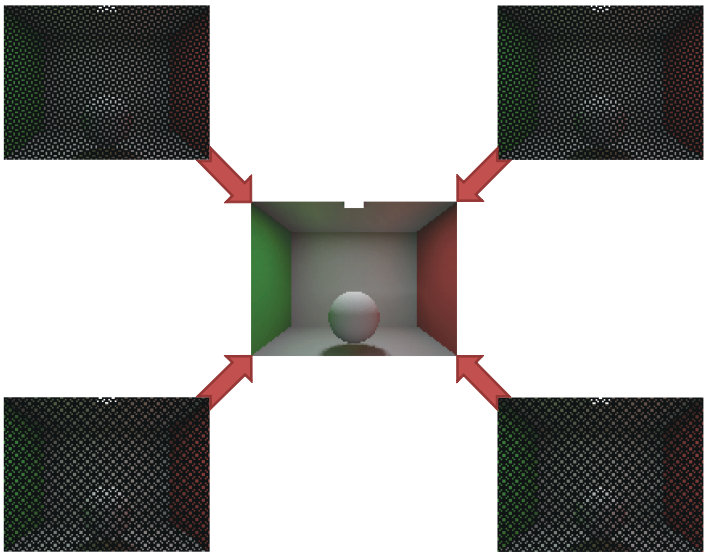


Figure 2: An image of the Cornell Box divided into 4 groups of quasi-randomly chosen pixels. Each of these groups can be independently computed in parallel. In practice, the image is divided into many more groups. Please note that the resolution of 256×192 used in these images is for illustrative purposes only.

(a) Quasi-random Sampling

(b) Tile-based Sampling

Figure 3: Image subdivision techniques and the resultant image in case of faults.

a job fails to complete, image reconstruction techniques are used to fill in the missing data, Figure 3a. It is difficult to reconstruct an image when a tile of pixels is grouped together as a job, since there would be significant holes in the case where task fails without redundancy as shown in Figure 3b. The advantage of using a quasi-random sequence over a purely random sequence is that it provides low discrepancy (fills the space more uniformly). A regular sampling pattern would, on the other hand lead to undesirable structured noise in comparison to quasi-random sampling in case of faults. By using image reconstruction algorithms, redundancy can be avoided for parallel rendering while maintaining time-constraints in a shared environment.

4.3. Display and Reconstruction

Our method is fully dynamic such that it allows the user to change camera view, object positions, object materials, lighting positions and lighting conditions while interacting with the scene. This interaction sets the scene state which the scheduler uses for queuing up the jobs. Before displaying a frame, the master needs to wait for the results to come back from the workers. It then reconstructs the partial frame on the GPU as explained further in Section 4.3.1 and displays it on the user's screen. It is possible that the job queue contains jobs for a frame which has been already displayed and in such a case those jobs are discarded from the job queue.

Our method uses a heuristic to guide the master to determine how long to wait for the results from the worker before reconstructing the partial frame. There are two parameters which form the heuristic, a quality constraint and an interaction constraint.

The quality constraint Q_t, at a time instant t measured from the start of computation for the current frame, is given by:

$$Q_t = \frac{\text{number of jobs finished for the current frame}}{\text{number of total jobs for the current frame}}$$

The quality constraint controls the visual quality of the reconstructed frame. The interaction constraint I_t, at a time instant t is given by:

$$I_t = \frac{\text{time taken for the current frame}}{\text{time taken for the last frame}}$$

The interaction constraint estimates the wait time using the previous frame and tries to maintain smooth user interaction even with variable computing resources. The master waits for time t while one the following conditions are true:

$$Q_t < Q_{max} \quad \wedge \quad I_t < I_{min}$$
$$Q_t < Q_{min} \quad \wedge \quad I_t < I_{max}$$

where Q_{min}, Q_{max}, I_{min} and I_{max} are user-defined minimum and maximum values for the quality and interaction constraints respectively. In the worst case scenario, if the master does not receive results to meet the minimum quality constraint in the time specified by maximum interaction constraint, it skips the current frame and moves on to the next frame. This ensures that even if most of the workers computing the same frame go down, the master would be able to adjust to this significant transient variation in the compute power.

If the user requires a fixed frame rate, the master waits for the specified amount of time before moving onto reconstruction. In this case, the reconstruction tackles the variability of resources and the visual quality of the frames changes.

4.3.1. Image Reconstruction

While rendering using volatile resources, we are not guaranteed to get all the results back due to faults. Therefore, we use image reconstruction techniques to fill in the missing pixels in a partial frame. We use a discontinuity function, $D_{i,j}$ which allows us to find if two pixels i and j are similar for interpolating across them. The discontinuity function is given by:

$$D_{i,j} = \max(0, \vec{n_i} \cdot \vec{n_j} - \alpha) \times \max(0, |z_i - z_j|^2 - \beta^2)$$

where,

$\vec{n_i}$	=	Orientation at pixel i
α	=	User-defined orientation threshold
z_i	=	Position at pixel i
β	=	User-defined position threshold

This discontinuity function can be used as a spatial discontinuity filter when i and j are different in spatial positions, while it can also be used as a temporal discontinuity filter when i and j represent same pixel in different frames.

Firstly, a nearest neighbour reconstruction technique is used to estimate the missing pixels. We chose this technique over other reconstruction methods due to its simplicity and the ability to run in real-time on a GPU. The luminance L_i, of a missing pixel i is calculated as:

$$L_i = \frac{\sum_{j \in \mathbb{N}\{i\}} L_j \times D_{i,j} \times k_{i,j}}{\sum_{j \in \mathbb{N}\{i\}} D_{i,j} \times k_{i,j}} \quad \forall i \in \Omega$$

where,

L_i	=	Luminance at pixel i
$\mathbb{N}\{i\}$	=	Set of calculated pixels in the neighbourhood of i
$D_{i,j}$	=	Spatial discontinuity function
$k_{i,j}$	=	User-defined weighting function
Ω	=	Set of all missing pixels

After nearest neighbour reconstruction, luminance values are accumulated and displayed if the scene state is unchanged from the previous frame. If this is not the case, then we apply a temporal filter to reduce flickering when user interaction occurs. The temporal discontinuity function removes the ghosting artefacts usually generated by a temporal filter and the luminance is updated as follows:

$$L_i = \frac{\sum_{j \in \mathbb{T}\{i\}} L_j \times D_{i,j} \times w_{i,j}}{\sum_{j \in \mathbb{T}\{i\}} D_{i,j} \times w_{i,j}} \quad \forall i \in \Pi$$

where,

$\mathbb{T}\{i\}$	=	Set containing previous values of pixel i
$D_{i,j}$	=	Temporal discontinuity function
$w_{i,j}$	=	User-defined weighting function
Π	=	Set of all image pixels

Finally, the spatial noise is reduced by applying a Gaussian filter with spatial discontinuities while the user is interacting with the scene. This filter is given by:

$$L_i = \frac{\sum_{j \in \mathbb{M}\{i\}} L_j \times D_{i,j} \times g_{i,j}}{\sum_{j \in \mathbb{M}\{i\}} D_{i,j} \times g_{i,j}} \quad \forall i \in \Pi$$

where,

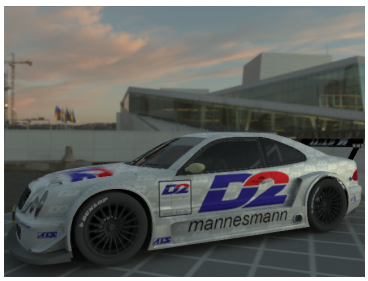
$$g_{i,j} = \frac{e^{-\frac{i^2+j^2}{2\sigma^2}}}{2\pi\sigma^2}$$

σ	=	Standard deviation of the Gaussian function
$\mathbb{M}\{i\}$	=	Set of pixels in the neighbourhood of i

(a) Cornell Box (63k)

(b) Kalabsha (861k)

(c) Kiti (243k)



(d) Race Car (69k)

(e) Sponza (66k)

Figure 4: The scenes used for evaluating our method. The number in brackets indicates the polygon count of the model.

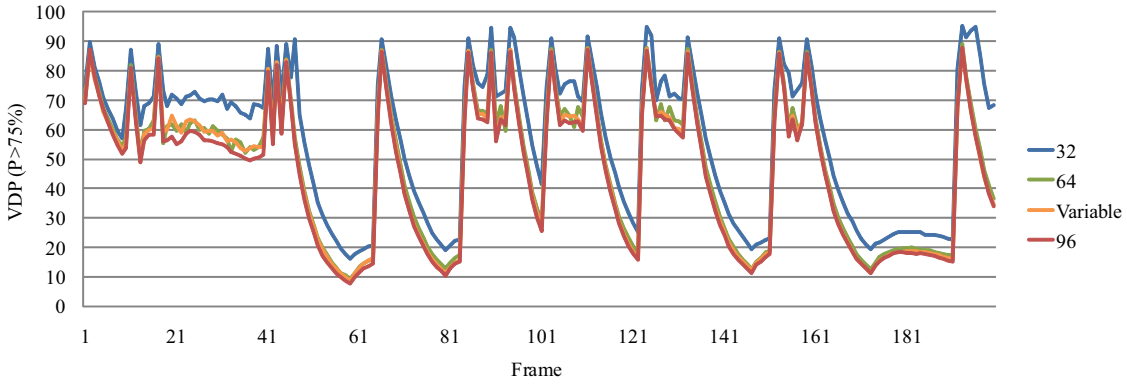
This reconstruction and filtering is only done on the luminance values calculated by the renderer, which does not contain the information about the albedo (surface colour of the material at the primary ray intersection that does not take into account any lighting). We calculate the albedo on the GPU and multiply it afterwards. This prevents filtering across colour boundaries contained in the albedo such as high frequency texture details.

4.4. The Worker

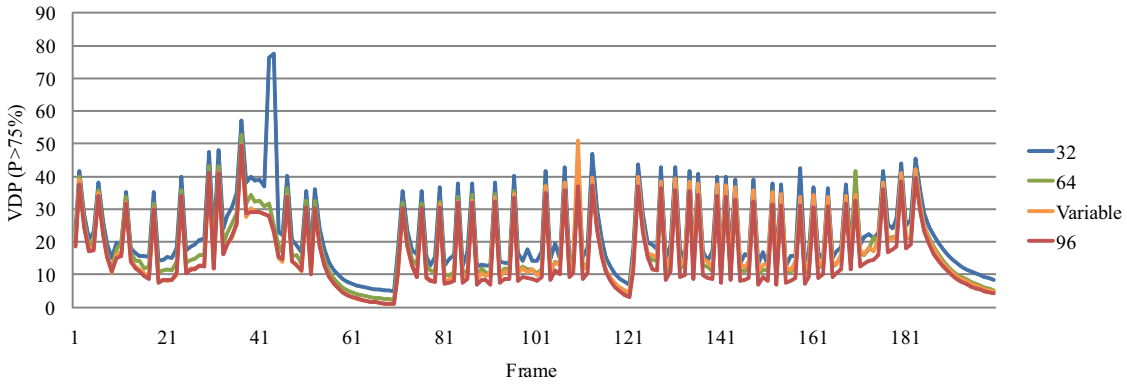
The worker is scheduled on any machine that becomes available on the desktop grid at any time. It is implemented as a single thread such that any core on a machine can be employed, if it lies idle. The worker connects to the master asking for work and receives the job packet for a description of the rendering job. It changes the scene state according to the description provided in the job packet and then renders the specified set of quasi-random pixels. It calculates the luminance of those pixels and sends them back to the master in a result packet and waits until the master sends more work in the next job packet.

5. Implementation Details

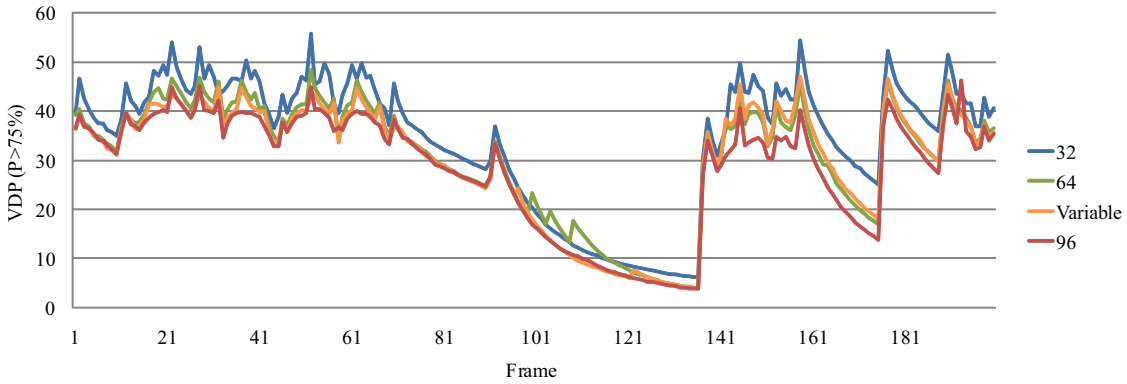
We have implemented the method described in the previous section and tested it on a desktop grid. We used the TCP/IP protocol with portable data serialisation to communicate between workers running on heterogeneous machines and the master. We employed Condor [LLM88] for managing workers on vacant resources and transferring the initial scene files and executables to start computation on them. Our test platform was a desktop grid consisting of two kinds of processors connected by a 100 Mbps Ethernet LAN. There were 48 Dual Core 2.6 GHz AMD Opteron processors with 4GB RAM each running Linux and 8 Quad Core 3.0 GHz Intel Extreme processors with 4GB RAM running Windows. We treated each of the 128 cores as a separate resource to obtain finer granularity in our experiments. The master ran on a machine having 2 Dual Core 2.6 GHz AMD Opteron processors with a shared memory of 8GB running Linux. The machine also had an NVidia 8800 GTX GPU for reconstructing the partial frames in real-time using the OpenGL shading language. The reconstruction was done on the GPU for optimum performance. For calculating the luminance values of the pixels on the workers we used the path tracing algorithm [Kaj86], however, other point sampling methods could also be employed. Each job consisted of rendering a set of pixels chosen quasi-randomly using Sobol and van der Corput sequences [KK02] from a 640×480 frame with four samples computed per pixel per update. The choice of four samples per pixel was made to achieve interactivity on our test bed. Since it has a direct impact on the amount of computation required, much more computational power would be needed to compute higher number of samples per update and maintain interactive frame rates. As desktop machines gradually improve, future iterations of this system would use more samples per pixel. Furthermore, to mask the effect of such low sampling, the filtering was applied as described in Section 4.3.1. Once the user stopped at a particular scene state, such filtering was removed and samples were accumu-



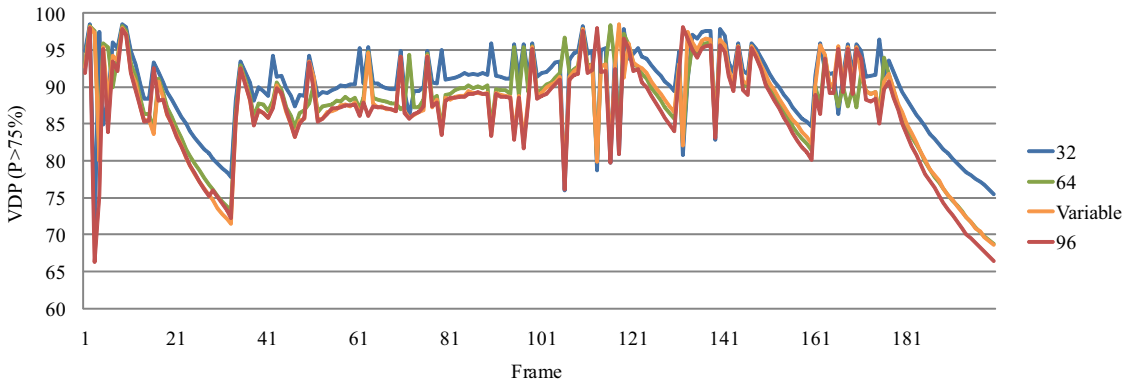
(a) Cornell Box



(b) Kiti

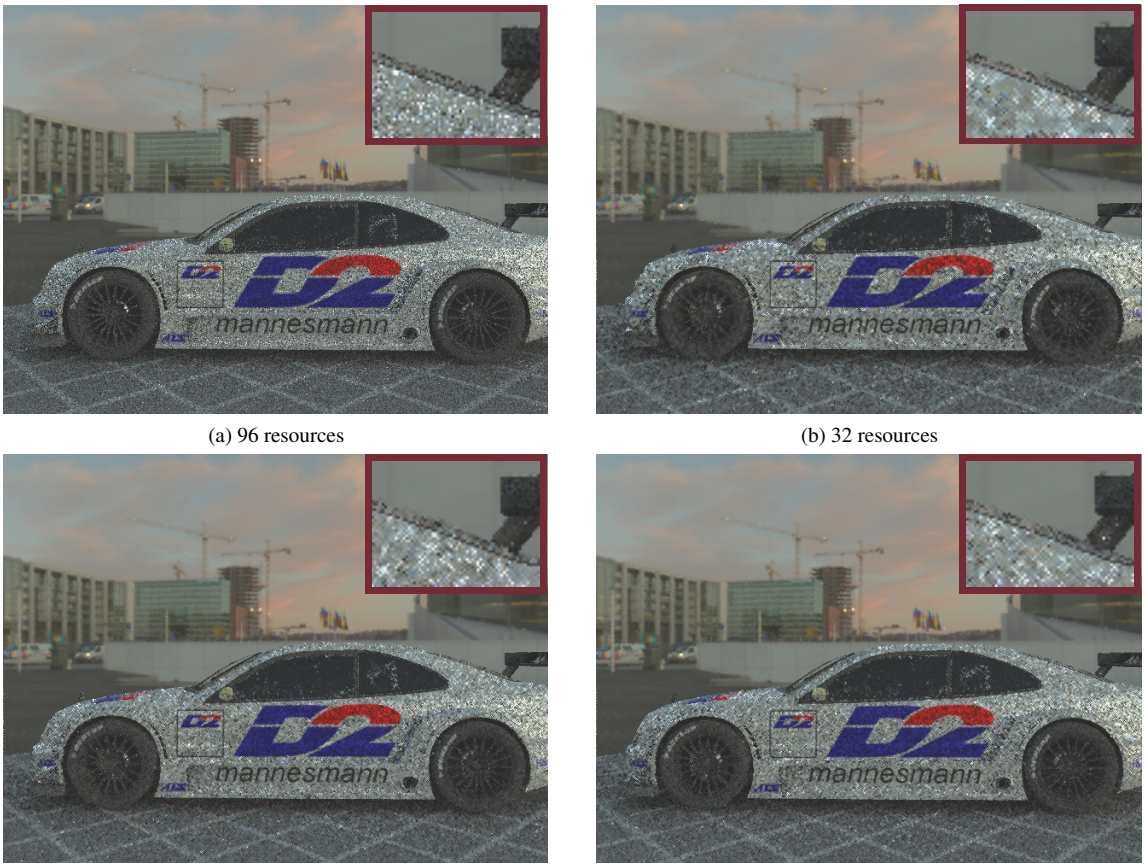


(c) Race Car



(d) Sponza

Figure 5: The graphs depicting the variation in visual quality of an interactive sequence with different resources for four scenes.



(a) 96 resources (b) 32 resources

(c) 64 resources (d) Variable resources

Figure 6: The frame 177 from the Race Car interactive sequence rendered on different number of resources. The inset is an enlarged portion of the image to show the reconstruction artefacts.

lated to offer a refined view. The method allowed the user to interactively change an object's position, material or colour along with a change of light position or direction or change the environment map in a scene while viewing it from different positions.

The master used $Q_{min} = 20\%$, $Q_{max} = 50\%$, $I_{min} = 2$ and $I_{max} = 4$ for estimating the wait time using the heuristics described in Section 4.3. Our job queue threshold was twice the number of jobs per frame. We chose a radius of 5 pixels and $k_{i,j} = 1$ to give equal weights for the nearest neighbour reconstruction step. For the temporal filter we used data from the four previous frames. A weighting function $w_{i,j}$, provided a lesser contribution from older pixel values:

$$w_{i,j} = \frac{1}{F_i - F_j}$$

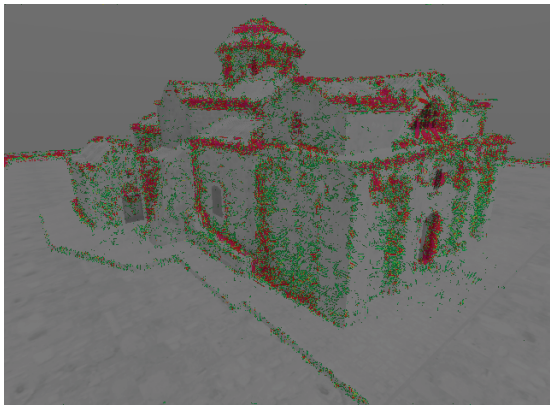
where, F_i represents the current frame number and F_j represents the older frames. The Gaussian filter kernel was seven pixels wide with $\sigma = 1$ while $\alpha = 0.75$ and $\beta = 1$ was used for the discontinuity function. These values were chosen so that the reconstruction could be performed in real-time on the GPU.

6. Evaluation

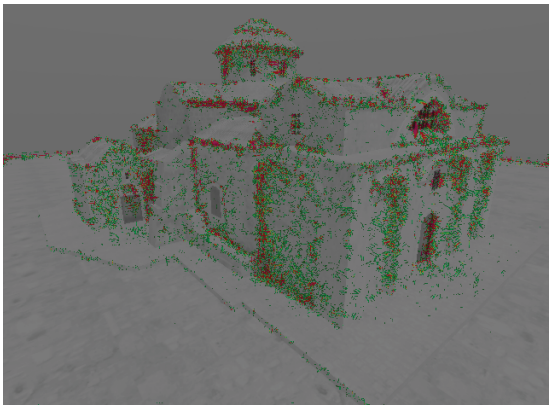
Using the desktop grid described in Section 5, we were able to achieve interactive rates for a variety of scenes. Even for a complex model such as Kalabsha (see Figure 4b) with 861k polygons, large indirectly lit areas and two light sources, the sky and a directional light, we achieved about 3 frames per second on our test platform. For a simpler scene such as the Kiti model (243k polygons), we could obtain a fixed frame rate of up to 10 frames per second.

We evaluated our implementation by comparing the visual quality of the scenes depicted in Figure 4, rendered on different number of resources. We used the well known, Visual Difference Predictor metric (VDP) [Dal93] for comparing the visual quality of different frames. The VDP provides the percentage of pixels of an image which would be perceived differently by a human observer when compared to a

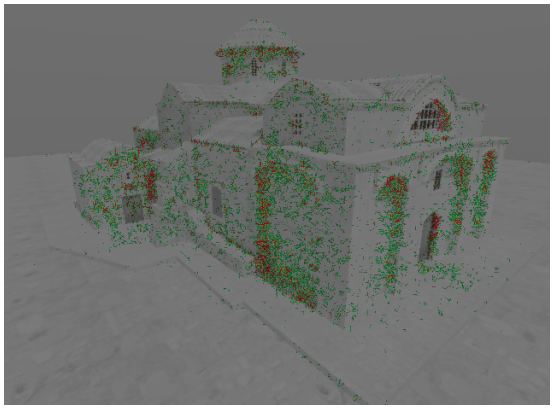
(a) Gold Standard rendered on 96 resources



(b) VDP output of frame rendered on 32 resources



(c) VDP output of frame rendered on 64 resources



(d) VDP output of frame rendered on variable resources

Figure 7: The VDP comparison for frame 70 from the Kiti interactive sequence. In the VDP output images, the grey pixels depict no perceivable difference. The green pixels are used to represent low probability of noticeable difference, while the red pixels are used for depicting high probability.

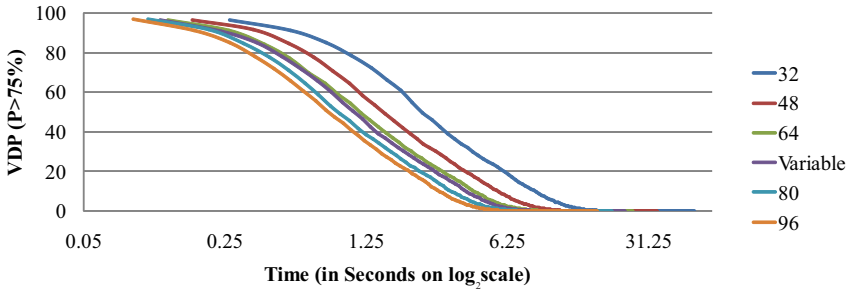
Scene	Frames per Second
Cornell Box	6.66
Kiti	6.00
Race Car	5.00
Sponza	2.50

Table 1: The table depicting the fixed frame rate used for various scenes.

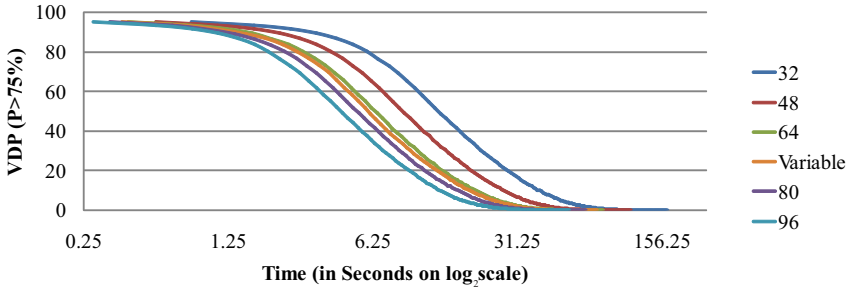
high quality gold standard. We present three kinds of visual comparisons in the following subsections, one while interactively rendering the scene at a fixed frame rate, second by rendering a static image with accumulation and the third comparing reconstruction quality for two hundred frames. For these comparisons, we employed 96 identical processors from the desktop grid to eliminate heterogeneity as a variable.

6.1. Interactive Sequence

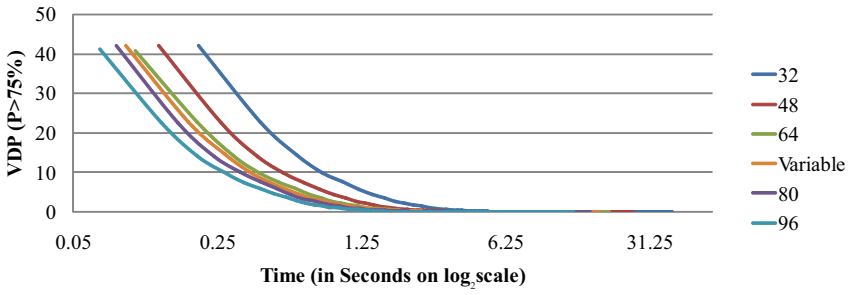
In the first case, we rendered the same interactive sequence using different resources at a fixed frame rate (see Table 1). The graphs in Figure 5 compare the VDP metric with a probability of detection greater than 75% for these sequences. We use the sequence rendered using 96 resources as the gold standard. These graphs show that, not surprisingly, with lower number of resources we get less result packets in the imposed time-constraint which means greater reliance on reconstruction to fill in the missing information as shown in the Figure 6. This deteriorates the image quality as expected and is confirmed by the VDP results (see Figure 7). For a predominantly indirectly lit scene such as Sponza, the VDP values are especially high. This is the result of the randomness introduced by path tracing only four samples per pixel for each update. As an indicator of the randomness, we provide a data series in Figure 5 comparing two sets of sequences both rendered with 96 resources, to serve as a reference. In addition to using fixed number of resources, we also present



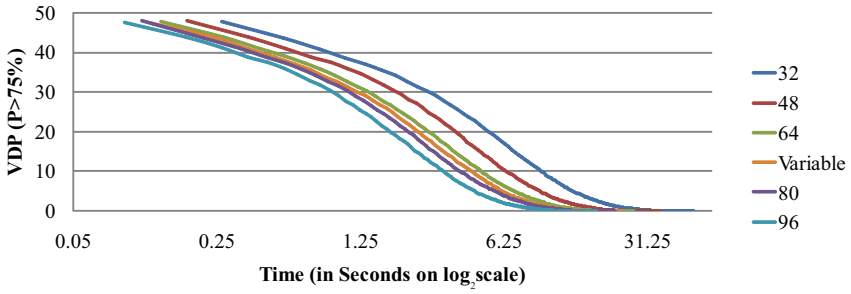
(a) Cornell Box



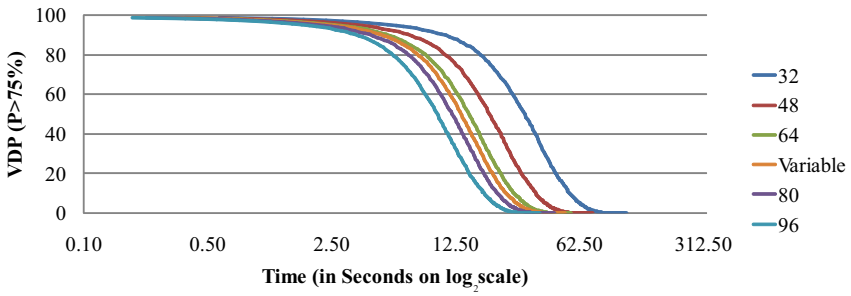
(b) Kalabsha



(c) Kiti

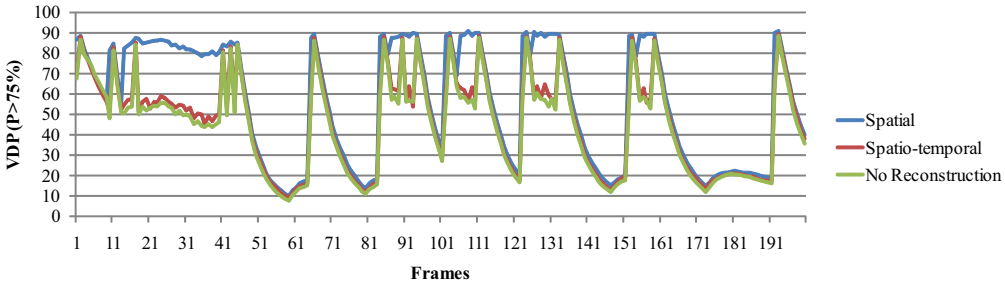


(d) Race Car

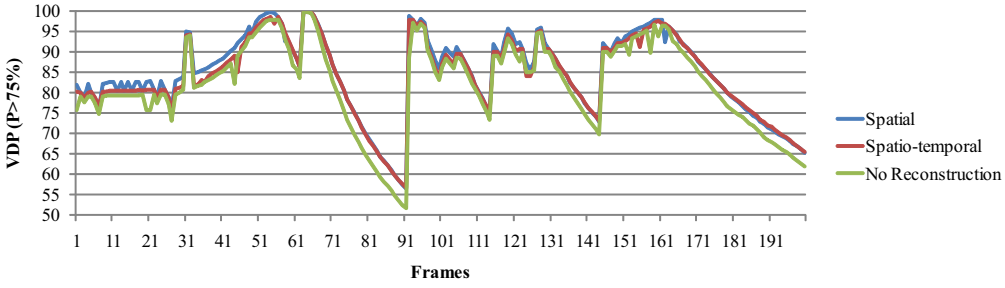


(e) Sponza

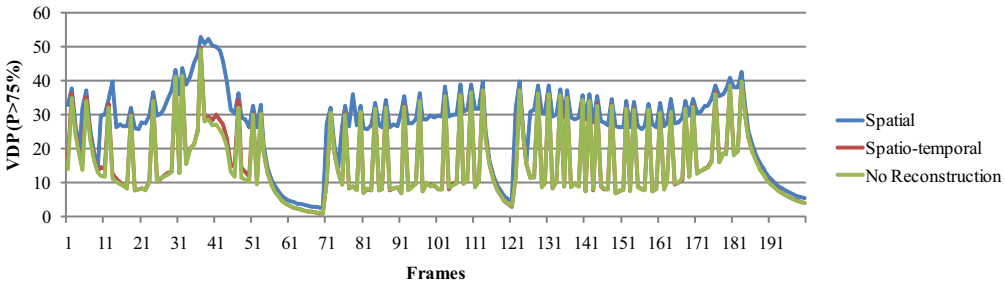
Figure 8: The graphs depicting the visual quality convergence of a static image with different resources for five scenes.



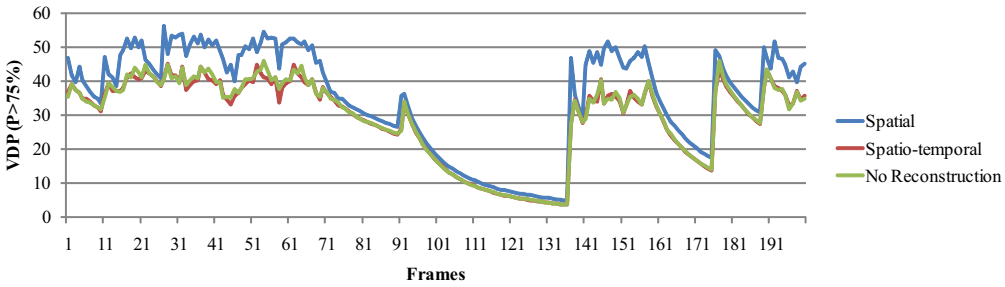
(a) Cornell Box



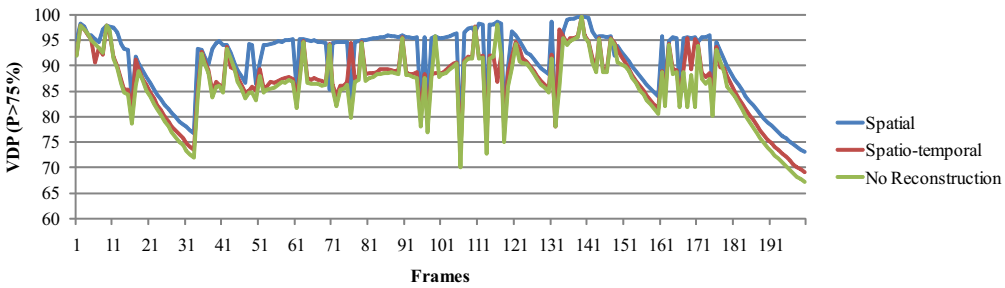
(b) Kalabsha



(c) Kiti



(d) Race Car



(e) Sponza

Figure 9: The graphs comparing visual quality of reconstruction methods for interactive sequences.

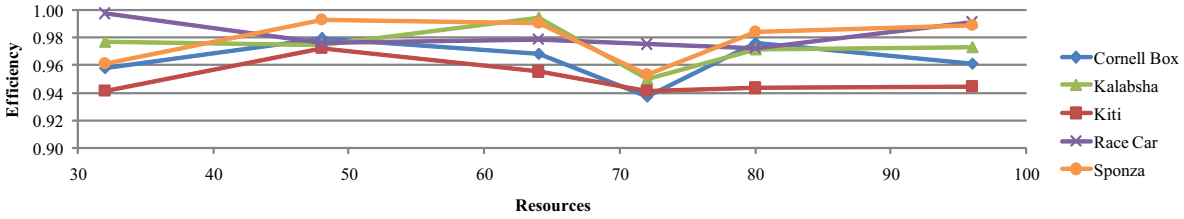


Figure 10: The graph showing efficiency of our method for rendering on different number of resources.

a data series for variable resources in Figure 5 where the number of resources varies per frame of the sequence. The resources, R at frame i are given by:

$$R_i = 96 - 24\sin(\tfrac{2\pi i}{100})$$

The variation of the resources was chosen to be sinusoidal to illustrate our algorithm's ability to compute at fixed frame rate with any number of resources. The system can handle sharp variations of resources as explained in Section 4.3 by skipping a frame in the worst case scenario.

6.2. Static Image

In the second case, we rendered a single static image using different number of resources. The graphs in Figure 8 compare the VDP values showing how the images converge with time. We use the converged image after 200 frames as the gold standard for evaluating the VDP metric. Even in the best case scenario (Figure 8c), it takes about a second for the VDP to fall below the 3% threshold where the two images are assumed to be the same. The variable resources data series uses the same function as before.

We calculated the efficiency of our method by comparing the time it took for rendering these images with the time it took to render using a single processor, see Figure 10. We found that the efficiency of the master did not decrease with an increase in number of workers on our test bed. The master running on a quad-core machine was not a bottleneck and with the use of non-blocking synchronisation, it scaled almost linearly on our test bed. The major bottleneck of the system was the rendering process. When we synthetically reduced the rendering cost to zero, we obtained about 55-60 fps, which is an upper bound on the frame rate due to the available network bandwidth. When the master starts lagging due to too many workers, an approach similar to a hierarchy of masters can be used where the user would need exclusive access to more than one machine for running the masters. Also the load on the master can be reduced by decreasing the frequency with which the workers send requests to the master. This can be achieved by increasing the work load of the workers by calculating more samples per pixel in such a case.

6.3. Frame Reconstruction

Finally, we compare the visual quality of our reconstruction for interactive scenes. The graphs in Figure 9 show the advantage of using spatio-temporal reconstruction in contrast to spatial reconstruction only (as used for static images in [ADD*09]). The VDP plot for spatial reconstruction is similar to the spatio-temporal reconstruction while the image accumulates. However, during the transition period while the user is interacting with the scene, the VDP for spatial reconstruction is considerably higher than the spatio-temporal reconstruction. During this period, the path tracing noise is especially high due to the low number of samples for each of the frames. Therefore to reduce this temporal noise, it is beneficial to use previous samples taking into account discontinuities for improving the visual quality of the frames. For these comparisons we used frames with no missing pixels as the gold standard. Furthermore, we plot a data series for each scene by comparing two sequences with no reconstruction to indicate the path tracing noise as before. The VDP plot of spatio-temporal reconstruction closely follows that of no reconstruction, indicating that our reconstruction method has a minimal impact on the visual quality while employing the quality constraints specified in the Section 5. On the other hand, if a fixed frame rate is specified instead of using quality constraints, reconstruction artefacts are visible when lower number of resources are used as shown in Figure 6. In such a case, to prevent deterioration of visual quality due to reconstruction artefacts beyond acceptable limits, either the fixed frame rate must be lowered or more resources must be employed.

7. Conclusions and Future Work

We have presented a novel fault-tolerant interactive rendering algorithm. This algorithm allows the users to achieve high-fidelity interactive rendering at a low cost using volatile computing resources such as a desktop grid. Desktop grids have been generally used for high-throughput computing, but using our method, we have demonstrated their potential for performing interactive visualisations. The heuristics employed by our method allowed successful monitoring of the variability in computational power to provide a smooth user interaction. Although our implementation used path tracing

for calculating the luminance, other point-sampling based rendering algorithms can be similarly adapted for interactive visualisations.

Our experiments indicate that the rendering still remains the major bottleneck in our system and any advances which would make it faster would potentially improve the performance of our system. Recently GPU-based ray tracing engines have emerged, which provide significant speed-up over CPU-based ones. We believe that in future, when the hardware for running such ray tracing engines becomes widespread, our method will naturally be able to exploit this hardware on idle machines to provide higher quality interactive rendering on desktop grids.

We have found that the current frameworks for computing on desktop grids are geared towards high-throughput computing and it takes a few minutes for them to transfer the files and start the job on idle machines. Future work will investigate ways for quick start of computations on desktop grids so that any ramp-up time can be minimised. Also, research needs to done to find a better way of synchronising scene state on the workers, so that it does not need to be communicated with each job packet.

8. Acknowledgement

The work of reconstructing rendering results using image reconstruction techniques formed part of Usama Mansour's PhD at the University of Warwick, before his tragic death in 2007. We wish to thank the Stanford Computer Graphics Laboratory for the Bunny model which was used as part of the Cornell Box, Marko Dabrovic for the Sponza model, Jassim Happa and Vedad Hulusic for their help with the Kiti and the Kalabsha models.

References

[ACD08] AGGARWAL V., CHALMERS A., DEBATTISTA K.: High-Fidelity Rendering of Animations on the Grid: A Case Study. In *Eurographics Symposium on Parallel Graphics and Visualization* (Crete, Greece, 2008), Favre J. M., Ma K.-L., (Eds.), Eurographics Association, pp. 41–48.

[ADD*09] AGGARWAL V., DEBATTISTA K., DUBLA P., BASHFORD-ROGERS T., CHALMERS A.: Time-constrained High-fidelity Rendering on Local Desktop Grids. In *Eurographics Symposium on Parallel Graphics and Visualization* (Munich, Germany, 2009), Debattista K., Weiskopf D., Comba J., (Eds.), Eurographics Association, pp. 103–110.

[BWS03] BENTHIN C., WALD I., SLUSALLEK P.: A Scalable Approach to Interactive Global Illumination. *Computer Graphics Forum 22*, 3 (2003).

[CDR02] CHALMERS A., DAVIS T., REINHARD E. (Eds.): *Practical Parallel Rendering*. A. K. Peters, Ltd., 2002.

[CSL06] CHONG A., SOURIN A., LEVINSKI K.: Grid-based computer animation rendering. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (New York, NY, USA, 2006), ACM, pp. 39–47.

[Dal93] DALY S.: The visible differences predictor: an algorithm for the assessment of image fidelity. In *Digital images and human vision*. MIT Press, Cambridge, MA, USA, 1993, pp. 179–206.

[GLH*08] GAO J., LIU H., HUANG J., BECK M., WU Q., MOORE T., KOHL J.: Time-Critical Distributed Visualization with Fault Tolerance. In *Eurographics Symposium on Parallel Graphics and Visualization* (Crete, Greece, 2008), Favre J. M., Ma K.-L., (Eds.), Eurographics Association, pp. 65–72.

[Kaj86] KAJIYA J. T.: The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1986), ACM, pp. 143–150.

[KK02] KOLLIG T., KELLER A.: Efficient multidimensional sampling. *Computer Graphics Forum 21*, 3 (2002).

[LLM88] LITZKOW M., LIVNY M., MUTKA M.: Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems* (June 1988).

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive Rendering using the Render Cache . In *10th Eurographics Workshop on Rendering* (Granada, Spain, 1999), Lischinski D., Larson G. W., (Eds.), Eurographics Association, pp. 19–30.

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive Global Illumination using Fast Ray Tracing . In *EGWR02: 13th Eurographics Workshop on Rendering* (Pisa, Italy, 2002), Debevec P., Gibson S., (Eds.), Eurographics Association, pp. 15–24.