A Global Visibility Classifier Based on a Multi-DSP-System

Authors: R. Hranitzky, N. Thurner

ESIEE, Paris
September 1996
SPRA338

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Tables

A Global Visibility Classifier Based on a Multi-DSP-System

Abstract

We present a classification algorithm for determination of global visibility relations. Results allow us to characterize visibility between any two points in 3D visibility space. Parallelism of computations is detected by data dependence analysis and leads to a parallel program. An experimental implementation of the classifier on a multi-DSP system (6 TMS320C40) is presented. Finally we discuss efficiency, scalability and dynamical load balancing of parallel computations.

This document was an entry in the 199x DSP Solutions Challenge, an annual contest organized by TI to encourage students from around the world to find innovative ways to use DSPs. For more information on the TI DSP Solutions Challenge, see TI's World Wide Web site at www.ti.com.

Keywords

Global Visibility Classification

Parallel Processing

Multi-DSP TMS320C40

Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Introduction

In our work we describe parallel aspects of visibility computations between polygons in a discrete visibility space. Such classifications are necessary for various kinds of virtual reality applications, like illumination computation methods (e.g. Ray Tracing, Radiosity, etc.) in computer graphics [1]. Classical methods use output device dependent methods (e.g. hidden surface removal) and are restricted to special cases [12] [3].

More general methods work on mathematical representations of visibility in 3D-space (some definitions can be found in the section *Classification of Global Visibility Relations*). The algorithms are designed to be hierarchical and modular [7]. In operation they are conservative, but exact: all 'visible' and 'invisible' classifications can be proved to be true (see the section *Classification Algorithm*) [10].

Global visibility determination is the most expensive geometric operation in image synthesis. With sequential computing this process is restricted to scenes of low visual complexity. Results of analysis state that digital signal processors *(DSPs)* are well suited for visibility calculations (see the section *Algorithm Analysis*).

Furthermore parallelism analysis encouraged us to design a parallel classifier, which runs on multiple DSPs. This greatly reduces execution time and allows to engage more complex scenes. A similar parallel implementation on a nCube2S-system can be found in (9]. Described experimental measurements show high speedups even for an increasing number of DSPs. Of course, workload has to be considerably large to compensate for low speed of inter-DSP communications (see the section *Results*). The presented parallel application is highly scaleable and is also convenient for massively parallel systems (see the section *Scalability*).

Classification of Global Visibility Relations

We describe an algorithm for classification of global visibility relations, which operates on the entire visibility space. The hierarchical classification algorithm (see the *section Classification Algorithm*) is based on a set of polygons being a discrete representation of the visibility space. Parallel aspects are discussed in the section *Parallelism*.

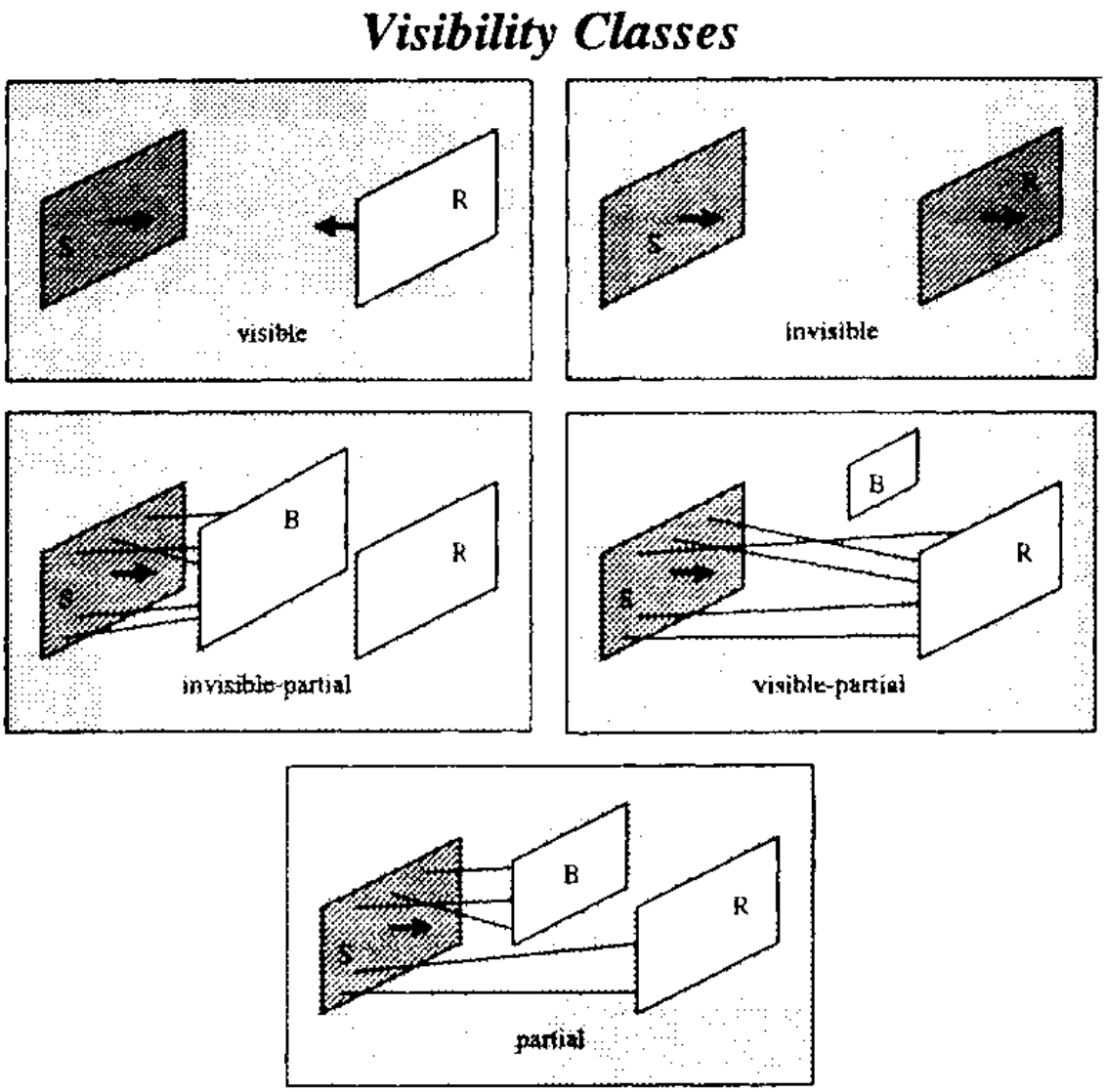
Visibility Space Model

First of all we give some definitions and restrictions of our visibility-space model. We define our visibility space to be a finite set of convex polygons residing in a discrete 3D-coordinate system.

Polygons The geometrical properties of polygons are defined by an arbitrary number of vertices, which span the polygon's area *A*. We assign a 'face' to each polygon and thus define a direction of visibility. Normal vectors indicate front faces. To simplify discussion we use rectangular polygons in all examples throughout this paper (see e.g. figure 5).

Visibility Classes and Attributes A local visibility pair is a set of two polygons {S,R}. This pair describes the visibility from S to *R* and vice versa. All local visibility pairs are members of the global visibility set V. The goal of the classifier is to assign an ordered pair of visibility attributes to each member of *V*. We define 5 classes of different visibility attributes. *R* is 'visible' to S, if every point of A_R can be seen from every point of A_S. Similarly for 'invisible' and 'partial'. That is depicted in figure 1. The classes 'visible-partial' and 'invisible-partial' are assigned, if the algorithm is not able to make clear decisions (see paragraph 'Conservative Triage' in the section *Classification Algorithm*).

Figure 1. Visibility classes and attributes



Classification Algorithm

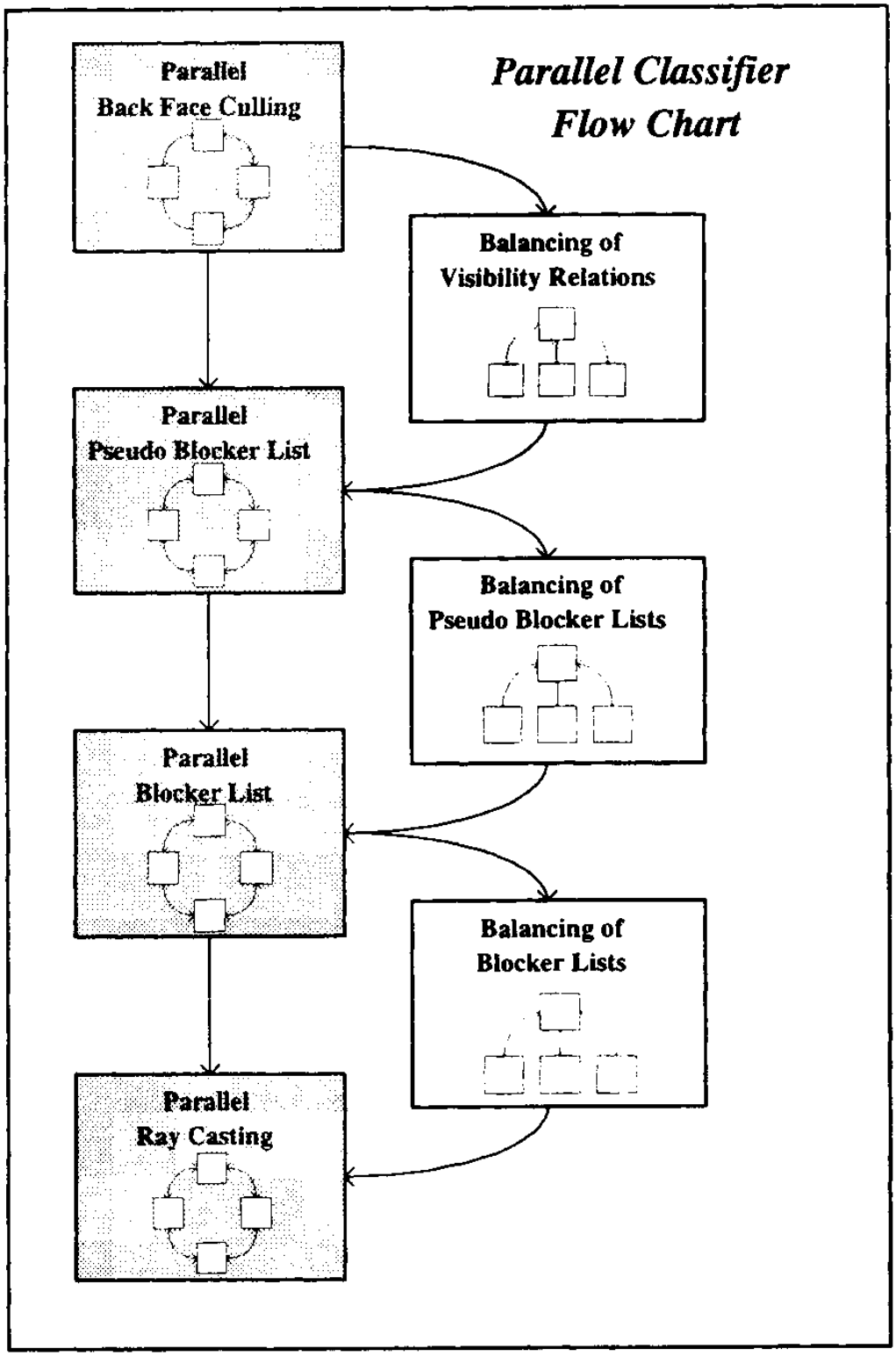
In accordance to [10] we base our algorithm on three simple ideas:

Visibility preprocessing The exact determination of visibility relations is a very time-consuming process. It can be accelerated by preprocessing the database. We have assigned 'faces' to all polygons and thus we can determine those pairs quickly, which consist of facing away polygons. They are 'invisible'. This procedure is called Back Face Culling [13].

Incremental visibility maintenance Several more and less complex operations are necessary to classify remaining pairs. To further accelerate runtime, we sort computations by increasing complexity. That leads to a modular design, which is depicted in figure 2. We determine pseudoblocking and blocking polygons in the second and third modules using increasing numerical costs. All relations without blocking polygons *B* can be classified 'visible' afterwards.

Next we perform Ray Casting, trace rays between {S,R} and try to intersect them with the related set of blocking polygons *{B} [5].* The number of rays can be set initially. If all rays are being blocked, the class 'invisible-partial' is assigned, because from the algorithms point of view this relation is invisible, but could eventually become partial, if further rays are being traced. Similarly the class 'visible-partial' is assigned if none of the rays is being blocked (see figure 1).

Figure 2. Flow chart of parallel classifier



Conservative Triage We use conservative triage to avoid the combinatorial complexity of exact visibility classification. The classification is conservative in that all interactions classified as visible or invisible are correct. It is acceptable for the classification to return ’partial’ instead of ’invisible’ or ’visible’ respectively. The user may reassign invisible-partial or visiblepartial relations to the ’invisible’ or ’visible’ classes if the number of rays was considerably high.

Algorithm Analysis

We analyze the presented algorithm to determine memory requirements and worst case execution time (see the section *Complexity*). Parallelism is detected using data dependence analysis. The parallel algorithm is a SPMD type (see the section *Parallelism*). To increase parallel performance we add dynamical load balancing strategies, which are described in the section *Dynamical Load Balancing*.

Complexity

An input data set of *N* polygons produces $O(N^2)$ visibility pairs. Each of these pairs has to be tested against *O(N)* polygons to detect a 'blocking' situation. We stress that complexity of time and memory of any sequential algorithm is $O(N^3)$.

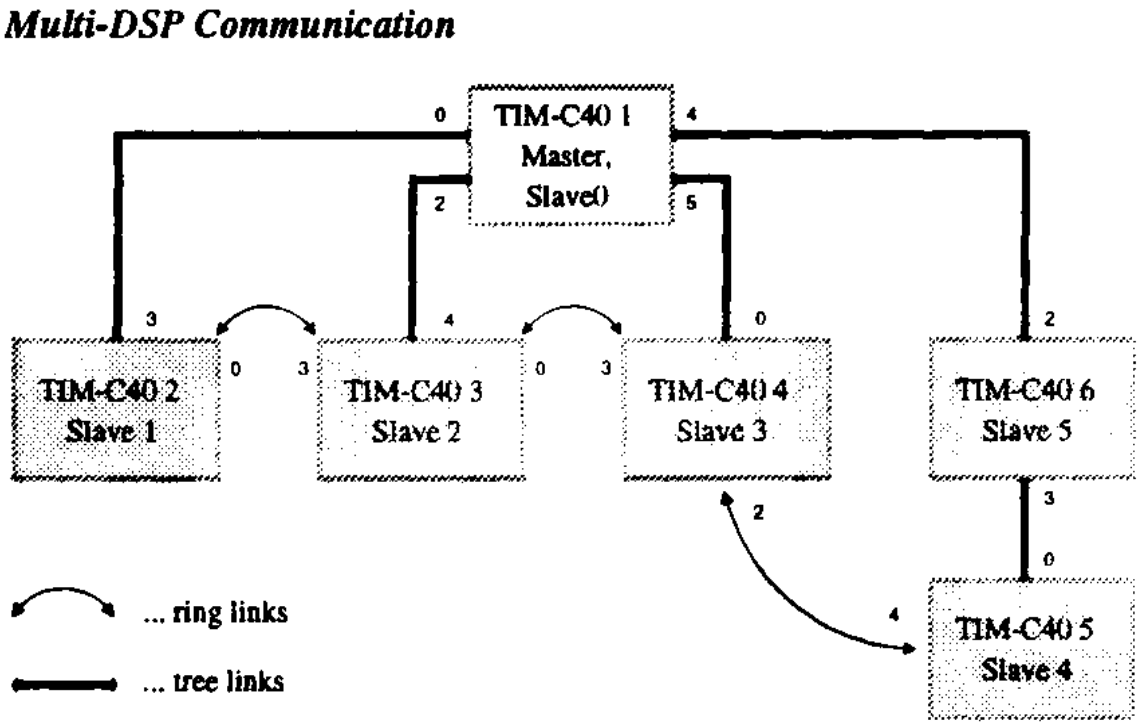
Parallelism

By means of data dependence analysis we can find sets of computations, which can be computed in parallel [4]. So called data dependence graphs (DDGs) present sequential dependencies in a graphical manner and simplify analysis [8]. Results show that each of the modules depicted in figure 2 can be split into independent submodules (not shown). All submodules can be computed in parallel, but require probably access to the entire input data set *N.* The final parallel algorithm, an SPMD architecture, is a combination of four output dependent parallel modules (Back Face Culling, Pseudo Blocker List, Blocker List, Ray Casting).

Implementation

We choose a distributed memory multi-DSP system with six nodes from Transtech Parallel Systems for experimental implementation. Each DSP resides on a TIMC40-module (four modules per motherboard) and has access to 8MBytes EDRAM of memory. The designed parallel algorithm has been implemented using the realtime-operating system Virtuoso [11]. We distingiush between the working processes called 'Slaves' and the coordinating process called 'Master', which performs also dynamical load balancing (see figure 3).

Figure 3. Multi-DSP communication scheme



Partitioning

Initially the set of *N* polygons is equally distributed among the *r* DSPs. Thus each DSP holds *N/r* polygons in its local memory (optimal memory efficiency). We want to stress that memory efficiency can be traded off against overall execution time by introducing replicas. That reduces much communication between the DSPs.

Communication

The described partitioning scheme requires possibly communication between all DSPs. It should be mentioned that these dependencies are introduced artificially due to the distribution of data structures (compare to the section *Parallelism*). Only 1/*r* of the required polygons are available locally. A hybrid network consisting of a ring-structure and an n-ary tree structure is used to solve nonlocal references (see figure 3, $r = 6$). Complexity of communication is *O(N).*

Parallel Back Face Culling Each of the *r* DSPs is able to calculate $O(N^2/r^2)$ visibility relations without having to communicate. For the remaining. $O(N^2/r)$ components each DSP requires to receive $N - N/r$ polygons from the other DSPs. Using the directed ring-network, it requires $(r - 1)$ shift-lefts of N/r polygons between any two DSPs respectively.

Parallel Blocker List The generation of a blocker list requires the same amount of communication. In contrast, the amount of computations with locally available polygons *is* $O(N^3/r^3)$.

Parallel Ray Casting Once blocker lists are established, rays from source polygons S to receiver polygons *R* are casted and intersected with the blocking polygons *B.* Those polygons, which are not available locally (i.e. $O(N - N/r)$ units) are transferred between the DSPs similarly to previous paragraphs.

Dynamical Load Balancing

Load balancing is required whenever waiting times between synchronizing DSPs tend to influence overall execution time. If individual DSPs have significantly higher workload, we may redistribute some work to 'underloaded' DSPs. There are two major sources of imbalance:

- A schedule, which favours certain DSPs.
- Input data sets, which produce unpredictable workloads.

The parallel dynamical load balancer has a hierarchical structure (see figure 2). Although individual blocks of the algorithm are working on different workloads, the balancing strategy remains the same and is based on the well known NP-complete bin-packing problem. Instead of optimal strategies, we use simple and fast heuristics: The kernel of the balancer resides on the master-DSP and retrieves load information from all slave-DSPs. This load information is made up of visibility sets {S,R,$B_{\#}$, *Owner*}, where $B_{\#}$ is the pair's weight (i.e. the number of blocking polygons) and *Owner* is the loaded DSP. The kernel has to deal with the constraint of minimizing communication between all DSPs. A pseudo code is depicted in figure 4.

Figure 4. Pseudo code of dynamical load balancer

```
function Balance_Load(setof{S, R, B#, Owner})
 compute sum(Load)
 do forall {S, R}
  if DSP_i = owner(S, R) and Load_i = min(Owner)
   if Load_i + B# < avg(Load)
    assign {S, R} to DSP_i
  elseif DSP_i = owner(S) and Load_i = min(Owner)
   if Load_i + B# < avg(Load)
    assign {S, R} to DSP_i
  elseif DSP_i = owner(R) and Load_i = min(Owner)
   if Load_i + B# < avg(Load)
    assign {S, R} to DSP_i
  else assign {S, R} to DSP with min(Load)
 enddo
 return setof{S, R, B#, NewOwner}
endfunction.
```

Results

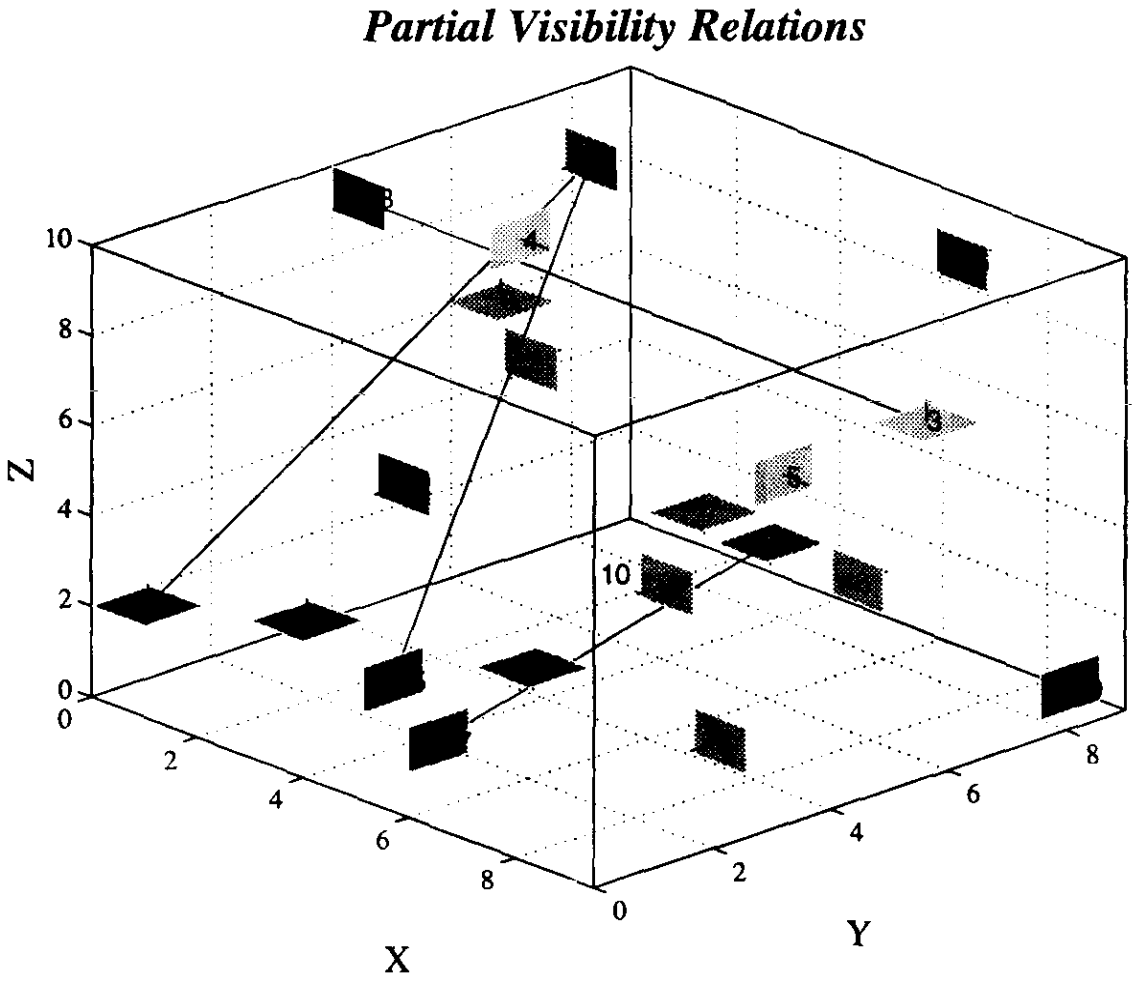
The parallel performance of the presented classifier is determined by several experimental measurements. During all experiments we measure execution time, which is directly influenced by the algorithm, by the programming system, by the multi-DSP-system, etc. The first sections *Experiments* and *Time, Speedup and Efficiency* define constraints of executed experiments and present their results.

Experiments

For our experiments we use a well defined set of polygons. A computer program generates an artificial room by distributing N polygons randomly throughout a three dimensional coordinate system. The labyrinth rooms allow us to simulate very dense rooms as well as very sparse rooms. According to the section *Visibility Space Model* we can use sets of very high densities to produce large lists of blocking polygons. That allows us to measure worst case runtimes of the parallel program near $O(N^3)$.

On the other side it has been shown that real rooms are normally of the sparse, inhomogeneous type [1]. These conditions can be simulated too. A simple test set with N = 20 is depicted in figure 5. Lines normal to surfaces are normal vectors and show orientations of polygons. Lines between polygons show partial visibility relations as they were determined by the parallel classifier (see the section *Visibility Results*).

Figure 5. Example Input Data Set. Lines indicate partial visibilities



Time, Speedup and Efficiency

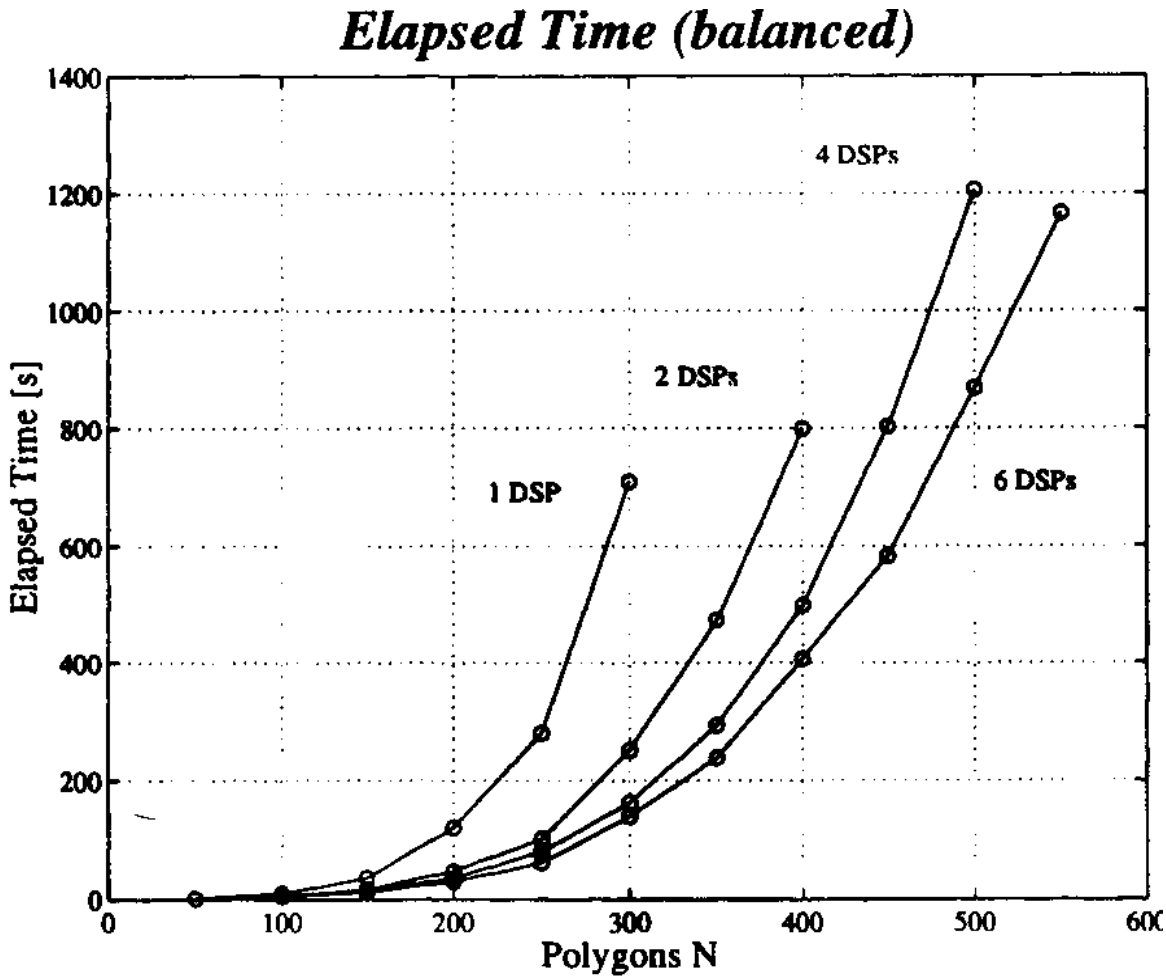
We have measured elapsed (wall clock) time, which accounts not only for the computational work, but also for the communicational work and task switching due to waiting for resources. Elapsed time states how long one has to wait for a complete solution [2]. All presented data for different numbers of DSPs have been evaluated by averaging the elapsed time of several program runs to ensure that the measured data is not some statistical fluke.

The elapsed time of the multi-DSP solution can be brought into relation to the single-DSP solution and results in the speedups and efficiencies from table 1. It should be noticed that although the single-DSP solution plays the role of a sequential basis for speedup and efficiency measurements, it is made up of two tasks (master and slave0) sharing the CPU concurrently. Thus the single-DSP solution is equivalent to the parallel program running on one DSP.

We assumed dynamical load balancing for all multi-DSP solutions presented in table 1. Figure 6 shows clearly a time complexity of $0(n^3)$ (see the section *Complexity*). Distances between lines become larger as the number of polygons increases. We conclude that speedup can be increased by increasing *r* and *N.* That indicates good scalability.

Scalability

Figure 6. Multi-DSP elapsed time (balanced).



To determine scalability of our parallel system, which is the ability to increase speedup as the number of DSPs increases, we evaluate the isoefficiency function [14]. The function relates problem size W (i.e. sequential time complexity $O(N^3)$) to the number of processing elements r [6].

Evaluation can be done approximately by using measured data from table 1 or more exactly by mathematical analysis. We have determined an isoefficiency function of $W^{2/3} = O(r)$. It states that r can increase linearly with N^2 to result in constant efficiency. We conclude that this multi-DSP application is highly scaleable.

Table 1. Multi-DSP performance

Experimental Measurements balanced load			
$N = 100$			
DSPs	**Time**	**Speedup**	**Efficiency**
1	10.476″	1	1
2	05.891″	1.778	0.889
4	05.501″	1.904	0.476
6	04.945″	2.119	0.353
$N = 200$			
DSPs	**Time**	**Speedup**	**Efficiency**
1	2′0.598″	1	1
2	47.337″	2.548	1.274
4	35.597″	3.388	0.847
6	29.518″	4.086	0.681
$N = 300$			
DSPs	**Time**	**Speedup**	**Efficiency**
1	11′50.365″	1	1
2	4′10.985″	2.830	1.415
4	2′42.930″	4.360	1.090
6	2′19.466″	5.093	0.849

Dynamical Load Balancing

As mentioned in the section *Dynamical Load Balancing*, there are two major sources of imbalances.

Code Imbalance Increased elapsed time due to imbalance, which is introduced by the algorithm can be seen in figure 7 and table 2. We find that especially for a small number of DSPs the algorithm introduces much load imbalances. The initial calculation of 'Back Faces' makes use only of *r*/2 DSPs in the last iteration. These DSPs are significantly higher loaded. The effect is less significant for an increasing number of DSPs. Furthermore imbalances increase with increasing load and we conclude that this multi-DSP application requires dynamical load balancing, especially if *r* is small or *N* is large.

Figure 7. Multi-DSP elapsed time (unbalanced load).

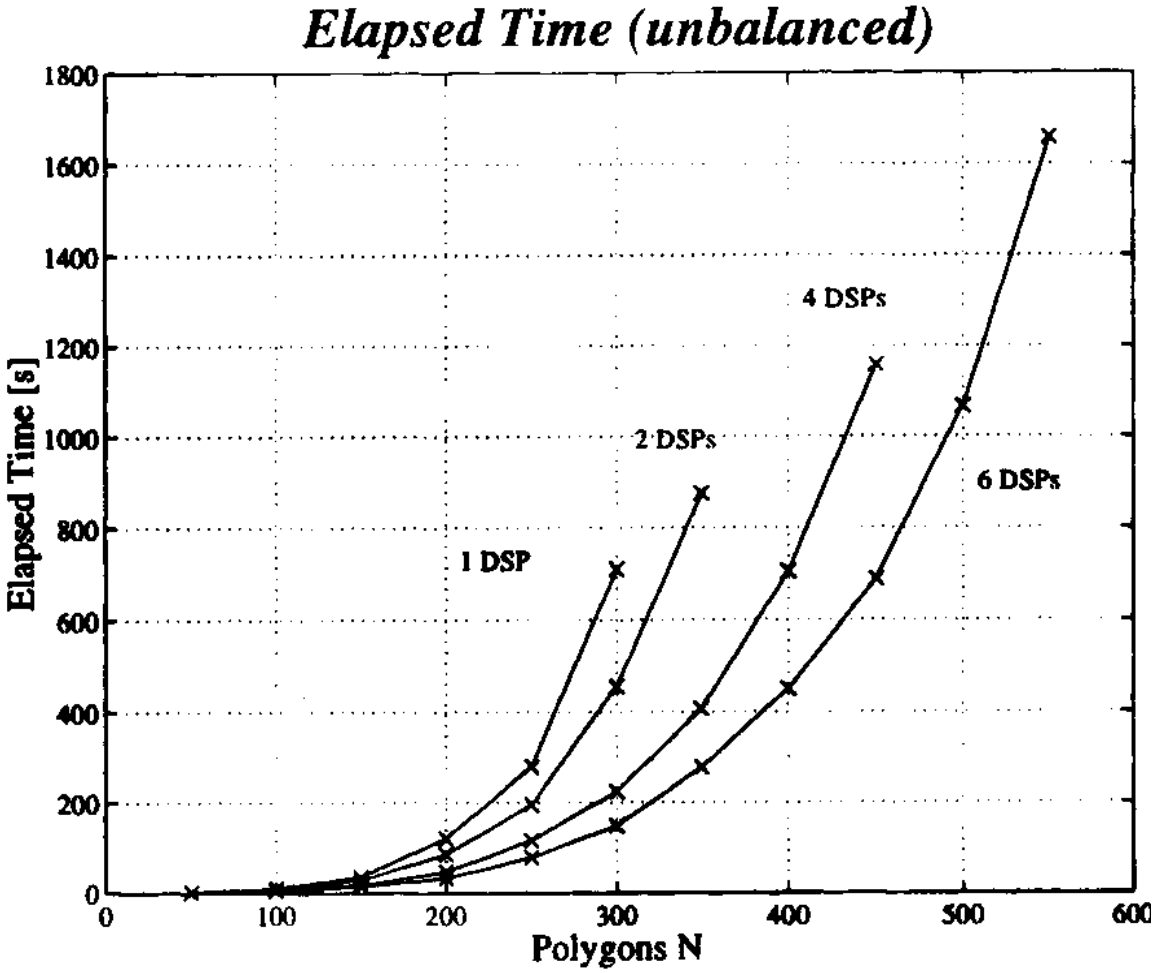


Table 2. Quality of parallel dynamical load balancer.

Experimental Measurements		
$N = 100$		
DSPs	**Time Reduction**	**Balancing Time**
2	40%	2.4%
4	7%	3.6%
6	2%	3.8%
$N = 300$		
DSPs	**Time Reduction**	**Balancing Time**
2	82%	6%
4	37%	5.8%
6	8%	4%
$N = 500$		
DSPs	**Time Reduction**	**Balancing Time**
2	–	–
4	–	3.2%
6	23%	3.7%

Data Imbalance We have to deal with imbalances introduced by the input data set. Obviously we could create almost any state of balance by designing some odd test sets. Several experiments showed that dependency of balance on input data is much more significant than dependency on algorithmical flow. From elapsed time's point of view, timing requirements from table 2 can be seen as 'best cases' if no load balancer is used.

We state that due to an 'open' definition of the input interface, this multi-DSP application requires a load balancer, which rebalances workload at runtime.

The quality of any dynamical load balancing algorithm can be determined by comparing gain of execution speed to time of rebalancing. This is depicted in table 2. We see that the balancer requires 4% of elapsed time on average but shortens runtime widely. Figure 8 shows the balancer's statistical output for the example from figure 5.

Figure 8. Statistical output of parallel dynamical load balancer for the example in figure 5.

Load Balancer Statistics

MASTER: Balancing Relations ...

P0: 4 Polygons, Local Relations 3, Global Relations 6	Assigned 5
P1: 4 Polygons, Local Relations 0, Global Relations 6	Assigned 5
P2: 3 Polygons, Local Relations 0, Global Relations 3	Assigned 4
P3: 3 Polygons, Local Relations 0, Global Relations 1	Assigned 4
P4: 3 Polygons, Local Relations 0, Global Relations 4	Assigned 4
P5: 3 Polygons, Local Relations 1, Global Relations 2	

MASTER: Balancing Pseudo Blockers ...

P0: 10 Local Pseudo Blocker, 11 Global Pseudo Blocker	Assigned 10
P1: 0 Local Pseudo Blocker, 10 Global Pseudo Blocker	Assigned 10
P2: 0 Local Pseudo Blocker, 02 Global Pseudo Blocker	Assigned 13
P3: 0 Local Pseudo Blocker, 03 Global Pseudo Blocker	Assigned 11
P4: 0 Local Pseudo Blocker, 17 Global Pseudo Blocker	Assigned 10
P5: 0 Local Pseudo Blocker, 10 Global Pseudo Blocker	Assigned 09

MASTER: Balancing Blockers ...

P0: 2 Local Blocker, 0 Global Blocker	Assigned 2
P1: 0 Local Blocker, 1 Global Blocker	Assigned 1
P2: 0 Local Blocker, 3 Global Blocker	Assigned 3
P3: 0 Local Blocker, 1 Global Blocker	Assigned 1
P4: 0 Local Blocker, 1 Global Blocker	Assigned 1
P5: 0 Local Blocker, 1 Global Blocker	Assigned 1

Visibility Results

For small sets of polygons results can be presented graphically, while results for large sets are written to a text file. A statistical output for the example in figure 5 is depicted in figure 9.

Figure 9. Statistical output of parallel classifier for example in figure 5.

Classifier Statistics

380 Visibility-Paths:
328 invisible (Back Face Culling)
43 visible
5 visible-partial
0 invisible-partial
4 partial
0 fail

P0: 3 RELATIONS

{1,2}->VISIBLE/VISIBLE
{1,3}->VISIBLE/VISIBLE
{0,1}->VISIBLE-PARTIAL/VISIBLE-PARTIAL

P1: 2 RELATIONS

{4,3}->VISIBLE/VISIBLE
{1,17}->VISIBLE-PARTIAL/VISIBLE

P2: 6 RELATIONS

{10,0}->VISIBLE/VISIBLE
{9,1}->VISIBLE/VISIBLE
{8,18}->VISIBLE/VISIBLE
{4,17}->VISIBLE/VISIBLE
{14,10}->VISIBLE/VISIBLE-PARTIAL
{1,15}->PARTIAL/VISIBLE-PARTIAL

P3: 7 RELATIONS

{13,9}->VISIBLE/VISIBLE
{0,12}->VISIBLE/VISIBLE
{15,12}->VISIBLE/VISIBLE
{1,13}->VISIBLE/VISIBLE
{4,0}->VISIBLE/VISIBLE
{7,14}->VISIBLE/VISIBLE
{3,13}->PARTIAL/VISIBLE

P4: 3 RELATIONS

{14,11}->VISIBLE/VISIBLE
{14,12}->VISIBLE/VISIBLE
{1,14}->PARTIAL/VISIBLE

P5: 5 RELATIONS

{17,18}->VISIBLE/VISIBLE
{17,12}->VISIBLE/VISIBLE
{4,15}->VISIBLE/VISIBLE
{7,0}->VISIBLE/VISIBLE
{17,11}->PARTIAL/VISIBLE

Conclusion

We have presented a multi-DSP based global visibility classifier with complexity $O(N^3/r)$ for calculation of visibility relations in the entire visibility space. Experimental measurements and scalability analysis showed that the presented parallel application is well scaleable, so many further DSPs can be used efficiently and result in a great reduction of overall execution time, especially for large problem sizes. A parallel dynamic load balancer helps to increase speedup and allows better utilization of available memory.

References

1) Larry Aupperle. *Hierarchical Algorithms for Illumination.* Doctoral thesis, Princeton University, Department of Computer Science, 1993.

2) Lawrence A. Crowl. How to Measure, Present, and Compare Parallel Performance. *IEEE Parallel & Distributed Technology,* pages 9-25, Spring 1994.

3) J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics.* Addison Wesley, Reading, USA, 1960.

4) Ian Foster. *Designing and Building Parallel Programs.* Addison Wesley, 1995.

5) Andrew S. Glassner, editor. *An Introduction to Ray Tracing.* Academic Press, San Diego, California, *1989.*

6) Ananth Y. Grama, Gupta Anshul, and Vipin Kumar. Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures. *IEEE Parallel & Distributed Technology,* pages 12-21, August *1993.*

7) Ned Greene, Michael Kass, and Gavin Miller. Hierarchical Z-Buffer Visibility. In *Computer Graphics,* pages *231-234, 1993.*

8) Dan I. Moldovan. *Parallel Processing.* Morgan Kaufmann Publishers, Inc., *1993.*

9) W. Sturzlinger and C. Wild. Parallel Visibility Computations for Parallel Radiosity. In *Winter School of Computer Graphics and CAD,* pages 405-413, University of West Bohemia, Plzen, Czech Republic, *1994.*

10) Seth Teller and Pat Hanrahan. Global Visibility Algorithms for Illumination Computations. In *Computer Graph*ics, pages *239-246, 1993.*

11) Eric Verhulst. Virtuoso: A virtual-single processor Programming System for distributed real-time applications. *Microprocessing and Microprogramming, Euromicro Journal,* pages *103-115, 1994.*

12) J. Warnock. A Hidden Surface Algorithm for Computer Generated Half Tone Pictures. *Technical Report, 1969.*

13) Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques.* Addison Wesley, Reading, USA, *1992.*

14) Albert Y. H. Zomaya, editor. *Parallel & Distributed Computing Handbook.* Computer Engineering. McGraw-Hill, *1996.*