A Meshing Scheme for Real Time Surface Subdivision

E. J. Padrón[1] **M. Amor**[1] **R. Doallo**[1]

M. Bóo[2]

[1] Department of Electronics and Systems, University of A Coruña,
E–15071 A Coruña, Spain. E–mail: {emilioj, margamor, doallo}@des.fi.udc.es

[2] Department of Electronic and Computer Eng., University of Santiago de Compostela,
E–15782 Santiago de Compostela, Spain. E–mail: mboo@dec.usc.es

ABSTRACT

Surface subdivision in real time is highly desirable for computer graphics, geometric modeling, and scientific visualization. In this paper we present a parallelization of the Modified Butterfly algorithm based on the subdivision of the original mesh into small groups. The groups are sorted in decreasing order of number of triangles per group, and the sorted groups are cyclically distributed on the processors in order to balance the load. So as to avoid cracking effects among groups a slight modification of the Modified Butterfly algorithm is used. Finally, we evaluate the algorithm on a SGI Origin 2000 system.

Keywords: surface subdivision, distributed memory multiprocessor, grouping algorithm, parallel implementation, Modified Butterfly

1 INTRODUCTION

Achieving smooth surfaces from coarse meshes is a frequently used operation in computer graphics, geometric modeling, and scientific visualization. The recursive subdivision schemes [Schrö00] are well suited for this purpose because they are easy to implement and computationally efficient. A recursive subdivision algorithm generates a smooth mesh from a coarse mesh as the limit of a sequence of successive refinements with a fixed set of subdivision rules. The evolution of each triangle is conditioned by the first order contiguous triangles. In general, these subdivision techniques can be categorized into two distinct classes: approximating and interpolating schemes.

Approximating schemes for arbitrary topology meshes are typically modifications of spline based schemes. This way, the vertices of the original mesh do not belong to the final computed mesh. A representative approximating scheme for triangle meshes is the Loop algorithm [Loop87]. On the other hand, for interpolating schemes the vertices of the original mesh are also points of the final surfaces.

The most well–known interpolation-based subdivision scheme is the Butterfly algorithm proposed by Dyn et al. [Dyn90]. This algorithm exhibits degeneracies when it is applied to a topologically irregular grid. This problem was overcome with the Modified Butterfly algorithm proposed by Zorin et al. [Zorin97]. Loop and Modified Butterfly are based on the same basic idea in which each edge is subdivided employing its neighbor information.

The main disadvantage of the subdivision schemes is their high memory requirements when finer meshes are considered. The number of mesh vertices and computations grow by a constant factor from iteration to iteration. Processing of a high detailed mesh in real time could be achieved by its partitioning on a set of processors. The smooth meshes calculated on each processor are sent to an application, like a graphics pipeline, to be drawn; then the achievement of high quality images in real time is feasible.

These algorithms present an important drawback from a parallel implementation point of view: computation of each edge implies the utilization of not only the coordinates of the extreme vertices of the edge, but also the corresponding coordinates of the neighbors of these vertices; it is necessary the mesh partitioning in such a way that the communications among processors are minimized. In this paper, we consider the parallelization of surface subdivision algorithms for triangular meshes on a distributed memory multiprocessor. A novel mesh partitioning algorithm to subdivide the coarse mesh into smaller groups is considered. This grouping algorithm [Amor00], [Bóo01] covers the full mesh in an efficient way, produces a balanced number of triangles per group, and reduces the number of bounding triangles per group. Load balancing among processors together with the reduction of the communications are algorithm design goals.

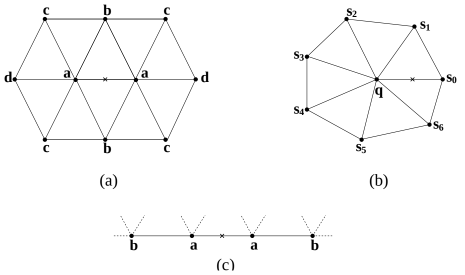


Figure 1: Modified Butterfly Subdivision Masks (the crosses indicate the midpoints of the edges for which a new value is computed) (a) Regular vertices (b) Extraordinary vertices (c) Boundary vertices

This work is organized as follows: in Section 2 we introduce the Modified Butterfly scheme for subdividing triangle meshes; the parallel implementation based on the mesh grouping algorithm is presented in Section 3; experimental results on the SGI Origin 2000 multiprocessor are shown in Section 4; finally, in Section 5 we present the conclusions.

2 THE MODIFIED BUTTERFLY ALGORITHM

The Modified Butterfly [Zorin97] is an interpolating algorithm. The triangulation scheme is refined through splitting each edge in two and reconnecting. The location of a new vertex is obtained by a weighted average of the neighboring vertices. Vertices are classified as regular vertices and extraordinary vertices: a regular vertex is characterized by a valence of 6 (the vertex is shared by 6 triangles); meanwhile an extraordinary vertex has a valence not equal to 6. Next, four different subdivision situations have to be considered:

1. The edge connects two regular vertices (Fig. 1.a). The new point, indicated with a cross in the figure,

is computed through the weighted sums of neighbors' values using these weights:

$$a = 1/2 \quad c = -1/16$$
$$b = 1/8 \quad d = 0$$

2. The edge connects an extraordinary vertex of valence K and a regular vertex. The mask to be employed for the extraordinary vertex is indicated in Fig. 1.b. The weights are computed as follows:

$$K{=}3 \Rightarrow \begin{cases} S_0{=}5/12 & S_2{=}-1/12 \\ S_1{=}-1/12 & \end{cases}$$

$$K{=}4 \Rightarrow \begin{cases} S_0{=}3/8 & S_2{=}-1/8 \\ S_1{=}0 & S_3{=}0 \end{cases}$$

$$\underset{(K\neq 6)}{K{\geq}5} \Rightarrow S_j{=}\frac{\frac{1}{4}+cos(2\pi j/K)+\frac{1}{2}\cdot cos(4\pi j/K)}{K} \quad with\ \ j{=}0,\cdots,K-1$$

3. The edge connects two extraordinary vertices. In this case we take the average of the values computed using the appropriate scheme from the previous case for each endpoint.

4. Boundary edges are subdivided using the 1-dimensional 4 point scheme of Fig. 1.c (with $a = 9/16$ and $b = -1/16$ as weights). This scheme was proposed to avoid cracking effects between contiguous meshes.

3 MESH PARTITIONING: PARALLEL IMPLEMENTATION

The solution we propose in order to achieve high speed surface subdivision is based on the mesh partitioning into groups to be processed in parallel by different processors. The scheme in Fig. 2 shows a simple system with two processors. The original coarse mesh is replicated in every processor (first row in Fig. 2); this fact does not represent a

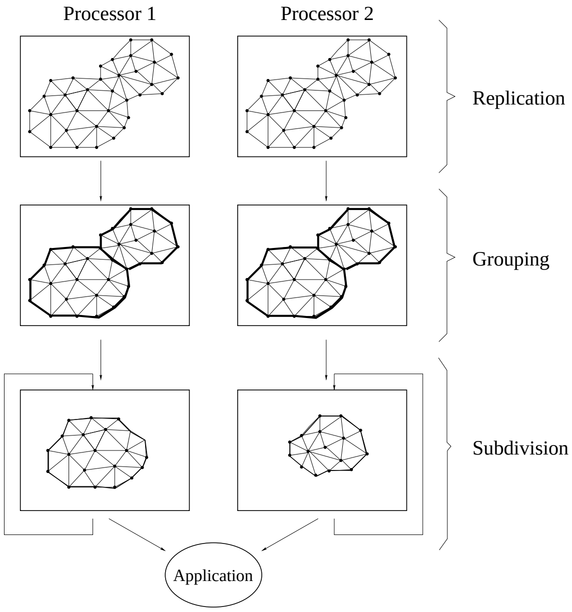


Figure 2: Generic Structure of the System

drawback due to the low storage requirements for coarse meshes. Next, each processor partitions the full mesh into small groups employing the new grouping algorithm [Amor00], [Bóo01] (second row in Fig. 2). Groups are assigned cyclically to the different processors in order to obtain load balancing. Once a group is assigned to a given processor the subdivision procedure explained in Section 2 is carried out (third row in Fig. 2).

3.1 Grouping Algorithm

In this section the grouping algorithm to subdivide the full mesh into small groups is presented. A detailed analysis can be found in [Amor00]. The grouping algorithm we use attempts to reduce the number of bounding triangles, generate well balanced groups, and cover the full mesh in an efficient way. The group construction algorithm is based on a central vertex (v_0) search, and the selection of the contiguous concentric triangle strips around this central vertex. Moreover, the central vertex is selected in such a way that successive processing groups are contiguous and without holes (non-assigned triangles) among groups.

```
1   L^start=NULL;
2   B = NULL;
3   v0= select_a_vertex(mesh);
4   while (vertices){
5        build_group(v0);
6        list_updating(L^start, B);
7        if(L^start) v=select_a_vertex(L^start);
8        else v= select_last_vertex(B);
9        v0 = find_group_center(v);
10  }
```

Figure 3: Grouping algorithm structure

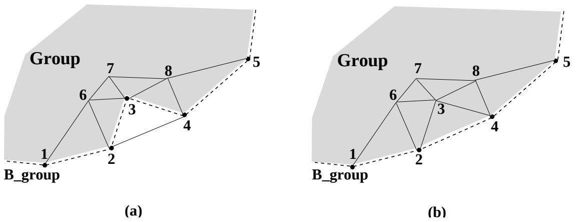


Figure 4: (a) Triangle full connected to *B_group* (b) Attachment to the group

The basic algorithm is summarized in Fig. 3. In lines 1 and 2, the lists of working starting points, L^{start}, and of bounding vertices of the global group are initialized. For the first processing group, B, a random central vertex, v_0, is selected (line 3). Once a central vertex is selected the group of triangles around this vertex is constructed (line 5). Next, the lists L^{start} and B are updated (line 6). After that, the bounding vertices of the actual group, *B_group*, are identified. Each time a group is identified, the list B of all building groups has to be computed. For the firstly computed group the list B coincides with the *B_group*. Each time a new group is attached to the previous computed groups the shared internal frontier edges have to be eliminated from B. The two extreme points of this frontier, characterized for not having all their triangles assigned, have to be considered as part of B and also as part of the list of working starting points, L^{start}. The working starting points are reference points for computing the new groups close to the previously computed triangles, allowing in this way an efficient covering of the full mesh of triangles (line 8). The next central vertex, v_0, is selected from this starting point (line 9). The basic idea consists of searching, from a starting vertex v of L^{start}, a v_0 point that is surrounded by a specific number of triangles rings without overlapping with any other group previously computed. With this central point a new group can be built. Then, this grouping process (lines 4 to 10) is repeated meanwhile non-assigned vertices remain in the mesh.

Once a group is made some additional steps [Amor00] have to be performed to assure a good coverage of the mesh. These steps are summarized in Figs. 4 - 6. First, triangles not included in the group with all their vertices located on the corresponding bounding *B_group* have to be attached to that group (Fig. 4).

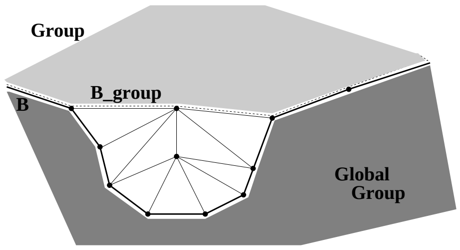


Figure 5: Hole identification

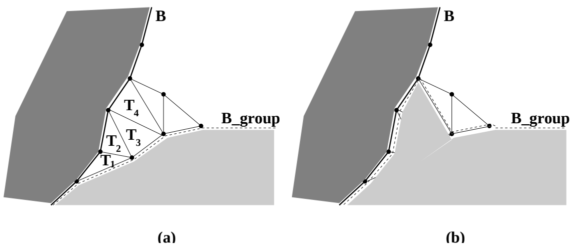


Figure 6: Cave (a) Identification (b) Elimination

Furthermore, some holes, that is, non-assigned triangles, can appear in the construction of a group. This can be observed in Fig. 5 where there are some non-assigned triangles between the last built group (marked in light grey) and the global group consisting of the previously built groups (marked in dark grey). These triangles have to be identified and attached to the current group under construction.

Finally, a strip of cave triangles can appear in the union between two groups. If these triangles are assigned to another new group, no neighbor information could be employed for their subdivision. This situation should be detected in order to include this strip of triangles in the current group under construction. As Fig. 6.a shows there is a strip of triangles $\{T_1, T_2, T_3, T_4\}$ delimited by the previous computed groups (B frontier) and the actual group (B_group frontier). These triangles have to be detected in order to be included in the current group; then the distribution indicated in Fig. 6.b is obtained.

3.2 Parallel Surface Subdivision Algorithm

In this section the parallel surface subdivision algorithm is presented (see Fig. 2). The parallel implementation is based on a coarse–grain approach, that is, each processor performs the whole computation of subdivision of triangles for a set of groups of the mesh.

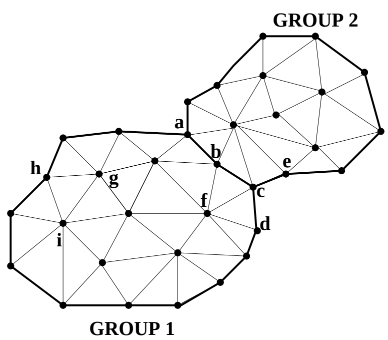


Figure 7: Vertices locations for two adjacent groups

With respect to the parallel implementation of the surfaces subdivision algorithm we have carried out a static assignment of the groups to the processors. The groups are sorted in decreasing order of number of triangles per group. Before beginning the recursive subdivision, the sorted groups are cyclically assigned to the processors, in order to achieve a load balancing. We assume p processors and a coarse mesh grouped into g groups; the number of groups assigned to each processor is g/p. Afterwards, during the recursive process each processor only computes the subdivision of the triangles assigned. It is important to note that the edges shared by two groups assigned to two different processors are this way computed twice. In order to avoid cracking effects the same new vertex coordinates in two groups have to be assured. Specifically, simple and local masks are employed on the border edges to avoid incoherent information among groups. Fig. 7 shows two adjacent groups assigned to different processors. Taking into account the neighbour information available in each processor the edges can be classified in three classes:

- Internal edges. In this case both vertices of the edge are internal to the group (eg. edge $g - i$ of Fig. 7). The masks for ordinary and extraordinary vertices (Fig. 1.a and 1.b) could be employed as all the neighbours information is available.

- Border edges. In this case both vertices of the edge are located on the border of the group. Let us consider as an example the edge between vertices b and c in Fig. 7. In this case the boundary mask (Fig. 1.c) is not suitable as it could produce cracking effects between neighbouring groups. From the point of view of the group 1, the boundary mask would imply vertices a, b, c and d, meanwhile from the point of view of group 2 the boundary mask would imply vertices a, b, c and e. This suggests the utilization of a mask which only takes into account the current working edge (vertices b and c).

- External edges. In this case only one of the vertices is located on the border of the group (eg. edges $h-g$ and $b-f$ of Fig. 7). In these cases only the neighbour information of the internal vertex (g and f) is available. We employ the simple mask that only takes into account the current working edge (vertices h and g, and vertices b and f, respectively).

As it was indicated, no neighbour information is considered in the border and external edges. Nevertheless, these conditions can be relaxed if contiguous groups are assigned to the same processor. For example, if group 1 and group 2 of Fig. 7 were assigned to the same processor, all the neighbour information for vertices b and f would be available, so that the original Modified Butterfly masks can be employed.

4 EXPERIMENTAL RESULTS

The Modified Butterfly subdivision algorithm has been implemented on the SGI Origin 2000 distributed-shared memory computer using the message passing programming model. We have used the MPI programming environment.

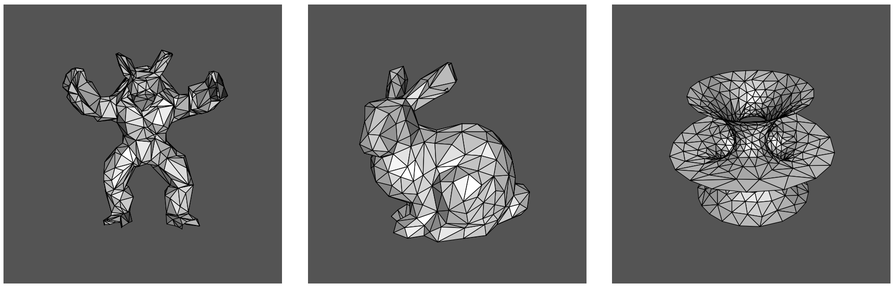
Table 1: The average errors of the parallel algorithm

N. Proc.	2 rings	3 rings	4 rings
2	0.0041	0.0034	0.0013
4	0.0057	0.0038	0.0024
16	0.0064	0.0044	--
N. groups	0.0065	0.0044	0.0037

Three different models, shown in Fig. 8, have been rendered using the algorithm: *armadillo*, *bunny* and *hypersheet*, with 799, 499 and 917 triangles respectively. In Fig. 9 the subdivided *bunny* meshes after 2 and 4 iterations are shown. The results in terms of speedups are shown in Fig. 10, both for 2 rings and for 4 rings after 5 iterations. The speedup of our parallel algorithm with respect to the sequential algorithm without grouping is shown. The execution time of the sequential algorithm for *hypersheet* is 8.45 seconds, and it is 0.99 seconds using 2 rings on 16 processors. As we can see, though better speedups are achieved by using 2 rings, from a certain number of processors the speedup does not increase accordingly. As future work, we intend to use a dynamic scheduling in order to achieve better load balancing.

The number of triangles per group for 2, 3 and 4 rings of the *bunny* model is depicted in Fig. 11. As we can see, the grouping algorithm covers the full original mesh in an efficient way, producing a balanced number of triangles per group. Obviously, the increasing number of rings produces an increase in the number of triangles per group, and a reduction in the number of groups. The number of simple masks employed on the border and external edges is lower so higher quality are obtained in the resulting images.

In order to evaluate the degree of error due to the grouping algorithm, and the simple masks employed on the border and external edges, we employ the average error between the subdivided mesh with the sequential algorithm, and the subdivided mesh with the parallel algorithm. In Table 1 the average error for 4 iterations is indicated. Specifically, cases from 2 to 4 rings were considered. In first, second and third rows the results for 2, 4, and 16 processors are indicated. In the last row a system with as much processors as num-

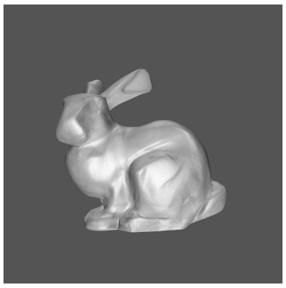


(a) (b) (c)

Figure 8: The original coarse meshes: (a) *armadillo* with 799 triangles, (b) *bunny* with 499 triangles and (c) *hypersheet* with 917 triangles

ber of groups was considered. This configuration represents the worst case as every border and external edge of all the groups are processed using the simple mask without employing neighbour information.

(a)



(b)

Figure 9: Subdivided *bunny* meshes after: (a) 2 iterations and (b) 4 iterations, with 7984 and 127744 triangles respectively

As we can see on the table, the increasing number of rings produces a reduction in the error because the number of border and external edges is lower. On the other hand, the increasing number of processors produces an increase in the error because there is more probability than neighbouring groups were located on different processors. However, though different numerical errors are obtained, no difference in quality can be appreciated in the final images. The grouping strategy and the corresponding simple masks employed do not affect this quality.

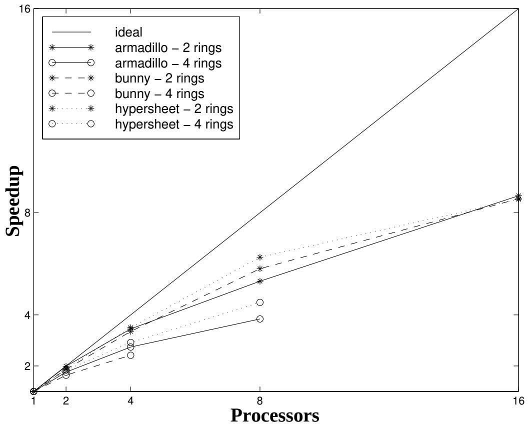


Figure 10: Speedup for the three different models for 2 and 4 rings

5 CONCLUSIONS

In this paper we have described a parallel implementation of the Modiffied Butterfly scheme for triangular meshes on

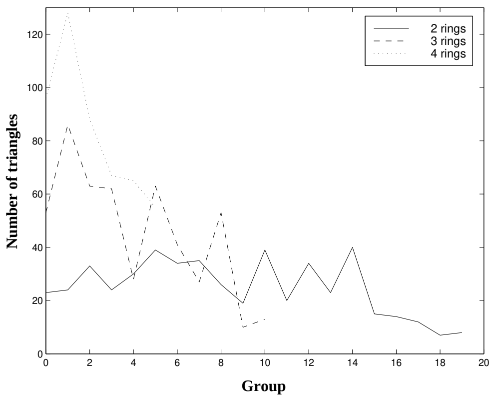


Figure 11: Number of triangles per group for 2, 3 and 4 rings of the *bunny* model

distributed-shared memory multiprocessors using the message passing programming model. The original mesh is partitioned into small groups employing a grouping algorithm. The groups are sorted in decreasing order of number of triangles. Afterwards, the sorted groups are distributed cyclically, trying to assign the same number of computations to each processor. Once a group is assigned to a given processor the subdivision procedure is carried out. In order to avoid cracking effects among groups a slight modification of the Modified Butterfly algorithm is used.

The parallel method maintains a high quality in the resulting images, in spite of the greatest number of border edges and the simplification introduced by the grouping process. Furthermore, a good speedup was achieved, so we are close to the objective of real time rendering (at least managing relatively large meshes).

Finally, it should be remarked that the Modified Butterfly algorithm was used in this paper as a testbed for checking the benefits of parallelization in surface subdivision for 3D image synthesis; actually, we think similar results could be obtained using the Loop algorithm and, in general, all those algorithms in which the first order neighbour information is employed.

ACKNOWLEDGEMENTS

This work was supported by the Ministry of Education and Science (CICYT) of Spain under the project TIC 2001-3694-C02-02 and by the Vice-Rector for Research of the University of A Coruña (Spain). The authors would like to thank to Centro de Supercomputación Complutense (Madrid, Spain) for providing access to the SGI Origin 2000.

REFERENCES

[Amor00] M. Amor, M. Bóo, M. Doggett, J. Hirche, and W. Strasser. A meshing scheme for memory efficient adaptive rendering of subdivision surfaces. Technical Report WSI-2000-21, Wilhelm-Schickard-Institut, Fakultät für Informatic, Univ. Tübingen, 2000.

[Bóo01] M. Bóo, M. Amor, M. Doggett, J. Hirche, and W. Strasser. Hardware support for adaptive subdivision surface rendering. In *SIGGRAPH/Eurographics Graphics Hardware Workshop 2001*, 2001.

[Dyn90] N. Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graphics*, 9(2):160–169, 1990.

[Loop87] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Univ. of Utah, 1987.

[Schrö00] P. Schröder and D. Zorin. Subdivision for modeling and animation. In *SIGGRAPH'00 Course Notes. ACM SIGGRAPH*, 2000.

[Zorin97] D. Zorin. Subdivision and multiresolution surface representations. Master's thesis, Univ. of Caltech, 1997.