Journal of Supercomputing, 58(3):357366, 2011
Post-print version

Parallel Hierarchical Radiosity on Hybrid Platforms

Emilio J. Padrón · Margarita Amor · Montserrat Bóo · Gabriel Rodríguez · Ramón Doallo

Abstract Achieving an efficient realistic illumination is an important aim of research in computer graphics. In this paper a new parallel global illumination method for hybrid systems based on the hierarchical radiosity method is presented. Our solution allows the exploitation of systems that combine independent nodes with multiple cores per node. Thus, multiple nodes work in parallel in the computation of the global illumination for the same scene. Within each node, all the available computational cores are used through a shared-memory multithreading approach. The good results obtained in terms of speedup on several distributed-memory and shared-memory configurations show the versatility of our hybrid proposal.

Keywords Hybrid Platforms · Global Illumination · Hierarchical Radiosity

1 Introduction

Radiosity [3] is one of the best solutions to get a physically-based illumination, essential key for realistic rendering. Unfortunately, the radiosity method, like other global illumination alternatives, has high computational and memory requirements which justifies the use of parallel computing techniques to implement it.

In the last years, the progressive popularization of multicomputers and multicore-based systems makes the design and implementation of efficient parallel algorithms an appealing alternative for high demanding computer graphics techniques. The main challenge nowadays is to take advantage of all the different computational resources in a system, putting together shared-memory

E. Padrón · M. Amor · G. Rodríguez · R. Doallo
Computer Architecture Group, Universidade da Coruña
E-mail: emilioj@udc.es, margamor@udc.es, grodriguez@udc.es, ramon.doallo@udc.es

M. Bóo
Computer Architecture Group, Universidade de Santiago de Compostela
E-mail: montserrat.boo@usc.es

and distributed-memory concepts by means of versatile and efficient hybrid approaches.

There are multiple parallel approaches in the literature that have been proposed to speed up the radiosity calculation. However, most of the existing proposals for parallel global illumination fall in one of these two categories: either purely shared-memory oriented or purely distributed-memory oriented. Shared-memory approaches [10] are simpler and achieve really good speedups, but they present the inherent scalability problems of this kind of systems. Furthermore, they are mostly fine-grain approaches, so a considerable overhead is introduced due to synchronization issues. On the other hand, distributed-memory approaches to parallel global illumination are notably more complex [1, 8] and have typically obtained worse performance, mainly due to the important communication overhead, above all if the geometric data is distributed among the memories of the system nodes.

In [6] we find a hybrid distributed-shared proposal, based on what authors call *task pool teams*, basically an extension of the task pool approach commonly used in SMP (Symmetric Multiprocessing) computing. It is a generic non-specific alternative for irregular algorithms, using global illumination and HR as an example of application. Unfortunately, only results for quite small and unrealistic scenes are shown, so the method can not be considered as a valid solution for real global illumination. More HR specific is the work in [2], but the details about the shared memory part of this hybrid proposal are roughly described and, since the work is previous to the advent and popularization of multi-core processors, it is not clear how it will scale to more than two threads per node (the maximun number of threads used in the paper).

Last trends in global illumination are mostly focusing on the exploitation of GPGPU (General-purpose computing on graphics processing units) [4, 7] taking the most data-parallel parts out of the CPUs to run on the GPU. However, since some important parts of global illumination methods are not suitable for this kind of processing they usually make a coarse approximation of the indirect illumination by using imperfect visibility or a simplified scheme for a plausible indirect lighting.

In this paper we propose a novel parallel global illumination method using the hierarchical radiosity algorithm [5] (HR), a radiosity solution based on an adaptive refinement that gets a good quality/performance trade-off. The irregular and mostly unpredictable workload of this approach makes traditionally difficult to achieve good parallel solutions, especially in distributed-memory contexts. Our parallel design is focused on obtaining a versatile hybrid approach. The message-passing scheme we have implemented allows the parallel execution of HR in independent nodes of a distributed-memory cluster, minimizing the communication among nodes. As far as each node is concerned, a multithreading scheduling for the efficient processing of the input scene in a multi-core environment has been implemented. This scheduling follows a coarse-grain approach, balancing the computational load within a node at a patch level and introducing a minimal overhead.

The paper is organized as follows: Section 2 presents the HR method and Section 3 outlines the generic structure of our parallel proposal. Experimental results and concluding remarks are presented in Section 4 and Section 5.

2 Hierarchical Radiosity Algorithm

Radiosity [3] tackles the global illumination problem by applying a finite element approach to compute the transport of energy in an environment. Thus, the scene to be illuminated is discretized into a set of surface elements, usually called *patches* in the context of radiosity, and the light energy leaving each surface is computed, obtaining the classical discrete radiosity equation:

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij},\ 1 \leq i \leq n; \tag{1}$$

where B_i is the radiosity value (light energy per unit time per unit area leaving the surface) of a patch P_i, computed as the emittance of that patch (E_i, light energy produced by the surface itself, i.e. in case of light sources) plus the energy coming from the rest of the scene and reflected by it. Thus, the term ρ_i is the diffuse reflectance index of P_i, and the summation represents the energy reaching the patch from the other patches of the scene. The interaction or link between two patches in the scene is based on a geometric term called *form factor*, F_{ij}, that represents the proportion of radiosity leaving P_i that is received by the patch P_j.

The hierarchical approach to radiosity [5] is based on the application of a basic idea taken from the classic N-body problem: the importance of small details decreases with increasing distance. Thus, the input patches are subdivided into a hierarchy of surface elements with links with different level of refinement between them that simulate the light transport in the scene.

In general terms, the sequential HR method consists of three main stages: *Initial Stage*, *Linking Stage* and *Iterative Stage*, this last one being the core of the HR process. The *Initial Stage* includes preprocessing work such as building auxiliary data structures to accelerate the visibility determination between patches during the radiosity computation.

In the *Linking Stage* the starting interactions between pairs of patches in the scene are computed, building a list of initial links for each patch in the scene. Basically, two patches are interacting when they are (at least partially) visible to each other. The corresponding form factor is computed and stored for each link. Visibility determination and form factor computation are the main tasks performed in this stage. Of course, since one half of the links are the reciprocal of the other half (P_a linked to P_b usually means P_b linked to P_a), only one visibility computation is needed for each two reciprocal links.

The global illumination of the scene is computed in the *Iterative Stage*. This is an iterative process which computes the energy being transported through all the links in the scene, refining those links when necessary. One common approach is to apply a three-step process to every patch of the scene in each iteration. In the first step (*Refinement*), each link of the target patch is analyzed and adaptively refined when the energy transported through this link exceeds a threshold value. Once the refinement step for a patch is completed, the energy received from the rest of elements in the scene is computed (*Gathering*). After gathering the light energy coming from the rest of the scene, the radiosity values of the patch are coherently updated along the hierarchical structure resulting from the two previous steps (*Sweeping*).

Each iteration is completed once all the patches of the scene have been processed. Then, the convergence is checked, comparing with a certain threshold the difference in the total energy transported between two consecutive iterations. If the convergence criterion is not fulfilled a new iteration begins.

3 Parallel Hierarchical Radiosity

Our parallel approach to HR targets systems that combine several independent nodes with multiple cores per node. The scene is partitioned into non-overlapping sub-scenes, and the computation of each sub-scene is carried out independently in a different node. Our method lies in an SPMD paradigm, with a unique process per node and message passing for communicating updated illumination values among nodes. Within a node, the HR algorithm is applied to a sub-scene concurrently by multiple tasks that exploit the patch-level parallelism in the *Linking* and *Iterative Stages*.

3.1 Distributed-Memory Solution for HR

The irregular behavior of the hierarchical approach to radiosity, based on an adaptive refinement, makes difficult to achieve a good parallel solution, above all in a distributed-memory context. Our approach is based on the minimization of communications and on avoiding to establish an excessive number of synchronization points among the different processes, yet without renouncing to process highly complex scenes. With that aim in mind, only the input patches (coarse geometric data) need to be replicated in the memory of every node. This allows the resolution of visibility queries with no communication at all, and the penalty is not too high anyway, given the large amounts of memory per node and the low storage requirements of the coarse patches.

In Fig. 1, an outline of the parallel algorithm executed in each node is depicted. The three stages seen in the sequential method are carried out in parallel by all the processes in the distributed-memory system, with a unique process running on each node. Communication among nodes is performed through asynchronous non-blocking messages, overlapping communication and computation as much as possible. The steps that involve communication with other nodes are shadowed in the diagram. Within a node, the *Linking* and the *Iterative Stages* can be executed concurrently, spawning multiple threads within the process, as will be described in Subsection 3.2.

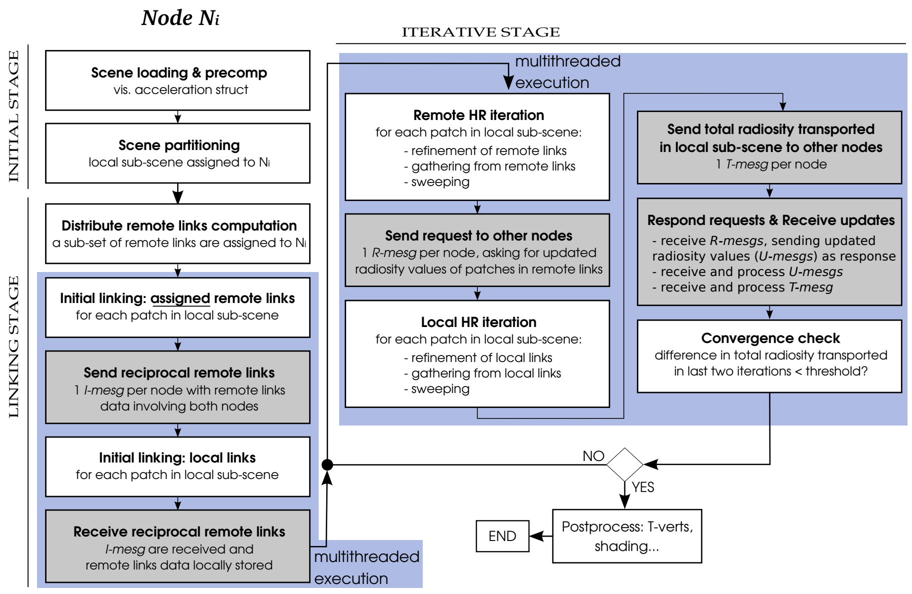
Node Ni
INITIAL STAGE
LINKING STAGE
ITERATIVE STAGE
Scene loading & precomp
vis. acceleration struct
Scene partitioning
local sub-scene assigned to Ni
Distribute remote links computation
a sub-set of remote links are assigned to Ni
Initial linking: assigned remote links
for each patch in local sub-scene
Send reciprocal remote links
1 l-mesg per node with remote links data involving both nodes
Initial linking: local links
for each patch in local sub-scene
Receive reciprocal remote links
l-mesg are received and remote links data locally stored
multithreaded execution
multithreaded execution
Remote HR iteration
for each patch in local sub-scene:
- refinement of remote links
- gathering from remote links
- sweeping
Send request to other nodes
1 R-mesg per node, asking for updated radiosity values of patches in remote links
Local HR iteration
for each patch in local sub-scene:
- refinement of local links
- gathering from local links
- sweeping
Send total radiosity transported in local sub-scene to other nodes
1 T-mesg per node
Respond requests & Receive updates
- receive R-mesgs, sending updated radiosity values (U-mesgs) as response
- receive and process U-mesgs
- receive and process T-mesg
Convergence check
difference in total radiosity transported in last two iterations < threshold?
NO
YES
Postprocess: T-verts, shading...
END

Fig. 1: Outline of the distributed-memory scheduling

The preliminary work regarding the loading of the input scene and the construction of auxiliary structures for accelerating visibility determination is performed concurrently in every node, but is not parallelized (first step of the *Initial Stage* in Fig. 1). As a result of this stage, all the nodes keep the initial patches (coarse geometric data) in its local memory. That will permit to avoid a lot of communication in the next two stages, as commented above.

The other main task carried out during the *Initial Stage* is to distribute the computation among the nodes by making a partition of the scene. Each node assigns itself one of the disjoint sub-scenes resulting from this partition. The process in the node will calculate the global illumination in that local sub-scene. Although a uniform geometric partition has been employed, specifically a regular 3D grid with a final volume optimization to obtain a tight-fitting bounding box of each sub-scene, this parallel proposal is independent of the kind of partition to be applied. Nevertheless, convex geometric partitions allow the exploitation of spatial locality of the objects in a scene.

With regards to the *Linking Stage*, all the patches in the local sub-scene are processed and two different lists of links are built for each patch: links to other patches in the local sub-scene (local links) and links to patches in the rest of the scene (remote links). Since all the initial geometry is accessible in the local memory, no communication among nodes is needed to compute the visibility between patches in different sub-scenes and the whole stage can be completely performed with no interaction between nodes. However, this would mean to replicate visibility computations in different nodes since for each remote link a reciprocal remote link is assigned to another node and both have the same visibility value. Therefore, the *Linking Stage* has been parallelized to avoid the bottleneck due to the duplicate visibility determination of remote links. The total remote links to be computed are previously distributed among the different nodes (*Distribute remote links computation* in Fig. 1). At this point, different strategies can be applied to try and balance the computation of all the links (local and remote) in the system. We have implemented a simple distribution that minimizes the number of messages among nodes: all the remote links between two nodes are assigned to the end with less links already assigned (counting both local and remote links).

After the distribution of the remote links, every node computes its local sub-set and sends the reciprocal remote links computed to the corresponding nodes. To overlap computation and communication this process is split into four steps. Firstly, only the remote links are computed for each patch in the node (*Initial linking: assigned remote links* in Fig. 1). Then, the reciprocal remote links are sent to the corresponding nodes (*Send reciprocal remote links* in Fig. 1): the data sent consists basically in the visibility values and the form factors. At this point, communication overlaps with the computation of the local links for each patch on the node (*Initial linking: local links* in Fig. 1). Finally, the remote links computed in other nodes are received (*Receive reciprocal remote links* in Fig. 1).

In the *Iterative Stage*, the three steps involved in the sequential HR iteration are computed for the local sub-scene using the links computed for each patch as a starting point. This stage entails communication among nodes, since data from remote nodes should be refreshed for each new iteration. Specifically, two different kinds of remote data need to be updated between iterations: radiosity values of patches, for the refinement and gathering of remote links, and the total radiosity transported in each sub-scene, for the convergence checking. A scheduling with 6 steps for the *Iterative Stage*, as depicted in Fig. 1, has been proposed. The objective is to favor an asynchronous and independent execution with few synchronization points among nodes

The first thing a node would do in our scheme would be the processing of the remote links associated with all the patches in the local sub-scene (*Remote HR iteration* in Fig. 1). The decision of splitting up the processing of local and remote links has been taken based on two reasons: firstly, processing the remote energy earlier assures the presence of energy to be transported in every sub-scene, even though some of them have no light sources; on the other hand, the radiosity values from remote patches that interact with the local patches must be updated after each iteration. This update is done by means of message passing, and it means a first message requesting the remote radiosity values needed to the rest of nodes. Our scheduling tries to overlap this communication phase with the processing of the local links.

The second step is to send a message to each of the rest of the nodes (*R-mesg* in *Send request to other nodes* in Fig. 1), asking for the updated radiosity values needed for the next iteration. The information that needs to be sent for this request is only the *id* of the elements whose radiosity value is needed.

In the step three, *Local HR iteration* in Fig. 1, the local links are processed. Once all the energy has been transported in the local sub-scene for an iteration, the total radiosity value obtained is sent to the rest of the nodes in step four, *Send total radiosity transported* (*T-mesg*).

The fifth step, *Respond requests & Receive updates* in Fig. 1, deals with the exchange of messages among nodes. Each node receives request messages, *R-mesg*, that must be properly responded by sending messages with the updated radiosity values requested by each node (*U-mesg*). The *U-messages* sent by the rest of the nodes are also received in this step, together with the *T-messages* with the total radiosity transported in each sub-scene.

The last step, *Convergence check*, does not involve communication and is carried out in every node as in the sequential version.

3.2 HR on SMT Multi-core Processors

At this point, HR computation on shared-memory parallel environments is addressed by applying a multithreading approach to exploit all potential computational resources available in each node: multiple computational cores, all of them with local access to a common memory, as well as potential SMT capabilities. Specific details about this scheduling and the mutual exclusion protocol designed to allow the parallel subdivision of patches during each HR iteration can be read in [9].

Initially, only one thread (*Main thread*) is being executed in the node. The preliminary work regarding the loading of the input scene and the construction of visibility acceleration structures is performed by this thread and is not parallelized within a node (*Initial Stage* in Fig. 1).

After this stage, multiple threads are spawned to carry out the radiosity computation. Thus, there is only one process running on the physical node, but it consists of multiple threads sharing the same virtual address space and exploiting the multiple cores and SMT capabilities available in the node. These threads will work concurrently until convergence is achieved in the *Iterative Stage*. The number of threads to be spawned, t, can be different on each node and can be either the total number of processing cores in the node or not.

A patch is assigned on demand to a thread, all the computation associated with that patch during the stage is carried out by this thread.

Of course, every piece of code executed by the threads must be thread-safe, since they are running simultaneously in a shared address space. Therefore, multiple access to shared data must be satisfied and protected, avoiding race conditions and deadlocks among the threads. All these issues are managed by setting critical sections in the code by means of mutual exclusion (*mutex*) algorithms and operations.

During the *Iterative Stage*, specific actions must be taken due to the refinement process performed during the HR computation. Thus, multiple threads could try to subdivide the same element while refining different links. A link refinement means that either the source or the interacting destination element is subdivided, so different threads may try to subdivide the same element at the same time. Therefore, element subdivision is an important critical section in this scenario.

To obtain an efficient multithreaded HR computation, a simple yet effective mutual exclusion protocol to deal with the refinement process has been implemented. This protocol allows an efficient thread-safe adaptive refinement with a minimal storage cost ($t + 1$ mutex variables for t threads spawned in the node). Specific details about how this protocol works can be read in [9].

During each HR iteration each thread uses a local variable, *localRad*, to accumulate all the radiosity being gathered. A thread-safe shared variable, *totalRad*, is needed to add up the contribution of the energy gathered by all the threads at the end of each iteration.

4 Experimental Results

The HR parallel solution presented in this paper has been tested on a system with eight nodes with 8 GB RAM and two Intel Xeon E5520 2.26 GHz quad core processors per node, with a 2-context SMT configuration enabled on each core (Intel HyperThreading), resulting in a total of 16 virtual processing units per node (though with only 8 physical cores per node). All nodes are equipped with IB 4X DDR cards (Qlogic IBA7220), so they communicate each other through a low latency InfiniBand network with 16 Gb/s of effective bandwidth.

Our parallel implementation was coded using the C programming language (gcc 4.1.2). The POSIX threads library (*Pthreads*) is used to implement all the thread-related issues. Pthreads are the best alternative when writing portable multithreaded code, as it offers a system-level standard library, much more flexible and versatile than higher-level libraries like OpenMP. Message-passing is managed through the MVAPICH2 library, a free implementation of the MPI API especially designed for InfiniBand and other low latency networks.

Two input scenes have been used for our tests (see Fig. 2): *Building* and *Classroom*, with respectively 2880 and 9253 input triangles, and 135480 and 360184 output triangles after the HR refinement. The scene *Building* has multiple identical rooms communicated by doors, so radiosity is transported between contiguous spaces. In contrast, *Classroom* has a unique clear room with many polygons seeing each other.

In the table of Fig. 2c the execution time and the corresponding speedup achieved for the HR computation of the two test scenes are shown. Different configurations of distributed and shared-memory resources have been checked: the first column shows the number of distributed-memory nodes used for the computation, whereas the second column indicates the number of threads spawned per node. Since each node of the target platform has really 8 physical cores, running 16 threads per node allow us to effectively exploit the SMT support available in Xeon processors.

In order to analyze the table, it should be noticed that the different configurations with only one node show a pure shared-memory scenario, allowing

(a) *Building*

(b) *Classroom*

		Building		Classroom	
Nodes	**Thrs**	**Time**	**Sp**	**Time**	**Sp**
1	1	42.91	1	1813.99	1
	2	21.76	1.97	965.86	1.88
	4	11.95	3.59	487.90	3.72
	8	6.52	6.58	256.54	7.07
	16	5.25	8.17	194.47	9.33
2	1	24.16	1.78	1114.30	1.63
	2	12.88	3.33	531.86	3.41
	4	6.90	6.22	297.38	6.10
	8	3.86	11.12	155.91	11.63
	16	3.05	14.07	139.10	13.04
4	1	12.86	3.34	716.02	2.53
	2	6.84	6.27	357.08	5.08
	4	3.70	11.60	190.34	9.53
	8	2.11	20.34	119.03	15.24
	16	1.78	24.11	102.66	17.67
8	1	6.60	6.50	488.94	3.71
	2	3.45	12.44	260.25	6.97
	4	1.89	22.70	132.22	13.72
	8	1.25	34.33	69.96	25.93
	16	1.15	37.31	61.91	29.30

(c) Performance: Time (s) and Speedup

Fig. 2: Test scenes and performance results

us to confirm the good performance of our multithreading approach. Thus, speedups of 8.17 and 9.33 have been obtained for the two scenes, significant values considering that there are only 8 physical cores within a node.

On the other hand, the shadowed rows in the table correspond to a pure distributed-memory configuration, with a unique thread running on each node. we can appreciate the drastic improvement achieved by our parallel approach for the *Building* scene in all cases: a speedup of 6.50 for 8 nodes with 1 thread per node, and up to 37.31 for 8 nodes with 16 threads per node. The *Classroom* scene achieves good results with regards to the multithreaded shared-memory part, but gets a poorer distributed-memory performance, probably due to the work imbalance across the different nodes produced by a more irregular geometric data distribution. Since our parallel HR approach is independent of the scene partitioning method, we expect to improve the results for irregular scenes through spatially adaptive, non-uniform partitions.

5 Conclusions and Future Work

This work approaches the parallelization of the HR method, a reference model in global illumination, in a hybrid context, exploiting distributed and shared memory architectures. Our approach is based on: a workload distribution among nodes through a convex partition of the scene; a minimum number of message-passing communications among nodes thanks to the replication of the coarse geometric data in the different nodes; an efficient multithreaded scheduling within a node based on kernel-level threads, taking advantage of the multiple computing resources with access to a shared memory; and a low-cost mutual exclusion algorithm for the concurrent refinement of the scene.

First results have been obtained using a uniform space partition, but we expect to improve the performance by means of a non-uniform partition that prevent load imbalance among nodes. Besides, the full hybrid system will be enhanced in a future version with an extension to heterogeneous multi-core systems, using GPGPU.

Acknowledgements This work was partially supported by the Ministry of Education and Science of Spain under the contract MEC TIN 2010-16735 and also supported by the Xunta de Galicia under the contracts 08TIC001206PR, INCITE08PXIB105161PR.

References

1. Baiardi F, Mori P, Ricci L (2006) Parallel hierarchical radiosity: the PIT approach. Applied Parallel Computing (LNCS) Volume 3732/2006:1031–1040
2. Caballer M, Guerrero D, Hernández V, Roman JE (2003) A parallel rendering algorithm based on hierarchical radiosity. LNCS 2565:523–536
3. Cohen MF, Wallace JR (1993) Radiosity and realistic image synthesis. Academic Press Professional
4. Dachsbacher C, Stamminger M, Drettakis G, Durand F (2007) Implicit visibility and antiradiance for interactive global illumination. ACM Transactions on Graphics 26(3):61:1–61:10
5. Hanrahan P, Saltzman D, Aupperle L (1991) A rapid hierarchical radiosity algorithm. In: Proc. SIGGRAPH'91, vol 25, pp 197–206
6. Hippold J, Rünger G (2003) Task pool teams for implementing irregular algorithms on clusters of SMPs. In: Proc. International Parallel and Distributed Processing Symposium (IPDPS'03), p 54.2
7. Kaplanyan A, Dachsbacher C (2010) Cascaded light propagation volumes for real-time indirect illumination. In: I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, ACM, New York, NY, USA, pp 99–107, DOI http://doi.acm.org/10.1145/1730804.1730821
8. Padrón EJ, Amor M, Bóo M, Doallo R (2007) A hierarchical radiosity method with scene distribution. In: Proc. 15th Euromicro Conf. on Parallel, Distributed and Network Based Processing (PDP 2007), pp 134–138
9. Padrón EJ, Amor M, Bóo M, Doallo R (2009) High performance global illumination on multi-core architectures. In: Proc. of the 17th Euromicro Conf. on Parallel, Distributed and Network Based Processing (PDP 2009), pp 93–100
10. Singh JP, Gupta A, Levoy M (1994) Parallel visualization algorithms: Performance and architectural implications. IEEE Computer Graphics and Applications 27(7):45–55