Parallelization of a Discrete Radiosity Method

Rita Zrour, Pierre Chatelier, Fabien Feschet, and Rémy Malgouyres

LLAIC, IUT, 63172 Aubière Cedex
{zrour,chatelier,feschet,remy.malgouyres@llaic3.u-clermont1.fr}

Abstract. In this paper we present three different parallelizations of a discrete radiosity method achieved on a cluster of workstations. This radiosity method has lower complexity when compared with most of the radiosity algorithms and is based on the discretization of surfaces into voxels and not into patches. The first two parallelizations distribute the tasks. They present good performance in time but they did not distribute the data. The third parallelization distributes voxels and required the transmission of small part of the voxels between machines. It improved time while distributing data.

1 Introduction

The radiosity method has been widely used in different fields concerned with the exchange of energy. It is mainly used in computer graphics (3D rendering) and in image synthesis to simulate global illumination and lighting effects. Most radiosity methods require large memory and time computation. This leads researchers to parallelize radiosity algorithms trying to minimize both time and memory.
Many parallelizations of different radiosity algorithms have been proposed [10, 14, 3, 2, 13, 9, 4, 11, 7, 12, 8, 16]. These algorithms are characterized by their memory system and their parallelization procedure that contribute together to the final parallelization. The memory system can be a shared memory system [8] or a distributed memory system. The distributed memory system can be divided into two branches: the cluster of workstations and the distributed shared memory. The cluster of workstations is composed of many machines where the exchanges between the machines is done via the message passing strategy [2, 10, 14]. The distributed shared memory system uses shared variables for communication [3, 12, 13, 16]. As for the parallelization procedure, radiosity algorithms are not easy to parallelize because the radiosity equation contains form factor calculation or visibility information that put restrictions and dependencies when dividing the data among a distributed memory system. Many parallel methodologies tried to find some local characterizations that can minimize these interdependencies. In [3] the environment is split into sub-environments and a visibility mask is used for the transfer of light. In [4] an idea of group iterative approach was used to split the radiosity solving between a master and slaves.
The purpose of this paper is to parallelize the discrete radiosity method proposed by Chatelier and Malgouyres [1]. This method is different from most radiosity

methods because it is based on discretization of surfaces into voxels and not into patches. In terms of complexity the method is quasi-linear in time and space and has a lower complexity than other methods for large scenes containing a lot of details. The parallelizations presented in this article have been tested on a distributed memory system, a cluster of workstations composed of bi-processors hyperthreaded Xeon Pentium processors.
The paper is organized as follows. Section 2 explains the sequential algorithm and its complexity. Section 3 presents the distribution of tasks which includes two parallelizations, the local parallelization and the distribution of the computation. Section 4 details the data distribution. Finally Section 5 states some conclusions and perspectives.

2 Sequential Algorithm

2.1 Discretization of the Radiosity Equation

Radiosity is defined as the total power of light leaving a point. The continuous radiosity equation has been discretized in [6]. The voxel-based radiosity equation is the following:

$$B(x) = E(x) + \rho_d(x) \sum_{\overrightarrow{\sigma} \in D} B(V(x, \overrightarrow{\sigma})) \frac{\cos\theta(x, V(x, \overrightarrow{\sigma}))}{\pi} \hat{A}(\overrightarrow{\sigma}) \qquad (1)$$

This equation shows that the total power of light $B(x)$ leaving a voxel x depends on two terms, the first is the proper emittance of this voxel as a light source (the $E(x)$ term), and the second is some re-emission of the light it receives from its environment (the sum). The term $B(\cdot)$ that is present in both sides of the equality, reflects interdependence between a point and its environment ; it does not consider any outgoing direction, so it supposes that each point emits light uniformly in every direction (diffuse hypothesis). The factor $\rho_d(x)$ indicates that a point re-emits only a fraction of the light that it receives. D is a set of discrete directions in space. $V(x, \overrightarrow{\sigma})$ is a visibility function, returning the first point y seen from x in the direction of $\overrightarrow{\sigma}$. The term $\hat{A}(\overrightarrow{\sigma})$ is the fraction of a solid angle associated to the direction $\overrightarrow{\sigma}$ it quantifies how much of an object is seen from a point, we call it a *direction factor*. The $\cos\theta(x, V(x, \overrightarrow{\sigma}))$ expresses that incident light is more effective when it comes perpendicularly to the surface. Finally the π factor is a normalization term deriving from radiance considerations.
This equation is usually solved using Gauss-Seidel relaxation requiring thus many iterations to obtain a convergence to the correct radiosity solution [15].

2.2 Computing Visibility Using Discrete Lines

The space can be partitioned into parallel 3D discrete lines, along a given direction. It means that a voxel belongs to one and only one of these lines which can be easily computed. In [1] Chatelier and Malgouyres relied on this space partition concept, that can allow with some constraints, to reflect the visibility

between the voxels. So, given a voxel $(x, y, z) \in \mathrm{Z}^3$, this voxel belongs to the discrete line $L_{i,j}$ where (i, j) are two integers calculated from both the coordinates of the voxel and the directing vector of the direction. The set of all $L_{i,j}$'s represents the space partition into 3D discrete lines.
An important information in the radiosity equation is the visibility factor, reflecting the visibility between the voxels. Given a direction, a voxel belongs to one and only one line characterized by a couple of integers (i, j). Putting each voxel in its line (i, j) does not ensure voxels visibility ; to ensure the correct visibility between the voxels, a possible solution consists in doing a precomputation step that sorts the voxels according to different lexicographic orders ; usually eight lexicographic orders are needed but four are sufficient because of symmetries.

2.3 Algorithm and Complexity

The sequential algorithm of the radiosity comprises the following steps:

1. Discretize the scene into voxels.
2. Sort the voxels according to the four lexicographic orders.
3. Choose the set of directions.

For each iteration, traverse all the directions and for each direction the following steps are done:

1. Choose from the four pre-computed lexicographic orders the one that is suitable depending on the directing vector of the direction.
2. Traverse the voxels and put each one in the list (i, j) to which it belongs.
3. Propagate light between successive voxels of the same list.

The sequential algorithm requires 2 parameters, the number of iterations "I" and the number of directions "D". If N is the number of voxels of the scene, the time complexity is: $4 \times O(NlogN) + I \times D \times (O(N) + O(N))$. The term $4 \times O(NlogN)$ represents the four lexicographic sorting of the voxels. It is negligible, because the four sorting are pre-computed once. The term $I \times D \times (O(N) + O(N))$ reflects the dispatch of the voxels in their lists and the propagation of the light in the lists. Note that "I" is a small constant. "D" is a constant that is independent of the number of voxels ; it depends on the intensity of the light sources. It is increased to avoid aliasing effects when the sources are small but having high intensities. As for the space complexity, we just need to store the voxels in memory. More details about the complexity can be found in [1].

3 Distribution of the Tasks

3.1 Local Parallelization

The first parallelization is local and uses threads. Our radiosity method contains two major computations that are expensive for large scenes and that can be locally parallelized using threads. These computations are the dispatching of the voxels and the propagation of the light in the lists.

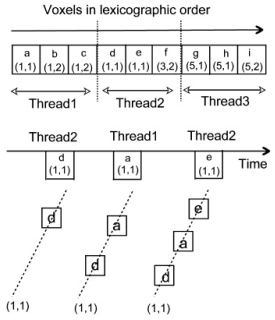

Fig. 1. Distributing the voxels in their lists using threads. There is a risk of loosing the correct lexicographic order

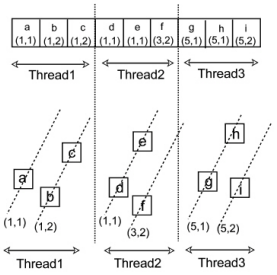

Fig. 2. Distributing the voxels in their lists using threads. Distinct data structures are created for each thread

Multithreaded Voxels Dispatch. The distribution of the voxels in their lists consists in traversing the voxels in the lexicographic order and putting each voxel in the list (i, j) it belongs to. The partition of this work among the threads consists in giving each thread an equal consecutive part of the voxels ordered in the lexicographic order (see Fig. 1). The threads then start the distribution of the voxels in their lists. If one data structure is created, a problem of priority will appear when the threads begin to work on the same lists. The problem arises from the fact that the distribution should respect the lexicographic order, thus giving priority to one thread on the others. For example for Fig. 1 thread 1 has priority on thread 2 and thread 3. The voxels "a", "d" and "e" belonging to the same list but to different threads may not be in the correct lexicographic order if thread 2 puts its voxel "d" before thread 1 has put its voxels "a". A sorting of the lists in the lexicographic order after the threads finish their work is possible however this sorting has a complexity $O(NlogN)$, N being the number of voxels. This sorting can be avoided by creating distinct data structures for each thread in which it will put its voxels in their lists (Fig. 2).

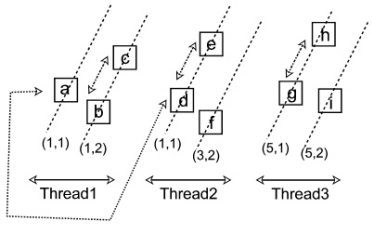

Fig. 3. Propagating light in the lists using threads

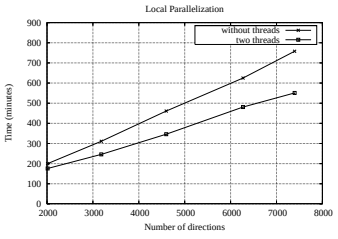

Fig. 4. Local parallelization

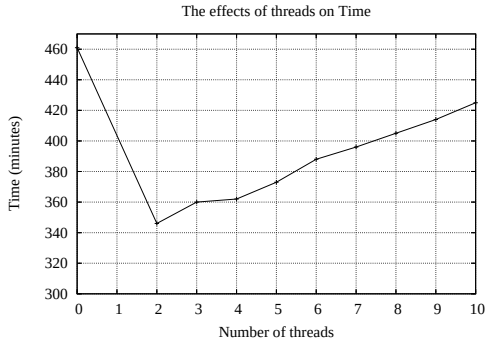

Fig. 5. The effects of increasing the number of threads on time (lower time obtained using two threads)

Multithreaded Light Propagate. The propagation of light in the lists consists in propagating light between consecutive voxels of the same list. The lists created by the threads in Sect. 3.1 are not complete lists since each thread creates its own lists. Gathering the partial lists to propagate light in them is an expensive step in time and it should be avoided. A complete list is the concatenation of partial lists, so each thread can propagate light inside its partial lists of voxels. Then an additional propagate is performed to propagate light between the partial lists (see Fig. 3).

Results. The local parallelization tests are done on a living room scene composed of 17×10^5 voxels. The results with 6 iterations and different number of directions without threads (sequential algorithm) and using two threads are shown on Fig. 4. The time improvement varies between 12 and 30 percent. It is important to note that tests are done with 2 threads because it was noticed that when the number of threads increases time improvement decreases. Fig. 5 tests the effects of increasing the number of threads for the living room scene using ≈ 5000 directions and 6 iterations, the optimal time is obtained with 2 threads. This is normal since the work is tested on bi-processors machines.

3.2 Distribution of the Computation

The discrete radiosity method is linear with respect to the number of directions, so it is possible to divide the summation done on all the directions in Equation (1), into several summations. Each machine of the cluster of workstations will take an equal set of directions. This parallelization requires the presence of

all the voxels on the machines, so voxels should be duplicated on the machines. It offers two major advantages:

- It minimizes the communication between the machines via the message passing interface ; the exchanges are done at the end of each iteration when all the machines have accomplished the radiosity computation for the set of directions given to them. These exchanges consist in adding the radiosity value gained by each voxel duplicated on the machines.
- It offers the ability of using threads to increase the parallelization.

Figure 6 shows the results of this parallelization for the living room scene with 6 iterations and 2000 directions. The time has been almost divided by the number of machines when no threads are used and it decreases even more with the use of threads. The efficiency of this parallelization can reach 98 percent. This parallelization showed good execution times but its main limitation is that it cannot be used for large scenes that do not fit in the RAM.

4 Data Distribution

The local parallelization as well as the distribution of the computation improve the time, however they do not distribute the voxels. For complex scenes that do not fit into the main memory, it is important to exploit a parallelization capable of distributing the voxels together while minimizing the time. In this section, a complete parallelization is proposed.

4.1 Principle

The data distribution dispatches the lists of voxels. Distributing lists of voxels has not the same implication as distributing the voxels. It guarantees having complete lists on one machine i.e. a set of all the voxels included in the lists. Having complete lists on each machine offers the advantage of propagating light in the lists belonging to one machine without having to exchange radiosities between voxels of another machine.
The distribution of the lists among the machines is done by assigning to each machine a distinct set of (i, j). However the (i, j) of each voxel are not constants, they change with the direction. This may splits the complete lists into non-complete ones necessitating some transmission of voxels between the machines to rebuild complete lists. Figure 7 illustrates the method principle. When changing the direction, the lists present on the machines may not be complete: for example the list number (6,7) is present on 2 different machines which requires the exchange of voxels to build one complete list.

The major steps of this parallelization are detailed in next sections and can be summarized as follows:

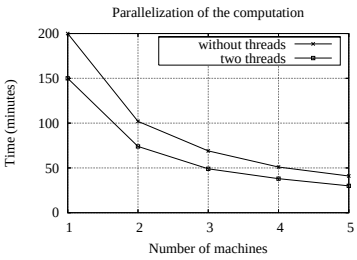

Fig. 6. Distribution of the computation

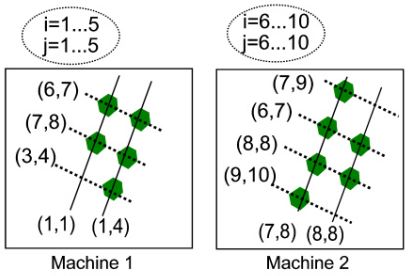

Fig. 7. The lists distribution on two different machines

For each iteration, traverse all the directions and for each direction:

1. Find (iteration equal to 1) or load (iteration different than 1) from memory the suitable list distribution that guarantees the presence of complete lists on each machine and establishes as well a load balancing.
2. Depending on the (i, j) of each voxel decide if it will stay or leave the machine, if it will leave fill it in the corresponding data structure.
3. Exchange data i.e. send and receive implicated voxels.
4. Merge the received voxels with the voxels already present on the machine. The merge is necessary for keeping the voxels sorted in their lexicographic order (see Section 4.4)
5. Dispatch the voxels in their lists.
6. Propagate light in the lists.

4.2 Sorting the Directions

The Lexicographic Order. When changing the direction, we may need to change the lexicographic order that is used to sort the lists. In the sequential algorithm, a precomputation step sorting the voxels according to the four different lexicographic orders was done [1]. When parallelizing the lists, voxels are exchanged between the machines, so this precomputation step can not be done. To solve this problem, a sorting of the direction according to their directing vector (a, b, c) is done ; we can distinguish eight possibilities according to different signs of a, b, and c. For each of these eight sorting will correspond a set of directions satisfying the sign of the directing vector and leading thus to a particular lexicographic order. This will maintain the lexicographic order stable for many directions and will lead to just eight changes in the lexicographic order during an iteration $((x \uparrow, y \uparrow, z \uparrow), (x \uparrow, y \uparrow, z \downarrow), (x \uparrow, y \downarrow, z \uparrow), (x \uparrow, y \downarrow, z \downarrow), (x \downarrow, y \downarrow, z \downarrow), (x \downarrow, y \downarrow, z \uparrow), (x \downarrow, y \uparrow, z \downarrow), (x \downarrow, y \uparrow, z \uparrow))$.

Close Directions. Minimizing the exchanges of voxels between the machines is an interesting aspect that could minimize the rate of exchange as well as the time wasted for sending and gathering data to or from other machines. It has been noticed that close direction would generate small changes in the value of (i, j) which will insure that most of the voxels stays in the machine and only a small amount will leave to another machine. Sorting the directions in such a way they are close to each other can be done by a special traversing of the discrete sphere as shown on the Fig. 8.

Final Sorting. Two sorting for the directions were needed. The best way to achieve these two sorting can be done by traversing every 1/8 quadrant of the discrete sphere using the traversing mentioned in Fig. 8.

4.3 Filling Data and Load Balancing

Filling Data. At every direction, each machine should traverse all of its voxels and calculate the new value of (i, j) (space partition to discrete lines) of each of them to decide whether it should be kept or send. The voxels to be sent are stored in a data structure labelled with respect to the destination they should go to facilitate the filling and the sending steps. The lexicographic order of the voxels is maintained in order to avoid resorting at the reception.

Load Balancing. The exchanges of voxels between the machines at each direction may cause an unbalance in the number of voxels between the machines. To achieve good performance and avoid having one machine waiting for the others to finish their work, it was important to find a suitable lists distribution that can establish load balancing by giving to each machines the appropriate interval $((i_1..i_n), (j_1...j_n))$ of lists to be treated. A static distribution was not efficient since the distribution changes with the direction so a dynamic load balancing was needed. The following steps are done by each cluster machine to obtain the appropriate lists distribution:

- Count the number of voxels belonging to each list in one machines ; lists are not build yet however the number of voxels in each list can be calculated by calculating the (i, j) of each voxel.
- Exchange the voxels number and do the summation between the same (i, j) lists on different machines so that each machine will have a global (i, j) distribution of the whole scene.
- Find the (i, j) distribution that distribute lists among the machine giving each machine almost the same number of voxels ; for this the elastic algorithm [5] is done. This algorithm finds the suitable distribution depending on the number of processors. It is achieved by summing rows to find the suitable row distribution, then summing column to find the column distribution.

It should be stated that it is desirable to have a load balance at every direction however doing it is expensive in terms of time because counting the number

of voxels in each list for large scenes is expensive. Exchanging these numbers between the machines to do the summation is also expensive. One interesting solution equivalent to doing the load balance at every direction consists in taking advantage of having the same directions in the same order for all the iterations. So it is possible to do the load balance for all the directions for the first iteration, memorize the distribution found for each direction then for the rest of iterations reload the correct distribution from the memory. Keeping the distribution in memory is not expensive because it maintains $(numberMachines)^{3/2}$ integers at every directions. Figure 10 shows the importance of increasing the frequency of doing the load balancing. Tests are done for the cabin room composed of 3 millions voxels, using four machines, $\approx$ 5000 directions and 6 iterations. It is clear that the best time is obtained at the value 100 i.e. when doing the load balance at every direction.

4.4 Exchanging and Merging Data

The exchange and the merge are two important operations that should handle the sent and received voxels according to the data structures of each machine. The exchange consists in sending and receiving the set of voxels between the machines just after the filling operation terminates. The Merge operation merges the voxels owned by the machine (ordered in their lexicographic order) together with each of the packs received (also ordered in their lexicographic order) from other machines ; it is expensive, however most of the time the exchanges are of small size (10 percent of voxels owned by the machines) and between limited number of machines (the direction are close, the couple (i, j) varies slowly), so it becomes less expensive.

4.5 Results

The data distribution tests were examined for a cabin room composed of 3 millions voxels. The tests are done for $\approx$ 5000 directions and 6 iterations. The speed-up are shown on Fig. 9. A speed-up of 2.7 is achieved with 8 machines. In the sequential algorithm we had just two important times, the dispatch and the propagate of light in the lists, however in the parallel version we still have the dispatch propagate time that is divided by the number of machines and in addition, we have four operations that are added by the parallelization (Filling, Load balancing, Exchange, Merge). The time study for these four operations is viewed in Fig. 11. It can be seen that the filling process is an expensive operation in time but it decreases with the number of machines and its time is divided by this number. The load balancing time does not vary a lot and it is almost constant with respect to the number of machines. As for the exchange time, it is negligible when compared to other times since only small percentage of voxels (10 percent) is usually exchanged between the machines. Finally the merging time depends on the quantity of exchanges between the machines and it does not vary too much.

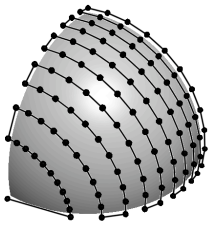

Fig. 8. Directions ordering on 1/8 of the sphere to minimize the exchange of voxels

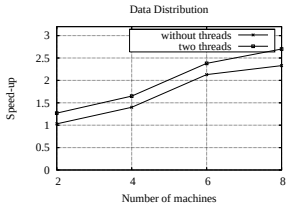

Fig. 9. Data distribution speed-up for different number of machines

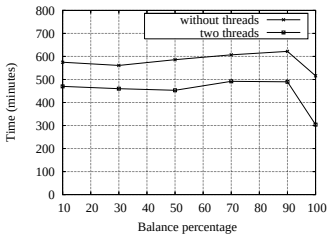

Fig. 10. Effect of increasing the frequency of doing load balance on time

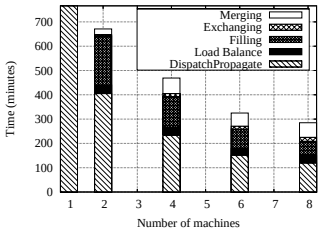

Fig. 11. Data distribution: Time distribution for each operation

5 Conclusion and Perspectives

In this paper we have examined three parallelizations. The first parallelization is just a local one that can help to minimize the time of the other parallelizations. The parallelization of the computation has shown good results, but it is limited to scenes that fit in the main memory. The data distribution has improved the sequential time ; it has a low speed-up because it demanded the construction of complete lists on each machine and necessitated though many additional operations that are added to the sequential operations. Despite its low speed-up, it has distributed the data, which is important when dealing with large scenes that do not fit in memory. As for the perspectives and further works, many tests are also needed to minimize the time of the data distribution. We also intend to apply the data distribution proposed in this paper with the computational distribution to exploit different levels of parallelizations ; this is expected to increase the speed-up and to minimize the time even further. Preliminary experiments have confirmed those hypotheses.

Acknowledgments

The authors wish to acknowledge the support of Conseil Regional d'Auvergne within the framework of the Auvergrid project.

References

1. Chatelier, P., Malgouyres, R.: A Low Complexity Discrete Radiosity Method. Discrete Geometry for Computer Imagery. **3429** (2005) 392–403
2. Arnaldi, B., Priol, T., Renambot, L., Pueyo, X.: Visibility Masks for solving Complex Radiosity Computations on Mutliprocessors. Parallel Computing **23(7)** (1988) 887–897
3. Renambot, L., Arnaldi, B., Priol, T., Pueyo, X.: Towards Efficient Parallel Radiosity for DSM-based Parallel Computers Using Virtual Interfaces. Proceedings of the IEEE symposium on Parallel rendering (1993) 76–86
4. Funkhouser, TA.: Coarse-grained parallelism for hierarchical radiosity using group iterative methods. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (1996) 343–352
5. Feschet, F., Miguet, S., Perroton, L.: Parlist: a parallel data structure for dynamic load balancing. Journal of Parallel and Distributed Computing **51** (1998) 114–135
6. Malgouyres, R.: A Discrete Radiosity Method. Discrete Geometry for Computer Imagery **2301** (2002) 428–438
7. Smith, B., Arvo, J., Donald, G.: A clustering algorithm for radiosity in complex environments. Proceedings of the 21st annual conference on Computer graphics and interactive techniques **28** (1994) 435–442
8. Podehl, A., Rauber, T., Runger, G.: A Shared-Memory Implementation of the Hierarchical Radiosity Method. Theoretical Computer Science **196** (1998) 215–240
9. Bouatouch, K., Mebard, D., Priol, T.: Parallel Radiosity Using a Shared Virtual Memory. Proceedings of Advanced Techniques in Animation, Rendering and Visualization (1993) 71–83
10. Sturzlinger, W., Wild, C.: Parallel Visibility Calculations for Radiosity. Proceedings of the Third Joint International Conference on Vector and Parallel Processing: Parallel Processing (1994) 405–413
11. Yu, Y., Oscar, H., Yang, T.: Parallel progressive radiosity with adaptive meshing. Journal of Parallel and Distributed Computing **42(1)** (1997) 30–41
12. BarLev, A., Itzkovitz, A., Raviv, A., Schuster, A.: Parallel Vertex-To-Vertex Radiosity on a Distributed Shared Memory System. Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel. J. Diff. Eq. **1457** (1998) 238–250
13. Sillion, F., Hasenfratz, J.M.: Efficient Parallel Refinement for Hierarchial Radiosity on a DSM Computer. Third Eurographics Workshop on Parallel Graphics and Visualisation (2000) 61–74
14. Guitton, P., Roman, J., Subrenat, G.: Implementation Results and Analysis of a Parallel Progressive Radiosity. Proceedings of the IEEE symposium on Parallel rendering (1995) 31–38
15. Cohen, M.F., Greenberg, D.F.: The hemi-cube: a radiosity solution for complex environments. SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques **19(3)** (1985) 31–40
16. Martnez, R., Sbert, M., Szirmay-Kalosv, L.: Parallel Multipath with Bundles of Lines. Third Eurographics Workshop on Parallel Graphics and Visualisation (2000)