Eurographics Symposium on Parallel Graphics and Visualization (2012)
H. Childs, T. Kuhlen, and F. Marton (Editors)

Time-constrained Animation Rendering on Desktop Grids

Vibhor Aggarwal[1,2,†], Kurt Debattista[2], Thomas Bashford-Rogers[2] and Alan Chalmers [2]

[1]Accenture Technology Labs, India
[2]The Digital Lab, University of Warwick, United Kingdom
[†] vibhor.aggarwal@accenture.com

Abstract

The computationally intensive nature of high-fidelity rendering has led to a dependence on parallel infrastructures for generating animations. However, such an infrastructure is expensive thereby restricting easy access to high-fidelity animations to organisations which can afford such resources. A desktop grid formed by aggregating idle resources in an institution is an inexpensive alternative, but it is inherently unreliable due to the non-dedicated nature of the architecture. A naive approach to employing desktop grids for rendering animations could lead to potential inconsistencies in the quality of the rendered animation as the available computational performance fluctuates. Hence, fault-tolerant algorithms are required for efficiently utilising a desktop grid. This paper presents a novel fault-tolerant rendering algorithm for generating high-fidelity animations in a user-defined time-constraint. Time-constrained computation provides an elegant way of harnessing desktop grids as otherwise makespan cannot be guaranteed. The algorithm uses multi-dimensional quasi-random sampling for load balancing, aimed at achieving the best visual quality across the whole animation even in the presence of faults. The results show that the presented algorithm is largely insensitive to temporal variations in computational power of a desktop grid, making it suitable for employing in deadline-driven production environments.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.1]: Hardware Architecture—Parallel processing; Computer Graphics [I.3.2]: Graphics Systems—Distributed/network graphics; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Animation; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Ray tracing;

1. Introduction

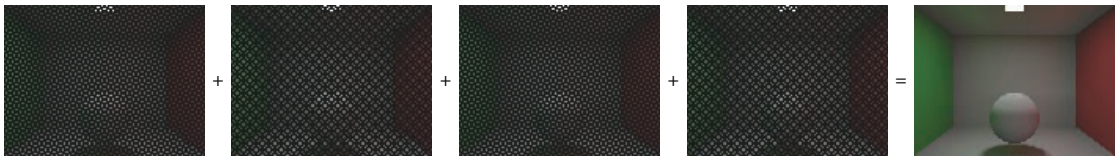
The generation of high-fidelity animations is a time consuming process, employed frequently by modern media industry for creating eye-catching visuals [KFC*10]. High-fidelity rendering is typically carried out by solving the rendering equation [Kaj86] using Monte-carlo estimation. This requires multiple samples to be computed for each pixel to obtain a converged solution, making the process computationally expensive. Parallel rendering infrastructure, known as a render farm, is frequently employed to produce the animations in a reasonable time. The cost of procuring and maintaining such an infrastructure prohibits ubiquity. In contrast, a desktop grid leverages the computational power from idle workstations in an institution at a minimal cost. However, these machines are non-dedicated and there is no guarantee that the tasks assigned to them would finish in time. This is merely an opportunistic environment where the resources process secondary tasks while the primary user is away.

The variable nature of computing on desktop grids makes it difficult to estimate the time span of a computation. However, restricting the computation time makes the employment of desktop grids attractive. Furthermore, a fault-tolerant algorithm designed with the intent of creating the highest quality animation in a given time limit by efficiently employing any idle computational power would be very attractive for rendering on variable resources. This would, for example, be valuable for an animator to leverage desktop grids in a production environment to obtain the best possible animation in a given amount of time from the available resources; this may be used if a new idea needs to be tested for which the expensive dedicate resources may not be currently available.

The rendering of animations on desktop grids conforms to the conventional view of them as a high-throughput re-

DOI: 10.2312/EGPGV/EGPGV12/149-158

(a) Tile-based Sampling



(b) Quasi-random Sampling

Figure 1: Image subdivision techniques [ADD*09]

source and therefore proper load balancing is important, as it has a significant effect for long running computations. Some frames of an animation may require more computational time than others due to the changes in scene and lighting complexity. Therefore, the computation needs to be load balanced across the frames in addition to load balancing on parallel resources when rendering towards a deadline. An unbalanced load would lead to undesirable differences in the visual quality of the different frames of the animation. This can be avoided by progressively updating the whole solution.

The traditional approach of scheduling frames independent of one another while rendering animations in parallel, such as those presented in [ACD08] and [CSL06], is not fault-tolerant. They would have to rely on conventional fault-tolerance strategies such as redundancy and checkpointing, which inhibit performance. In addition, animations are generally synthesised by either rendering a frame until it is finished or by spending a fixed amount of time on each frame, rather than imposing a time-constraint on the whole computation. Therefore, traditional approaches require modification for effectively employing desktop grids for time-constrained animation rendering.

Tile-based image subdivision is usually employed for parallel rendering [CDR02] so that multiple tiles can be independently rendered in parallel, see Figure 1a. The image is formed by combining each of the rendered tiles. Redundancy could be employed for fault-tolerance to ensure all the tiles are computed while rendering on a variable system. If a tile fails to render due to a delay or fault, a duplicate copy of the tile would be computed. However, this is not ideal when rendering towards a deadline as it can add substantially to the rendering time, thus hindering performance.

An improved strategy would be to subdivide into groups of pixels, chosen quasi-randomly [ADD*09, ADBR*10] over the complete image space instead of using tiles. This enables a fair coverage of the whole image per job, see Figure 1b . If a job fails to complete, image reconstruction techniques may be employed to fill in the missing data. The advantage of using a quasi-random sequence over a purely random sequence is that it provides low discrepancy (fills the space more uniformly). On the other hand a regular sampling pattern would lead to undesirable structured noise in the presence of faults when compared to quasi-random sampling. The imposed time-constraint for the presented approach is less strict than that used for previous work, as the aim is to render at a higher quality. Hence, the reliance on reconstruction for algorithms presented in this paper is minimal.

This paper extends the idea of combining sparse sampling and image reconstruction as a fault-tolerant mechanism for rendering animations in a time-constrained fashion on desktop grids. Two algorithms are presented and compared, which show that by employing multi-dimensional quasi-random sampling through space and time, the quality of the whole animation can be progressively enhanced. Also, this allows the system to become resistant to temporal variations in the computational power of the desktop grids and changes in the computational complexity across the frames of the animation.

This paper is organised as follows: Section 2 contains the related work. Section 3 discusses the novel time-constrained fault-tolerant animation rendering algorithms, and their implementation is described in Section 4. A comparison be-

tween the two algorithms is presented in Section 5. The paper is concluded and future work is discussed in Section 6.

2. Related Work

2.1. Time-constrained Rendering

Rendering systems have often been subjected to time-constraints due to the various possibilities of continually adapting and refining the computation. Funkhouser and Séquin [FS93] devised a greedy algorithm for choosing the appropriate level of detail while maximising the visual quality such that the cost associated with rendering at that refinement was less than the constraint. Gobbetti and Bouvier [GB99] enhanced this approach by using continuous level of detail models. An importance metric was presented by Gao et al. [GLH*08] for distributed visualisation of large data sets to determine the rendering order and the level of detail for a block of data set based on view-dependent, application-dependent and data-dependent criteria.

Reisman et al. [RGS00] used time-constraints for interactive parallel ray tracing. They used a progressive sampling strategy based on Delaunay triangulation while treating the image plane as a continuous space. They refined their solution until a given deadline and then reconstructed the image from calculated samples using piecewise linear interpolation. Debattista et al. [DSSC05] provided a framework for controlling the pixel quality in a time-constrained setting for generation of high-fidelity images without perceivable difference. They proposed a regular expression to specify the pixel computations based on different components. Debattista [Deb06] further enhanced the approach by using time-constraints with a progressive selective rendering pipeline.

2.2. Parallel Rendering

A detailed survey of parallel rendering techniques especially in the context of global illumination and ray tracing is presented by Chalmers et al. [CDR02]. Ray tracing algorithms are relatively easy to parallelise if the entire scene description can be duplicated on each processor, as each processor can be designated to independently work on a part of the image-space. However, load balancing can be challenging.

Badouel and Priol [BP89] described a dynamic demand-driven load balancing based on the master-worker paradigm whereby each worker is assigned a 3×3 tile of pixels when it becomes idle. Reisman et al. [RGS00] devised a dynamic load balancing strategy for progressive ray tracing on distributed clusters exploiting temporal coherence for obtaining interactive rates. The image was subdivided into regions which were assigned to separate processors and during runtime the regions were dynamically adjusted to rectify load imbalances. Aggarwal et al. [ACD08] presented a two-pass algorithm for rendering animations on a computational grid. The first pass calculated the irradiance cache [WRC88] data which was distributed in the second pass for rendering the animation to eliminate visual artefacts and speed-up the process. Yao et al. [YPZ10] presented a system for parallel animation rendering on distributed resources, which took advantage of the spatial and temporal coherence between animation frames while scheduling parallel tasks.

Chong et al. [CSL06] presented a system for rendering animations on a computational grid. Their focus was to develop a lossless compression algorithm for transferring data between the nodes. Gonzalez-Morcillo et al. [GMWV*10] used a multi-agent architecture for decentralised rendering which employed importance maps to decide the workload distribution. Many scientific visualisation algorithms have been modified for parallel rendering using grid computing. A survey of such techniques can be found in [BBC*05,Mel08]. Research in this area has focused on reducing the impact of such strategies by minimising redundant computations, for example [GLH*08,ZA10].

The approaches mentioned above were either not designed for fault-tolerance or used traditional fault-tolerance mechanisms and hence they are not suitable for time-constrained computation on a desktop grid. Aggarwal et al. [ADD*09] presented a fault-tolerant approach for time-constrained image rendering on a desktop grid which employed quasi-random sampling for job subdivision. This was further enhanced for interactive rendering in [ADBR*10]. However, a simple extension of this approach for rendering high-fidelity animations presented in Section 3.1 leads to sub-optimal results as it is susceptible to temporal variations in computational power of a desktop grid (see Section 5).

3. Fault-tolerant time-constrained animation rendering

This section presents two fault-tolerant algorithms for rendering animations on variable resources in a user-specified time interval.

3.1. Straightforward Approach

A straightforward approach for rendering an animation under a time-constraint on a desktop grid would be to divide the time-constraint equally for each frame of the animation. The task then becomes to render all the frames with Equal Time-constraint Per Frame (ETPF) approach in a manner similar to the ones presented in [ADD*09]. Each frame can be subdivided into sets of quasi-random pixels and then rendered in parallel using the master-worker paradigm and reconstruction may be used in case of missing pixels.

The ETPF approach has two major limitations. Firstly, it would be susceptible to temporal variations in computational power of the desktop grid. If the computational power of the desktop grid varies significantly in the duration of the total time-constraint imposed, then some frames would be rendered at a higher quality than the others resulting in uneven frame quality across the animation. Secondly, spending

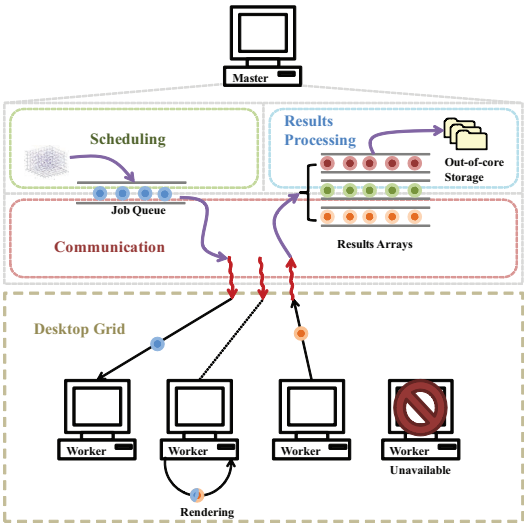


Figure 2: The overview of the time-constrained animation rendering system showing the interactions inside the master and between the master and the desktop grid.

equal time for each frame would also result in non-uniform visual quality since the computational complexity can vary significantly across different frames of an animation. Hence, to overcome these two limitations, a better load balancing strategy is required for rendering animations under time-constraints on variable resources.

3.2. Multi-dimensional Quasi-random Sampling Approach

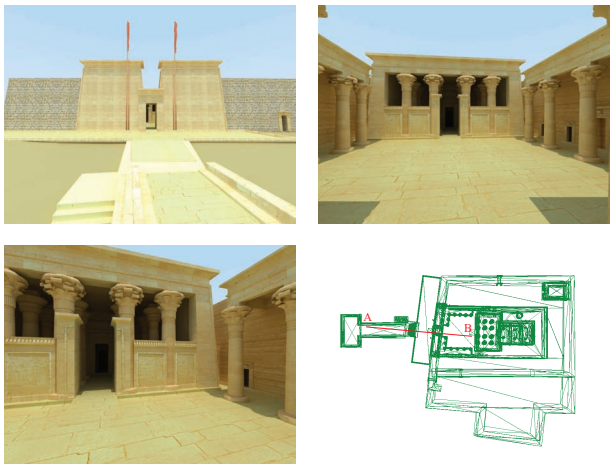
An enhanced strategy for rendering animations under time-constraint on a desktop grid would be to use a Multi-dimensional Quasi-random Sampling (MQS) approach for subdividing the computations. This would entail that each job would consist of rendering pixels which would be spread not only on the image plane of a single frame, but they would be quasi-randomly selected across multiple frames as well. An animation can be considered as a volume of pixels which can be sampled using a three-dimensional quasi-random sequence.

The two major limitations of the ETPF approach can be overcome by quasi-randomly sampling in three dimensions. The MQS approach would be more robust to temporal fluctuations in computational power of a desktop grid in contrast to the ETPF approach, as it does not need to decompose the time-constraint for each frame and hence any pixel of the animation may be scheduled at any given time. Also, it would achieve better load balancing by scheduling pixels from multiple frames simultaneously and therefore tackling the issue of variance in computational complexity across the animation. Furthermore, the MQS approach has another advantage over the ETPF. It progressively refines the whole animation and hence this gives the flexibility to stop and later continue the computation at any given time. In contrast, the ETPF approach employs a progressive rendering algorithm but the whole approach is not progressive as it tackles one frame at a time.

4. Implementation Details

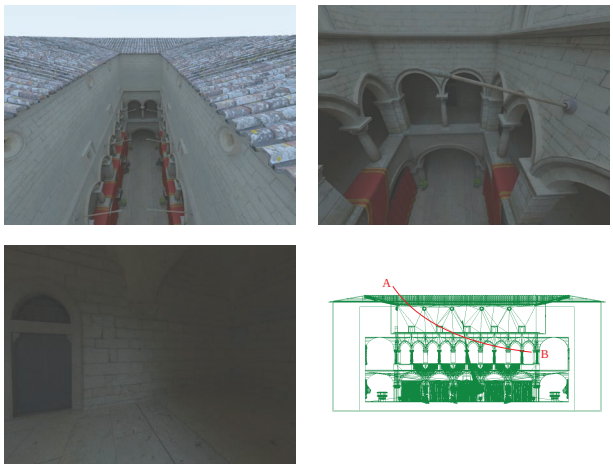
The implementation of the MQS approach needs to be carefully planned. The details for each pixel of the animation, such as pixel colour and number of samples computed, need to be stored in memory as any of the pixels of the animation may be processed at any given time. Even for a small animation of 720 frames with 1024×768 resolution, storing this data would require approximately 3.16 GB of memory. To overcome this potential problem, an out-of-core storage mechanism is necessary and memory mapped files were employed for this implementation.

An overview of the system is depicted in Figure 2. First, the scheduler divides the computation into smaller jobs and places them on a job queue. Next, these are communicated to idle workers executing on the desktop grid when a request is received from them. They then process the job and send the results back. As pixels from multiple frames can be scheduled at the same time, multiple result arrays are used for storing the received data individually for each frame. A multi-threaded architecture is employed on the master to handle and prioritise the communication, such that the results are transferred to the out-of-core storage only when idle. This prevents the master from becoming a bottleneck in the whole process. A producer-consumer problem arises in the system as the workers produce the results while the master transfers (consumes) them to the out-of-core storage. Therefore, a balance between the rate of production and consumption is



(a) Kalabsha (861k polygons)

(b) Kiti (243k polygons)



(c) Sponza (262k polygons)

Figure 3: Start (top-left), middle (top-right), end (bottom-left) frames and the animation path (bottom-right) from A to B chosen for the three scenes.

Scene	Time-constraint (minutes)	Number of Frames
Kalabsha	360	720
Kiti	120	720
Sponza	150	240

Table 1: Time-constraints used for various animations

needed since the master has a limited memory space along with a managed synchronisation between the threads. The size of the job sent to the workers can be dynamically adjusted to maintain the equilibrium, as a larger job size would decrease the rate of production and vice versa.

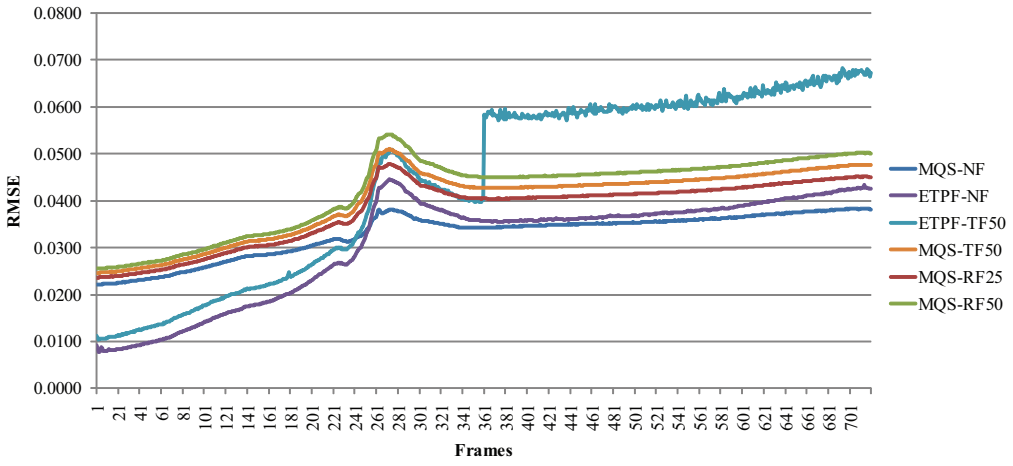
The scheduler needs to keep track of the time-constraint and monitor the job queue before adding more jobs based on one of the two approaches presented. For the ETPF approach, jobs were scheduled by splitting each frame quasi-randomly, and rendering each frame separately for equal portions of the total time-constraint. The animation subdivision for the MQS approach was carried out using a three-dimensional quasi-random Sobol sequence [Sob67]. A base-2 Sobol sequence was used rather than using those presented in [KK02] as they are valid for a single-dimension only. However, a three dimensional base-2 Sobol sequence is fixed and cannot be changed as described in [KK02]. Hence, to increase the fault-tolerance, the sequence was shifted circularly to obtain a different grouping of set of pixels between iterations.

5. Results

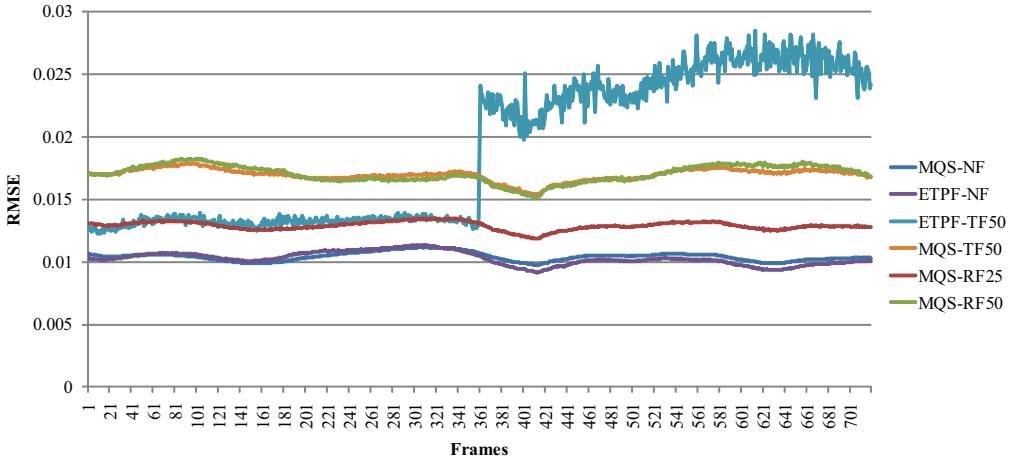
Both the algorithms have been implemented and tested on a desktop grid consisting of twenty four machines with two dual-core AMD Opteron processors running at 2.6GHz at each node. Each machine also had 8GB RAM shared among the four CPU cores. Each of the 96 cores were used independently, as the number of idle CPUs in a machine vary with the load on it. The rendering was carried out by the workers using path tracing [Kaj86], however, other point sampling methods can also be employed. The animation frames were rendered at a resolution of 1024×768. The nearest neighbour algorithm was used in the rare cases where reconstruction was required.

The two time-constrained rendering approaches were compared for the three animation sequences depicted in Figure 3. The computational complexity of the Kiti animation is fairly constant across the animation while it varies substantially for both the Kalabsha and the Sponza animations. The time-constraints used for rendering and the number of frames for the animations are listed in Table 1. The time-constraints were chosen to be approximately 10% of the time it took to render the reference animation on the desktop grid.

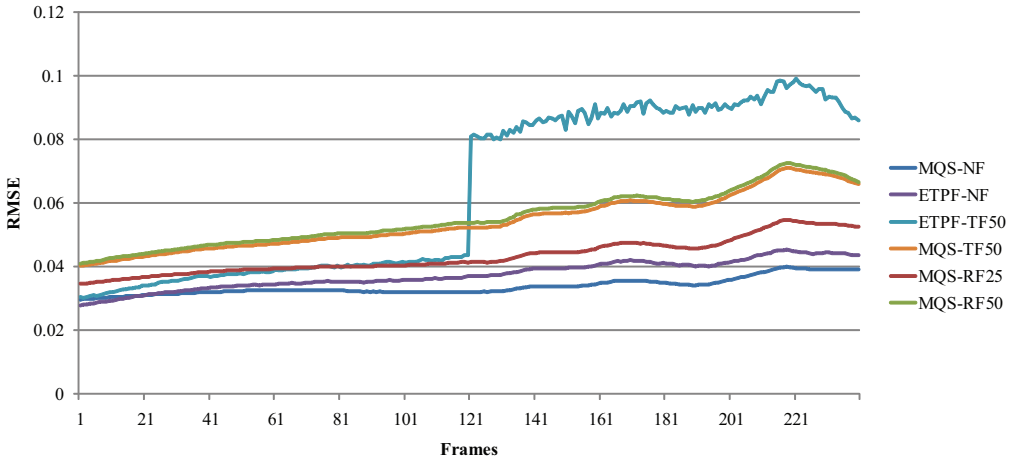
Three types of fault variations were used for comparisons: no faults (NF), random faults (RF) and temporal faults (TF).



(a) Kalabsha



(b) Kiti



(c) Sponza

Figure 4: The RMSE comparisons for the animations rendered with two algorithms under different fault models

(a) ETPF-TF50 (b) MQS-TF50 (c) Reference Frame

Figure 5: Cropped portion of a frame from Kalabsha animation for comparing the visual quality of MQS and ETPF algorithms. Note that the algorithms were constrained to only 10% of the time it took to compute the reference.

The whole desktop grid was dedicated to render the animations for the NF condition. The unpredictable nature of shared resources at run-time on a desktop grid was modelled by using RF. In order to simulate RF condition, a result sent to the master was rejected with a probability of either 25% (RF25) or 50% (RF50). This was achieved by employing a Mersenne Twister pseudo-random number generator [MN98]. Finally, to mimic the time-variant nature of desktop grids TF was used. For the first half of the time-constraint, 25% random faults were generated while for the second half 75% random faults were generated. On an average, this is similar to RF50 and hence the notation TF50 is used.

5.1. Visual Quality

The visual quality of animations computed with the two approaches under different fault variations was measured using the Root Mean Square Error (RMSE) quality metric. A high-quality reference animation was calculated with 1000 samples computed per pixel (SPP) and no time-constraints or faults for comparison. The results for each frame have been presented in Figure 4 and average RMSE over the sequence is shown Table 2. It can be seen from these results that MQS-NF has the best visual quality overall as expected. The MQS and the ETPF approach have similar average RMSE for the NF condition while the MQS is at least 10% better than the ETPF for the TF50 condition. The average RMSE values increase for the three cases of MQS: MQS-NF, MQS-RF25 and MQS-RF50, as the number of faults generated increase.

The RMSE curve for ETPF-NF is very similar to MQS-NF for the Kiti animation as the complexity of this animation is relatively constant. However, for the Kalabsha and the Sponza scenes, ETPF-NF performs worse than MQS-NF for the computationally difficult frames. This is due to the fact that ETPF allocates equal time for each frame irrespective of the complexity, while MQS progressively refines the complete solution.

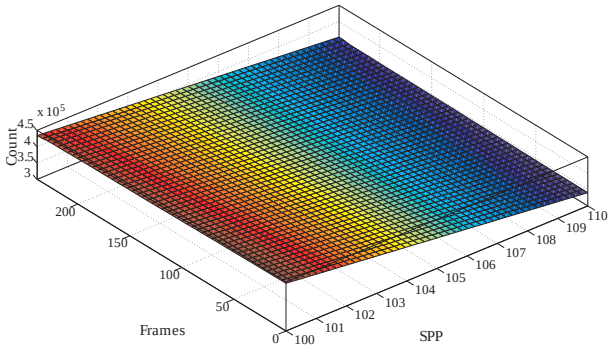
The difference between the two algorithms is much higher for the TF50 condition, see Figure 5. MQS-TF50 and MQS-RF50 have very similar RMSE plots showing that the MQS approach can resist temporal fluctuations of computational power. On the other hand, ETPF-TF50 is similar to MQS-RF25 for the first half of the time-constraint as only 25% faults occur in this period. Hence, it is better than MQS-TF50 for those frames. However, for the second half of the time-constraint with 75% faults, the RMSE curve for ETPF-TF50 shows a drastic increase, while MQS-TF50 doesn't depict any such phenomenon. Hence, the MQS approach aims at obtaining an even visual quality across the animation while the ETPF approach is susceptible to uneven visual quality especially in presence of temporal variations.

Scene	Kalabsha	Kiti	Sponza
MQS-NF	0.0309	0.0102	0.0337
ETPF-NF	0.0309	0.0103	0.0374
MQS-TF50	0.0399	0.0170	0.0540
ETPF-TF50	0.0444	0.0189	0.0636
MQS-RF25	0.0379	0.0129	0.0432
MQS-RF50	0.0419	0.0170	0.0553

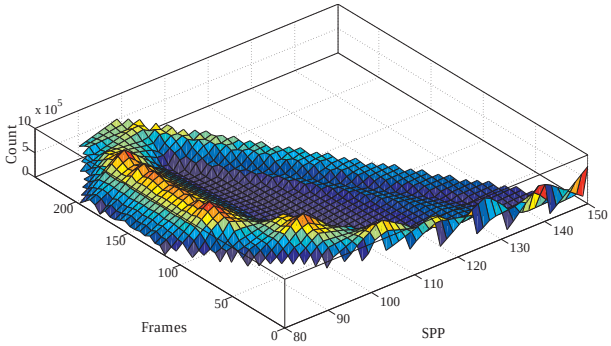
Table 2: Average RMSE Values for the two algorithms with no faults (NF), random faults (RF) and temporal faults (TF)

5.2. Fault-tolerance

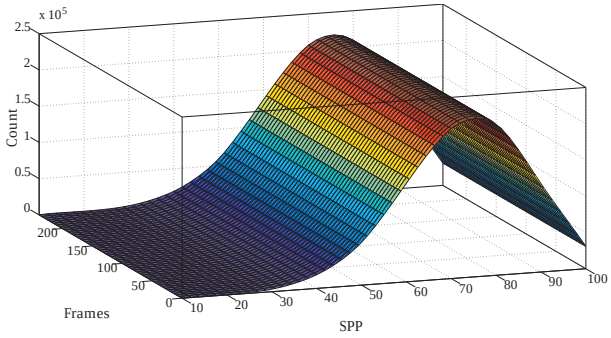
The amount of work completed within a given time-constraint for a path tracing based renderer can be broadly measured by the SPP. Due to the quasi-random sampling employed in the two algorithms described in this paper, SPP can be different for each pixel of the animation. The number of pixels (count) in a frame for which a given SPP have been computed, serves as an indicator of the load balancing of an algorithm in the presence of faults. The graphs between count and SPP for each frame of the Sponza animation have been shown in Figure 6 and indicate the fault-tolerant properties of the presented algorithms.



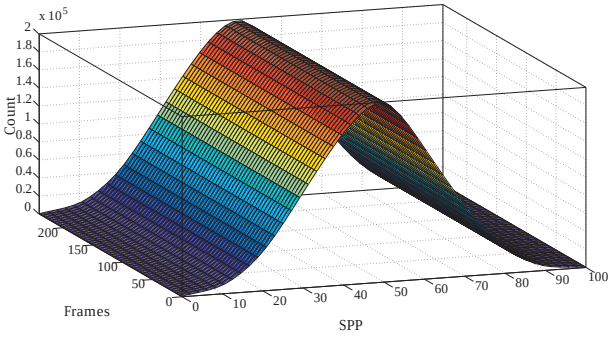
(a) MQS-NF



(b) ETPF-NF

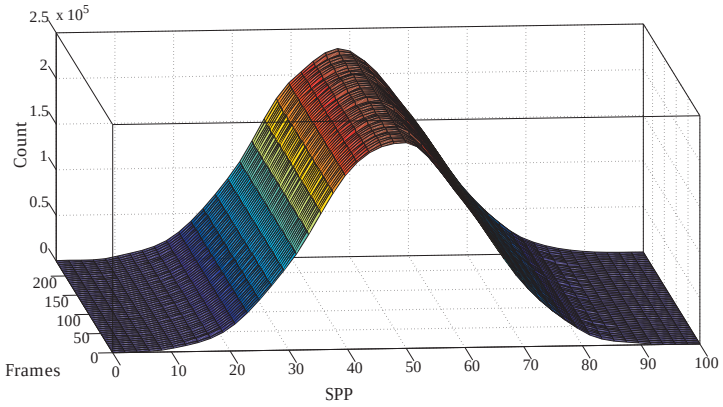


(c) MQS-RF25

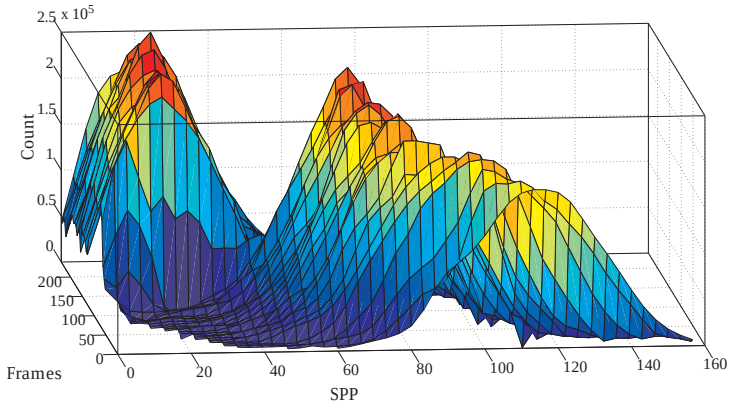


(d) MQS-RF50

Figure 6: Count versus SPP for different frames of the Sponza animation rendered with the two algorithms under various fault models



(e) MQS-TF50



(f) ETPF-TF50

Figure 6: Count versus SPP for different frames of the Sponza animation rendered with the two algorithms under various fault models (continued)

The graph in Figure 6a shows the progressive nature of the MQS approach. The MQS-NF plot shows that all the pixels in the animation have been calculated at a minimum of 100 SPP, and some of them have been further refined to 110 SPP, resulting in a smooth planar plot across the frames. However, the variation of the plot across the animation frames for ETPF-NF (see Figure 6b) depicts that the level of refinement for the ETPF approach is affected by the computational complexity of the frames and hence it calculates unequal SPP for different frames. The variation of SPP versus count for a frame for the MQS approach, in the presence of faults, follows a Gaussian distribution (see Figures 6c, 6d, 6e). Also, this variation is constant across the frames of the animation. As the number of faults increases, the peak of the Gaussian curve shifts leftwards illustrating the fact that less work is completed. Once again the graphs (Figures 6e and 6f) for MQS-TF50 and ETPF-TF50 show the effect of temporal variation. ETPF-TF50 shows two distinct lobes for the two halves of the time-constraint with different number of faults while MQS-TF50 remains unaffected by temporal fault variations and is similar to MQS-RF50.

6. Conclusions, Limitations and Future Work

This paper presented two novel approaches for time-constrained rendering of animations on desktop grids. The results obtained showed that the MQS approach achieved better load balancing than the ETPF approach, while progressively refining the animation in case of faults. The MQS approach tackled the two flaws of the ETPF approach effectively, that is it was insensitive to both temporal variations of computational power of a desktop grid and difference in computational complexity across the animation frames. The MQS approach can restrict performance in the case where each frame requires expensive modifications to the scene as these modifications would need to be performed repeatedly as multiple frames are rendered in each job. Further research needs to be carried out to develop an efficient memory management technique for storing these scene modifi-

cations, possibly in an incremental fashion thereby reducing any performance issues.

The MQS approach presented in this paper used three-dimensional quasi-random sampling for job subdivision. For simplicity, this approach assumes that each pixel of the animation contributes equally to the visual quality of the animation. This assumption can be removed by sampling a function which maps the pixels to their contribution to the visual quality as a fourth dimension, for example [FPSG96], while quasi-randomly selecting the pixels. However, as this function would change while the rendering progresses it would have to be evaluated on the fly. This would both raise the memory requirements of the MQS approach as well as increase the complexity of the sampling process and this will be tackled in future.

Acknowledgements

The authors wish to thank Crytek for the Sponza model, Jassim Happa and Vedad Hulusic for their help with the Kiti and the Kalabsha models. This work was partially supported by EPSRC grant EP/I038780/1.

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), Eurographics Association, pp. 41–48. 2, 3

[ADBR*10] AGGARWAL V., DEBATTISTA K., BASHFORD-ROGERS T., DUBLA P., CHALMERS A.: High-fidelity interactive rendering on desktop grids. *IEEE Computer Graphics and Applications 99*, PrePrints (2010). 2, 3

[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), Eurographics Association, pp. 103–110. 2, 3

[BBC*05] BRODLIE K., BROOKE J., CHEN M., CHISNALL D., FEWINGS A., HUGHES C., JOHN N. W., JONES M. W., RIDING M., ROARD N.: Visual supercomputing: Technologies, applications and challenges. *Computer Graphics Forum 24*, 2 (2005), 217–245. 3

[BP89] BADOUEL D., PRIOL T.: An efficient parallel ray tracing scheme for highly parallel architectures. In *Eurographics Hardware Workshop* (1989), Springer-Verlag. 3

[CDR02] CHALMERS A., DAVIS T., REINHARD E. (Eds.): *Practical Parallel Rendering*. A. K. Peters, Ltd., 2002. 2, 3

[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* (2006), ACM, pp. 39–47. 2, 3

[Deb06] DEBATTISTA K.: *Selective Rendering for High-Fidelity Graphcs*. PhD Thesis, University of Bristol, 2006. 3

[DSSC05] DEBATTISTA K., SUNDSTEDT V., SANTOS L. P., CHALMERS A.: Selective component-based rendering. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2005), ACM, pp. 13–22. 3

[FPSG96] FERWERDA J. A., PATTANAIK S. N., SHIRLEY P., GREENBERG D. P.: A model of visual adaptation for realistic image synthesis. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), SIGGRAPH, pp. 249–258. 10

[FS93] FUNKHOUSER T. A., SÉQUIN C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM, pp. 247–254. 3

[GB99] GOBBETTI E., BOUVIER E.: Time-critical multiresolution scene rendering. In *VIS '99: Proceedings of the conference on Visualization* (1999), IEEE Computer Society Press, pp. 123–130. 3

[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), Eurographics Association, pp. 65–72. 3

[GMWV*10] GONZALEZ-MORCILLO C., WEISS G., VALLEJO D., JIMENEZ-LINARES L., CASTRO-SCHEZ J. J.: A multiagent architecture for 3d rendering optimization. *Applied Artificial Intelligence: An International Journal 24*, 4 (2010), 313–349. 3

[Kaj86] KAJIYA J. T.: The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), ACM, pp. 143–150. 1, 5

[KFC*10] KŘIVÁNEK J., FAJARDO M., CHRISTENSEN P. H., TABELLION E., BUNNELL M., LARSSON D., KAPLANYAN A.: Global illumination across industries. In *ACM SIGGRAPH Courses* (2010), SIGGRAPH. 1

[KK02] KOLLIG T., KELLER A.: Efficient multidimensional sampling. *Computer Graphics Forum 21*, 3 (2002), 557–563. 5

[Mel08] MELIGY A.: Parallel and distributed visualization: The state of the art. In *CGIV '08: Proceedings of the Fifth International Conference on Computer Graphics, Imaging and Visualisation* (2008), IEEE Computer Society, pp. 329–336. 3

[MN98] MATSUMOTO M., NISHIMURA T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation 8*, 1 (1998), 3–30. 7

[RGS00] REISMAN A., GOTSMAN C., SCHUSTER A.: Interactive-rate animation generation by parallel progressive ray-tracing on distributed-memory machines. *Journal of Parallel and Distributed Computing 60*, 9 (2000), 1074–1102. 3

[Sob67] SOBOL I. M.: On the distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R. Computational Mathematics and Mathematical Physics 7* (1967), 86–112. 5

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988), ACM, pp. 85–92. 3

[YPZ10] YAO J., PAN Z., ZHANG H.: A distributed render farm system for animation production. In *Entertainment Computing – ICEC 2009*, vol. 5709. Springer Berlin / Heidelberg, 2010, pp. 264–269. 3

[ZA10] ZHU Q., AGRAWAL G.: Supporting fault-tolerance for time-critical events in distributed environments. *Scientific Programming 18*, 1 (2010), 51–76. 3