Hardware Support for Adaptive Subdivision Surface Rendering

M. Bóo[†] M. Amor[††] M. Doggett [†††] J. Hirche [†††] W. Strasser[†††]

[†] Dept. of Electronic and Computer Engineering. University of Santiago de Compostela. Spain
[††] Dept. of Electronic and Systems. University of La Coruña. Spain
[†††] WSI/GRIS, University of Tübingen. Germany

Abstract

Adaptive subdivision of triangular meshes is highly desirable for surface generation algorithms including adaptive displacement mapping in which a highly detailed model can be constructed from a coarse triangle mesh and a displacement map. The communication requirements between the CPU and the graphics pipeline can be reduced if more detailed and complex surfaces are generated, as in displacement mapping, by an adaptive tessellation unit which is part of the graphics pipeline. Generating subdivision surfaces requires a large amount of memory in which multiple arbitrary accesses are required to neighbouring vertices to calculate the new vertices. In this paper we present a meshing scheme and new architecture for the implementation of adaptive subdivision of triangular meshes that allows for quick access using a small memory making it feasible in hardware, while at the same time allowing for new vertices to be adaptively inserted. The architecutre is regular and characterized by an efficient data management that minimizes the data storage and avoids the wait cycles that would be associated with the multiple data accesses required for traditional subdivision. This architecture is presented as an improvement for adaptive displacement mapping algorithms, but could also be used for adaptive subdivision surface generation in hardware.

1 Introduction

A significant rendering bottleneck for high performance graphics systems is the CPU to graphics processor bus. High quality surfaces require the transmission of millions of triangles over this bus reducing the performamce of the system. Reducing this communication requirement has recieved significant attention in recent years. Different attempts to improve the communication problem have been developed including: compression [5], simplification of the images complexity [3], interactive multiresolution meshes [6], and displacement mapping [7].

Displacement Mapping is an effective technique for encoding the high levels of detail of surface models through the utilization of coarse triangle meshes together with displacement maps. Extending the hardware rendering pipeline to be capable of handing displacement maps as geometric primitives will allow highly detailed models to be constructed without requiring a large amount of triangles to be passed from the CPU to the graphics pipeline.

The displaced subdivision surface strategy proposed in [8] combines the non-adaptive Loop subdivision scheme [9] with the displacement map representation. The original mesh is simplified and the displacement map accordingly sampled in order to be sent to the rendering pipeline. Although in [8] the details for rendering the surface using hardware is not presented, extensions have to be added to the pipeline to subdivide the coarse mesh according to the Loop algorithm and afterwards, to apply the displacements to the resulting coordinates. A simple and regular hardware implementation for the Loop subdivision algorithm has been proposed in [2]. This solution exploits the piecewise polynomial structure of the Loop algorithm with the application of forward difference techniques. This hardware can be easily extended for other non adaptive subdivision algorithms which also generate piecewise polynomial surfaces but it can not be generalized to any adaptive subdivision scheme.

Using Displacement Mapping to generate detailed triangle meshes can generate a large number of triangles for rendering. To reduce this number of triangles, an adaptive strategy was presented in [4]. Each triangle is conditionally subdivided as a function of the displacement map information. The hardware proposal is based on the tessellation of individual edges of each triangle by means of the local information of each edge. Using an adaptive subdivision the number of triangles to be rendered are reduced, but neighbouring triangle vertex information is not used in each subdivision step.

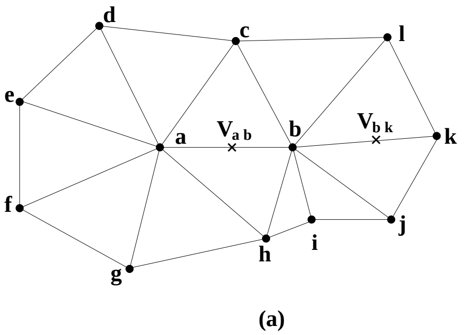
Smoother surfaces can be obtained if more than just the local information is employed for the adaptive subdivision procedure. This results in the final detailed mesh being obtained in a more coherent manner in accordance with the displacement map surface. This requires that each triangle to be tessellated uses the information of the adjoining triangles. In order not to send each triangle more than once from the CPU to the graphics pipeline, its information should be maintained in the hardware unit. On the other hand, this information has to be stored taking into account that multiple data has to be accessed simultaneously per subdivision step. In this paper we present a methodology and an architecture for the implementation of the adaptive subdivision of surfaces in which neighbour information is employed. The solution we propose starts by grouping the neighbouring triangles in the CPU before sending them only once to the graphics pipeline. Each group is fully processed in the new hardware unit before sending the following group. The efficient management of the data we propose permits the simultaneous access to all the neighbour information required for each partitioning step, avoiding wait cycles. This is due to the data distribution in the memories and to the simple and compact mesh connectivity specification proposed. The mesh representation strategy, together with the regular hardware structure developed, makes this solution interesting not only in the displacement map environment, but also in any algorithm where an adaptive tessellation using neighbouring triangle vertices is required.

In the next section, we present the adaptive tessellation algorithm based on neighbour information and the strategy employed to minimize the storage requirements. In Section 3 we introduce the methodology we propose for the efficient data management and, in Section 4, the architectural model we have employed for its implementation. In Section 5 we evaluate the algorithm. Finally, in Section 6 we present the main conclusions of the work.

2 Adaptive Tessellation based on Neighbour Information

Displacement mapping requires a coarse triangle mesh that approximates the surface to be modeled with a displacement map containing the finer geometric detail. If the base triangle mesh has a coarser resolution than the displacement map, the mesh should be retessellated according to the displacement map. To improve the previous

HWWS '01 Los Angeles, CA USA
© ACM 2001 1-58113-407-X…$5.00



Interior Edges

$$d_1 = \sum_{i=1}^{number\_neighb(a)} d(a, neighb(a,i))/number\_neighb(a)$$

$$d_2 = \sum_{i=1}^{number\_neighb(b)} d(b, neighb(b,i))/number\_neighb(b)$$

New point:

Vab = $\frac{d_1}{d_1+d_2} \cdot coord(a) + \frac{d_2}{d_1+d_2} \cdot coord(b)$

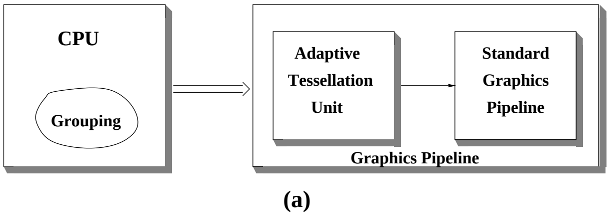
Border Edges

New point:

Vbk = $\frac{1}{2} \cdot coord(b) + \frac{1}{2} \cdot coord(k)$

(b)

Figure 1: *(a) Neighbour information required for vertex insertion. (b) Coordinates computation*



```
$L^{start}$=NULL;
B = NULL;
$v_0$= select_a_vertex(mesh);
while (vertices){
        build_group($v_0$);
        list_updating($L^{start}$, $B$);
        if($L^{start}$==NULL) v=select_a_vertex(B);
        else v= select_last_vertex($L^{start}$);
        $v_0$ = find_group_center(v);
}
```

(b)

Figure 2: *(a) Generic Structure of the System (b) Grouping algorithm*

adaptive tessellation algorithm implemented in hardware [4], not only local information of each edge to be split should be considered, but the neighbour information as well. Smoother surfaces can be obtained using masks such as those employed in subdivision surfaces algorithms [10], Loop [9] and modified Butterfly [11].

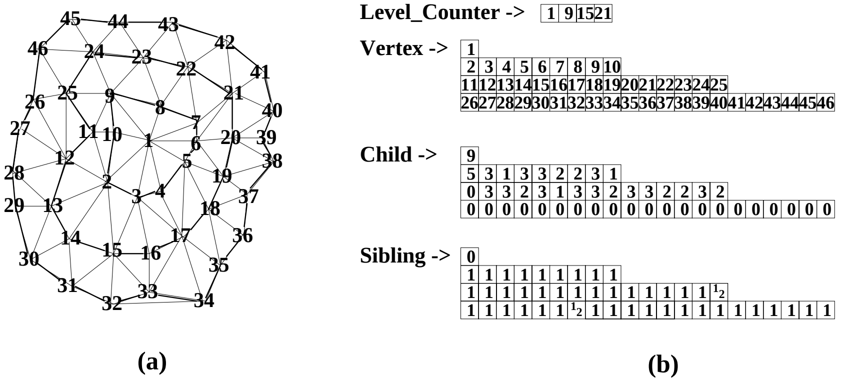
For the example in Figure 1(a) these algorithms split the edge between vertices a and b (vertex Vab) taking into account not only the information of the extreme vertices a and b but also the information of vertices $c, d, \cdots l$. The utilization of these subdivision algorithms for the adaptive scheme implies the recomputation of the original displacement map values. This is due to the fact that the computed coordinates do not lay on the original triangles so the initial displacement map would not be appropriate for the processed mesh. If the displacement map is not recomputed, other division masks could be proposed. Figure 1(b) shows our initial proposal for the coordinates computation. For the edge between two interior vertices a and b the coordinate of the new point Vab inserted in the edge is computed through the weighted sum of the a and b coordinates. The weights associated with a (d_1) and b (d_2) are obtained as a function of the average distance (d) of the neighbour vertices to the vertices a and b respectively. In case at least one vertex of the edge is on the border of the mesh we propose the utilization of the current working edge information, that is, $d_1 = d_2 = \frac{1}{2}$. For example, in the figure the vertex k is on the border, so the new vertex Vbk implies computations based on local information.

In this paper the non recomputation of the displacement map was considered so the new mask was used. In any case, it is important to note that the methodology and architecture we proposed are generic and valid for all masks employing the first order neighbour information.

Each triangle to be tessellated requires the information of the adjoining triangles to be processed. In order not to increase the communication between CPU and graphics pipeline the computation of the full fine mesh is computed using the new hardware added to the graphics pipeline. The solution we propose is schematically presented in Figure 2(a). The neighbour triangles have to be grouped in the CPU in order to be sent to the graphics pipeline. These groups of triangles are processed in an adaptive tessellation unit and then final rendering is performed by a standard graphics pipeline. The number of triangles in each group has to be determined as a function of the storage requirements during the tessellation operation. The utilization of only local information to the edge on the borders (Figure 1(b)) avoids possible cracking effects in the joining among groups but, obviously, the number of border edges have to be minimized in order to reduce the subdivisions in which only local information is employed. In the following subsections the proposed grouping algorithm and its connectivity representation are presented. A detailed analysis can be found in [1].

2.1 Grouping Algorithm

In this section the grouping algorithm to subdivide the full mesh into small groups is presented. These groups are sent from the CPU to the graphics pipeline where the adaptive tessellation is processed (Figure 2(a)). The grouping algorithm we propose reduces the number of bounding triangles, covers the full mesh in an efficient way and minimizes the amount of information to be sent. On the other hand, the mesh representation employed simplifies the storage and data management in the adaptive tessellation unit. The group construction algorithm is based on the selection of a central vertex v_0 and the contiguous concentric triangle strips around this central vertex. The central vertices are selected in such a way that successive processing groups are contiguous and without holes (non-sent tri-



Level_Counter -> 1 9 15 21

Vertex ->
1
2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46

Child ->
9
5 3 1 3 3 2 2 3 1
0 3 3 2 3 1 3 3 2 3 3 2 2 3 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sibling ->
0
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 $^{1}{}_{2}$
1 1 1 1 1 1 $^{1}{}_{2}$ 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 3: *(a) Group example (b) Group representation*

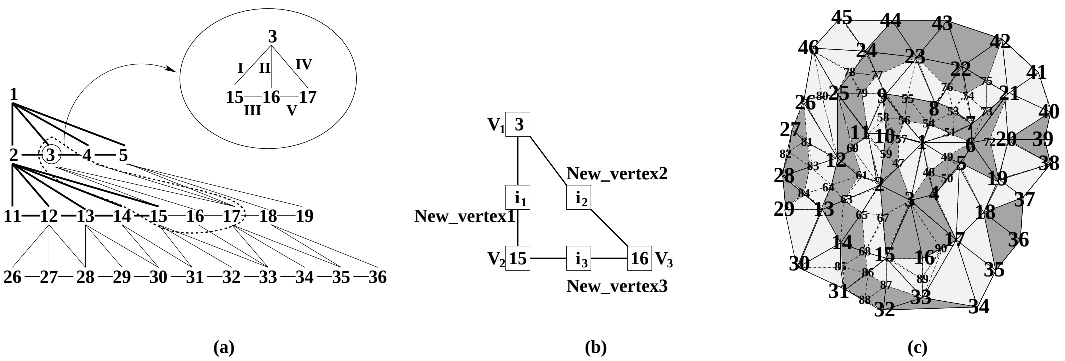


Figure 4: *(a) Scheduling example (b) Notation* (c) Example of tessellated group

angles) between them. The collection of all identified groups constitutes a global group, already sent to the graphics pipeline, characterized by a list B of vertices in the boundary.

The search for each central point v_0 is based on the utilization of a set of starting points from the list B of vertices in the boundary. Specifically, each time a new group is computed, two new starting points are identified: the non-interior common vertices between the previous global group and the new group. Each starting point is employed as a seed for the identification of the central point of a new group. Taking as a reference this starting point each central point candidate v_0 is analyzed in order to build up an adequately sized group contiguous to the global group.

The basic algorithm [1] is outlined in Figure 2(b). For the first processing group a random central vertex v_0 is selected (line 3). Then, the group of triangles around this vertex is selected (line 5) and the list of working starting points L^{start} and the list B of vertices in the boundary of the global group are updated (line 6). To construct another group a starting point has to be selected (lines 7 and 8) and the next central vertex v_0 identified (line 9). This grouping process (from lines 4 to 10) is repeated whileever there are still non processed triangles remaining in the mesh.

2.2 Group Representation

In this subsection the group representation we propose is analyzed. It is based on the identification of strips of triangles around the central vertex v_0 and the identification of their connectivity. For example we will look at the group of triangles of Figure 3(a). The central vertex $v_0 = 1$ is surrounded by a first strip of triangles defined by the lists $\{1\}$ and $\{2, 3, \cdots, 9, 10\}$ meanwhile the second strip is defined by the interior list $\{2, 3, \cdots, 9, 10\}$ and the exterior $\{11, 12, \cdots, 24, 25\}$ and so on. These lists are indicated in consecutive rows of the $Vertex$ list of Figure 3(b) where the number of elements in each row is stored in the $Level\_Counter$ list. In order to specify the vertex connectivity two additional lists are included: $Child$, that specifies the connectivity among rows of vertices, and $Sibling$, that specifies the connectivity inside a row.

In the i-th row of the $Child$ list the connectivity between the i-th and $i+1$-th rows of the $Vertex$ list is specified. Specifically, it stores the number of vertices of the $i+1$-th row that are connected to each vertex of the i-th row. Note that two contiguous vertices of the i-th row are connected to two contiguous groups of children of $i+1$-th row with a common child between them. To clarify let us considerer the example of Figure 3. The connectivity among $\{2, 3, \cdots, 9, 10\}$ and $\{11, 12, \cdots, 25\}$ (second and third rows in $Vertex$ list) is given by children = $\{5, 3, 1, 3, 3, 2, 2, 3, 1\}$ (second line in $Child$ list). This means that the vertex 2 (second row of $Vertex$ list) is connected with the first five points in the third row $(11, 12, 13, 14, 15)$, the following vertex 3, with three children, is connected to the last child of vertex 2 (vertex 15) and another two children (16 and 17). The following vertex, 4, with one child, is connected to vertex 17, and so on.

To complete the description, the connectivity among vertices in the same row of the $Vertex$ list has to be specified. It is enough to indicate the connectivity of each vertex with the following vertices on the row (connectivity with the previous points can be easily deduced). This information is stored in the $Sibling$ list. If the i-th vertex of a row has connectivity 1 this indicates that it is connected to the $i+1$-th vertex of the same row. In the same way, a connectivity 0 indicates that the vertex is non-connected to the following vertices. On the other hand, a connectivity value $j > 1$ indicates that the i-th vertex is connected to the non contiguous $i+j$-th vertex. If the i-th vertex has connectivities $1, j_1, j_2, \cdots j_n$ with $j_r \neq 0, 1$ $(r = 1, \cdots n)$ then it is connected to the $\{i+1, i+j_1, \cdots, i+j_n\}$-th

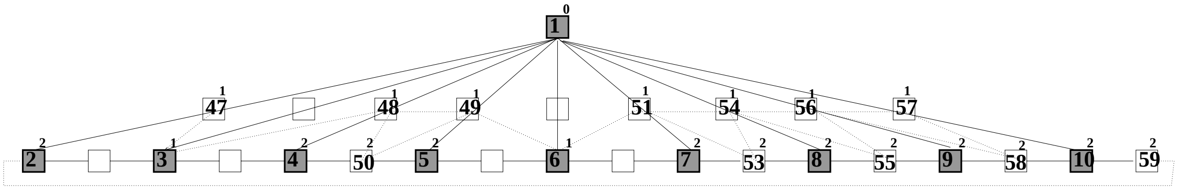


Figure 5: *Detailed example of vertex classification procedure*

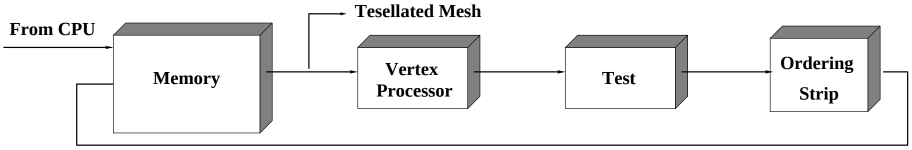


Figure 6: *Block diagram of the architecture*

vertices of the row. As an example let us consider the connectivity of the fourth row $\{26, 27, \cdots, 45, 46\}$ of the $Vertex$ list. The sibling connectivities of $26, \cdots, 31$ is 1, that is, each vertex is connected with the following one. But the connectivity of vertex 32 is $1, 2$ because it is connected to vertices 33 and 34.

3 Towards Hardware Implementation of Adaptive Tessellation

The adaptive tessellation implies conditional subdivision of each edge [4]. In our new proposal, each new vertex computation implies the utilization of not only the coordinates of the extreme points of the edge, but also the corresponding coordinates of the neighbours of these vertices. So the neighbours information has to be stored in such a way that it can be efficiently accessed. Moreover and taking into account that the adaptive tessellation has to be performed repeatedly over the full mesh, the resulting mesh obtained in each iteration has to be efficiently stored in order to be reused in the following iteration. In this section the data storage and management we propose to solve this problem is presented. The following presentation focuses on the way of covering the mesh first and the storage method employed in order to recompute it follows.

It is important to note that the algorithm we propose is based on the efficient mesh representation previously presented. According to this representation, the mesh is traversed through the computation of the edges of the triangles along each strip before starting with the following strip [1]. The strings of vertices are computed sequentially through the analysis of all vertices in each row. For each vertex v of each string all the children $c_0, \cdots, c_{children-1}$, that is, all the connected vertices of the following string of vertices are analyzed. Firstly, the computations related with the edge connecting vertex v and the first child c_0 are performed. After this, two edges have to be analyzed, the edge connecting the root vertex v with the new child c_1 and the edge connecting the children c_0 and c_1. To clarify let us consider the example of Figure 4(a) where the adaptive tessellation of the group of Figure 3 is considered. The vertex 1 (first list) and its children $(2, 3, 4, 5, \cdots)$ are analyzed first. After this the second list $\{2, 3, 4, 5, \cdots\}$ is processed starting from the first vertex 2 and its corresponding children $(11, 12, 13, 14, 15)$. Let us suppose that these edges have been already processed (indicated in the figure with dark lines) and let us consider the computations related with the vertex 3 and its corresponding children $(15, 16, 17)$. The edges to be computed and the order of computation are detailed in the Figure 4(a). The first edge to be computed (label I) involves the edges $3 - 15$. The following pair of edges to be computed are edge $3 - 16$ (label II) and edge $15 - 16$ (label III). Then, the last pair of edges related with vertex 3 are computed: edge $3 - 17$ (label IV) and $16 - 17$ (label V).

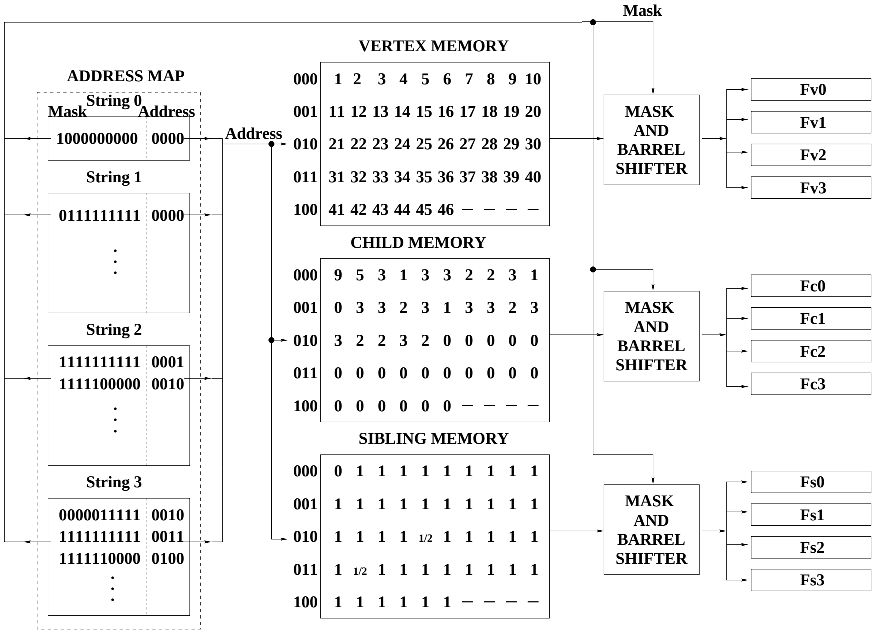
String	Edge (3-15)				Edge (3-16)		
1	1				1		
2	2	3	4		2	3	4
3	14	15	16	17	15	16	17
4	31	32	33		33		

Table 1: Example of required neighbour information

An important characteristic of this way of covering the mesh can be outlined from this example: the neighbours required to compute one edge are mostly reused to compute the following edge. For example, the necessary neighbours to compute the edges $3-15$ and $3 - 16$ are indicated in Table 1. Note that four groups of data, from four consecutive strings of vertices, are required for each edge. For this specific example new data is not required for the computation of the second edge. This method of reemploying the data permits, as shown later, a reduction in the number of accesses to the memory where the data is stored.

To complete the presentation the tessellated mesh storage method is analyzed [1]. The new tessellated mesh has to be stored according to the concentric triangles strips notation previously presented. The algorithm we propose is basically a vertex classification in which the vertices are classified in strings. Each time an edge is processed, the new vertex candidate coordinate is determined and conditionally inserted. After this, the surrounding vertices are assigned to a string. In Figure 4(b) the notation and algorithm we propose is briefly introduced. The first child of vertex 3, that is, vertex 15, is being processed. Once the midpoint, New_vertex1, is computed a test [4] has to be made in order to determine if the splitting is performed. The decision is labeled as i_1, where $i_1 = 0$ if there is no splitting and $i_1 = 1$ if the edge is split. When the second child of vertex 3 (vertex 16) is considered, two new vertices have to be computed, New_vertex2 with decision i_2 and New_vertex3 with decision i_3. Once the decisions have been performed a classification procedure has to be realized, that is, the vertices that delimit the concentric triangle strips have to be identified. This requires the application of simple string classification rules (tabulated in [1]) where each vertex is classified on a string as a function of the previously computed vertices and their classification. Similar rules are also applied to compute the sibling and child connectivity.

To clarify the algorithm let us considerer the mesh of Figure 3 and the tessellation indicated in Figure 4(c). The new vertices originated during the tessellation operation are indicated in small labels.



VERTEX MEMORY

000	1	2	3	4	5	6	7	8	9	10
001	11	12	13	14	15	16	17	18	19	20
010	21	22	23	24	25	26	27	28	29	30
011	31	32	33	34	35	36	37	38	39	40
100	41	42	43	44	45	46	–	–	–	–

CHILD MEMORY

000	9	5	3	1	3	3	2	2	3	1
001	0	3	3	2	3	1	3	3	2	3
010	3	2	2	3	2	0	0	0	0	0
011	0	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	–	–	–	–

SIBLING MEMORY

000	0	1	1	1	1	1	1	1	1	1
001	1	1	1	1	1	1	1	1	1	1
010	1	1	1	1	1/2	1	1	1	1	1
011	1	1/2	1	1	1	1	1	1	1	1
100	1	1	1	1	1	1	–	–	–	–

Figure 7: *Memory unit architecture*

Six strips of triangles were identified and marked in different grey levels. As an example of string classification let us analyze the processing of the original first string of triangles delimited by lists $\{1\}$ and $\{2, \cdots, 10\}$ of Figure 4(c) detailed in Figure 5. In this figure the vertices related with the original mesh are indicated in grey boxes, the new vertices in thinner boxes and the non inserted vertices with empty boxes. The string identifier[1] of each vertex, obtained with the classification rules in [1] is indicated as a number over the corresponding box. The classification method has an easy intuitive interpretation: the string identifier of all roots of a vertex are analyzed so that the minimum value plus one is assigned as its string identifier. As an example, all the children of the vertex 1, of the string 0, are assigned to the string 1.

4 Adaptive Tessellation Architecture

Figure 6 shows the block diagram of the architecture for adaptive tessellation. The architecture has four modules: 1) Memory unit, 2) Vertex processor, 3) Test unit and 4) Ordering strip unit. The incoming mesh is stored in a Memory unit in order to be processed. This Memory unit is also employed for the storage of the intermediate processed meshes and also for the management of the data. It provides four consecutive strings of vertices (see Table 1 example) to the second unit (Vertex processor) in such a way that the wait cycles of this unit are minimized. The strings of vertices are processed by the Vertex processor where the coordinates of the vertex candidates are obtained. These coordinates are delivered to the Test unit [4] to decide if each edge is split. The decisions together with the corresponding coordinates (if the splitting is performed) are introduced in a final unit, the Ordering strip unit, where the vertices are reordered in strings of concentric vertices. The final mesh is stored in the first unit in order to be processed again. The recursive processing of the mesh is repeated a given number of iterations (defined by the user) or until the mesh is fully tessellated accordingly to the displacement map. Due to length constraints in the following subsections only the structure of the Memory and Ordering strip units are briefly introduced although an extended revision of all modules can be found in [1].

4.1 Memory Unit

To process each edge of the mesh, the Vertex processor requires the information of all the neighbours around the extreme vertices of the edge. All edges are processed covering a strip of triangles before starting with a new strip. This implies that all data of the four contiguous strings of vertices (see Table 1 example) will be required consecutively before starting with a new strip of triangles. Then, the data has to be stored in such a way that an easy access to the strings of vertices could be developed. The architecture we propose is depicted in Figure 7. Three different memories (Vertex, Child and Sibling) are employed to store the vertices, the corresponding children and the sibling information. As an example the data indicated in the figure corresponds to the group example of Figure 3. In order to optimize the storage utilization we propose $m + 1$ word memories where m is the maximum number of vertices connected to a given one. The information has to be stored in a coherent way in the three memories.

The best storage would imply that the information associated with consecutive vertices of a string would be in consecutive positions of the memory. But this distribution is possible only for the initial mesh (coming from the CPU) as during the subdivision the vertices of each string are not obtained in a consecutive way. A set of memories (indicated as address maps in the figure) to store the addresses of successive sections corresponding to the same string is employed. Specifically, the address map of the data corresponding to each string i of vertices is stored in a $String\ i$ memory. The reason for this storage scheme is detailed in the next subsection. As usual the number of vertices in each string section does not fit with the word size of the memory, so a mask should be employed to indicate the part of the word that stores the correct information. For example, the string 2 of vertices of Figure 7 is stored in addresses 0001 with mask $\{1111111111\}$ and 0010 with mask $\{1111100000\}$. In the vertex memory the information relative to vertices $\{11, 12, \cdots, 25\}$ is identified.

Taking into account that the neighbours information can be

[1] The row number of the vertex in the current mesh group data structure.

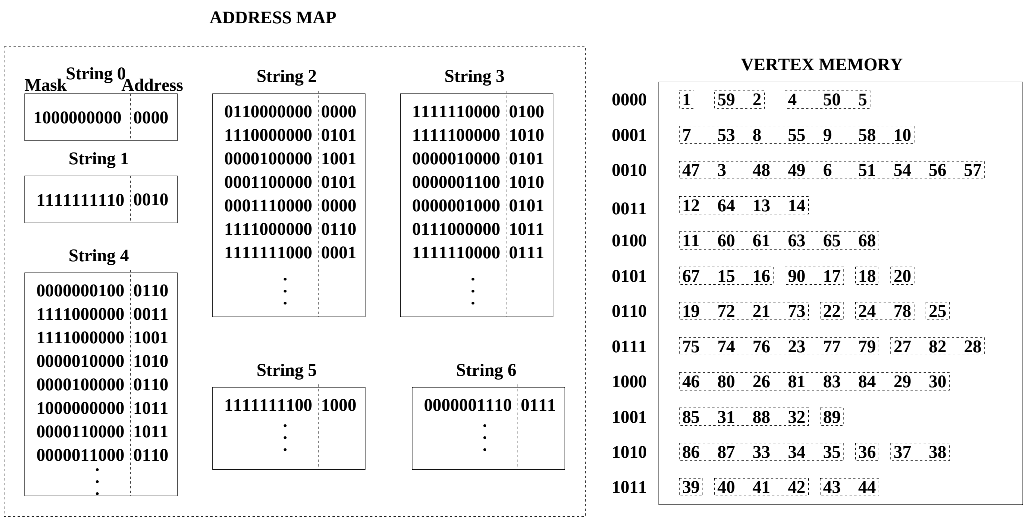


Figure 8: *Example of storage of a tessellated mesh*

reemployed for successive computations (see Table 1 example) the memory system can be optimized through the utilization of a system of FIFO (first-in-first-out) queues. As four strings of vertices are involved in each edge computation, four FIFO queues have to be employed. Specifically, three different sets of four FIFO queues are required for keeping the vertex (Fv), the children (Fc) and the siblings (Fs) information. To select the adequate part of the m+1-word memory (mask information) a set of normalization units (barrel shifters) are employed. The FIFO system, together with the reutilization of the data assures an efficient management avoiding any wait cycles for the Vertex processor [1].

4.2 Ordering Strip Unit

The ordering strip unit performs the vertex strings classification and computes the children and sibling connectivities accordingly to the simple rules tabuled in [1]. The strips of triangles are processed so the new vertices have to be classified and adequately stored in the memory system. Due to length constrains we will focus the presentation on the memory information updating procedure.

For the computation of each strip of triangles the strings of vertices of the tessellated system are obtained in sections. This can be observed in Figure 5 were the computation of consecutive triangles of the strip generates a non complete string of vertices with a identifier 2, that is, the string 2 is obtained in sections. Specifically, the computation of each triangle generates vertices of at most two strings (together with the already computed root vertices) and, it can be ensured that the computation of the contiguous triangle of the same strip will maintain the continuity of, at least, one of the vertex strings. Then, two FIFO queues can be employed for storing the one/two strings under construction. Once a string is broken, the content of the FIFO queue is stored in the memory and the corresponding address map updated. A procedure was developed [1] to reserve memory positions for storing the addresses of consecutive strings sections not already computed.

As an example the final memory distribution after computing all the group (Figure 4(c)) is indicated in Figure 8. The different sections of strings stored in the vertex memory are indicated with dashed boxes. According to this storage, the address map memories are coherently updated in such a way that each string i of vertices can be reconstructed by reading the consecutive addresses (of the vertex memory) indicated in the $String\ i$ memory. The same reconstruction operation can be performed over the memories of children and siblings in order to obtain the Child and Sibling lists. As an example let us reconstruct the string 3 corresponding to the tessellated mesh. In order to do this the seven addresses indicated in the corresponding $String$ 3 memory have to be accessed. The stored information corresponding to the consecutive addresses is: $\{11, 60, 61, 63, 65, 68\}$, $\{86, 87, 33, 34, 35\}$, $\{18\}$, $\{37, 38\}$, $\{20\}$, $\{40, 41, 42\}$, $\{75, 74, 76, 23, 77, 79\}$, which corresponds to the fourth string of vertices of Figure 4(c).

5 Results

The methodology we propose permits the communication requirements between the CPU and the graphics pipeline to be reduced by performing complex surface generation in hardware. The classification unit is mainly compounded of memories and FIFO queues and simple additional hardware (adders, substractors, minimum and maximum selectors) for the vertex classification into strings. This reduces the communication problem with a reasonable increase in hardware requirements. The surface generation possibilities achieved through the introduction of this unit contrast with its simplicity.

Let us suppose a group with n triangles is to be processed a number i of iterations. The proposed system ensures that each edge is computed only once and that the processing rate is one edge per cycle. If the border of the group is not taken into account, each edge is shared by two triangles which implies, in the first iteration, the computation of $\frac{n \cdot 3}{2}$ edges. This is performed in $\frac{n \cdot 3}{2}$ cycles. If all the triangles are fully subdivided in each iteration, $4^i \cdot n$ triangles are finally obtained. This leads to the maximum computational rate of the system given by:

$$R_{max} = \frac{4^i}{\frac{3}{2}\left(\sum_{j=0}^{i-1} 4^j\right)} triangles/cycle$$

On the other hand the minimum computational rate is obtained if no triangles are subdivided in any iteration:

$$R_{min} = \frac{1}{3i/2} triangles/cycle$$

The computational rate values of any group is in the interval $[R_{min}, R_{max}]$. As an example, for 3 iterations $R_{min} = 0.22$

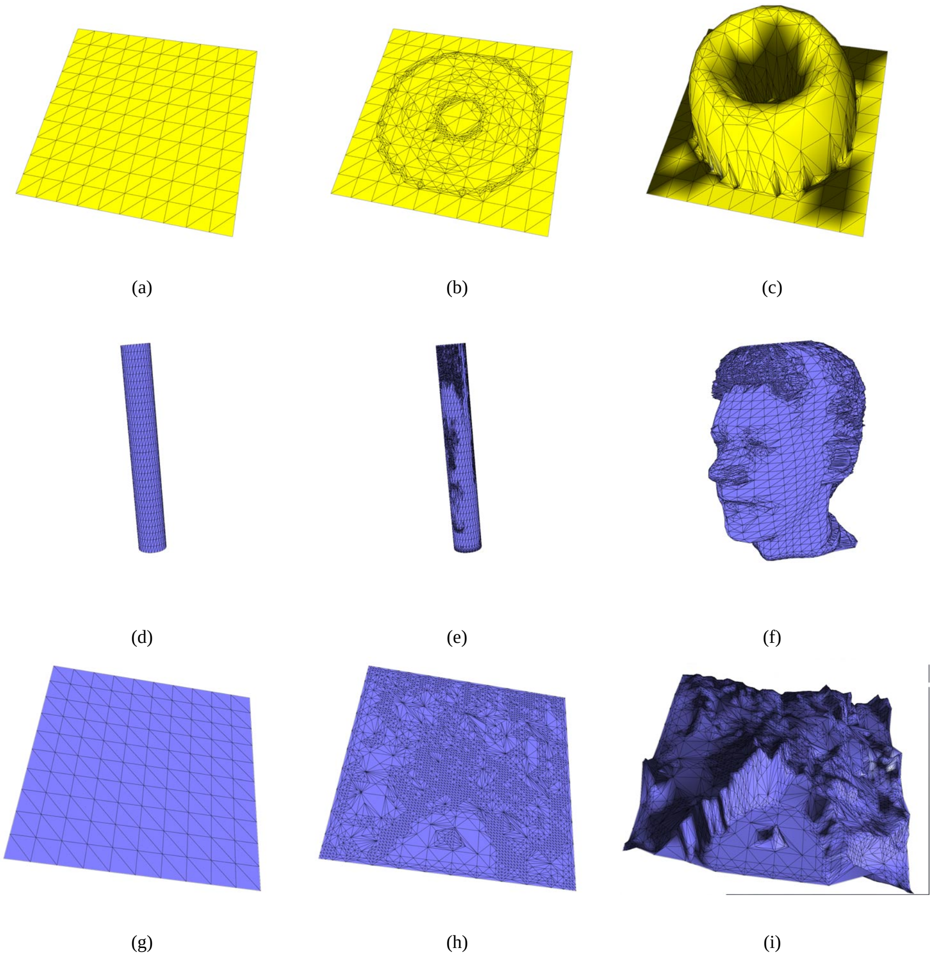


Figure 9: *Donut, Volker's head and Crater images. (a), (d), (g) shows the original coarse meshes, (b) (e), (h) the adaptively subdivided meshes and (c) (f), (g) final meshes with displacements.*

and $R_{max} = 2.03$ rates are obtained. Note that the R_{min} value corresponds to a non practical case in which no subdivision is performed, which means that the original mesh is very detailed. The system also has no recursive structure (only for the recomputation of the groups), so it can be pipelined with a minimum cycle period.

We have implemented our adaptive tessellation solution (Figure 6) through a software simulation. We have rendered several models to demonstrate the performance of the solution, the advantages/disadvantages of an adaptive tessellation based on neighbour information and the repercussions of the grouping strategy. In the following results 3 iterations per mesh were employed.

Three different models have been rendered using the algorithm and are shown in Figure 9. Specifically, we have worked with a donut displacement over a flat square plane (Figures 9 (a),(b) and (c)), the Volker Blanz's head displacement map over a cylinder mesh (Figures 9 (d),(e) and (f)) and the Crater Lake also over a flat square plane (Figures 9 (g),(h) and (i)). In the first column (Figures 9 (a),(d) and (g)), the original coarse meshes are depicted, in the second column (Figures 9 (b),(e) and (h)) the meshes subdived according to the displacement map are shown and on the third column (Figures 9 (c),(f) and (i)) the final images after applying the displacements are indicated. While the original meshes, to be sent from the CPU to the graphics pipeline, have 200 triangles the final meshes have 1242, 4389 and 6379 triangles respectively. This method allows for high quality images to be obtained with very low transmision requirements. The adaptive structure of the subdivision can be observed in all images. For example the donut is less subdivided in regions close to the top where no detail is required. Note that the transition between high and low detailed regions is soft, resulting in smooth surfaces. The images correspond to the utilization of only 2 rings. This size was selected as a tradeoff between image quality and memory requirements as only slight differences are obtained using a greater number of rings per group. On the other hand, improvements are expected for other coordinate computation schemes alternative to the one proposed in Figure 1(b).

Due to the repercussions on the memory requirements of the graphics pipeline the number of triangles generated per group has to be analyzed. In Figure 10 the number of triangles per group when 2 rings are employed is detailed. In Figure 10(a) the number of original triangles (information sent from the CPU to the pipeline) per group for the two original meshes employed are indicated. It can be deduced that the grouping algorithm we propose permits coverage of the original image with a balanced number of triangles per group. Obviously, when the number of non sent triangles is lower (high group number) the possibility of constructing a group with the full rings is lower, so lower numbers of triangles are ob-

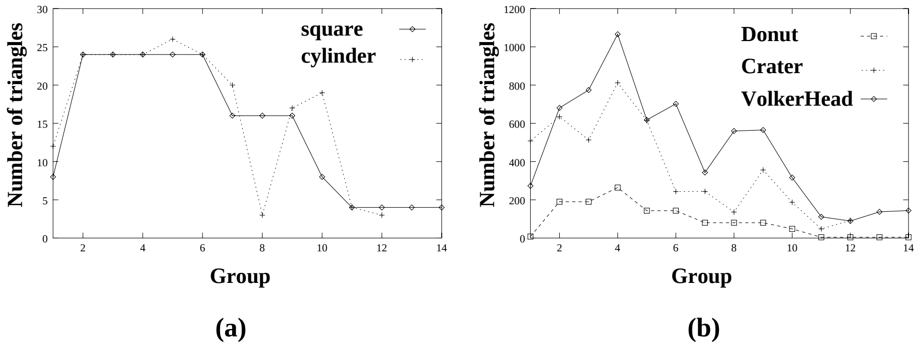


Figure 10: *Number of triangles per group (of 2 rings) (a) Original mesh (b) Subdivided mesh*

tained. On the other hand, in Figure 10(b) the number of triangles generated with the adaptive tessellation procedure are indicated. It is important to note that these results permit the estimation of the memory requirements. As we previously mentioned a greater number of rings would increase the memory size, but without important quality results.

6 Conclusions

In this paper we have presented the architecture for an additional unit of the standard graphics pipeline to enable the rendering of subdivision surfaces in hardware. Our proposal permits the adaptive subdivision of each triangle based on the neighbour information of each edge. The method is based on the grouping of neighbour triangles in the CPU to be sent once to the graphics pipeline. The architecture we propose also permits a quick access to all the required information so that no wait cycles are required for each subdivision.

To reduce the memory requirements a new mesh grouping scheme was presented. The grouping process we propose minimizes the number of exterior triangles and ensures a good covering of the full mesh at the same time. This grouping scheme could also be employed in other applications as, for example, non adaptive subdivision surface rendering.

The architecture is simple and regular, and is mainly composed of a memory and a set of FIFO queues. The efficient memory storage method employed together with the new mesh codification proposed, permits the exploitation of the temporal and spatial locality of the data. The storage is performed in such a way that all the neighbours information required for computing each edge subdivision can be accessed simultaneously. As the methodology and architecture proposed permits an efficient management of irregular meshes the scheme can be easily extended to other applications in which meshes have to be recursively computed and stored. The biggest advantage of our proposal is the development of a new memory scheme for the storage of irregular meshes in which the access to multiple neighbour information does not require wait cycles. This permits the hardware implementation of adaptive subdivision surface rendering employing neighbouring information.

Acknowledgements

This work was partially developed during the stay of Dr. M. Amor and Dr. M. Bóo in the WSI/GRIS Department, University of Tübingen, Germany (stay supported by “Secretaría Xeral de Investigación e Desenvolvemento”, Xunta de Galicia, Spain). The work has been also partially funded by the SFB grant 382 of the German Research Council (DFG). Thanks to Volker Blanz for use of the digitized model of his head.

References

[1] 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, Universität Tübingen, WSI/GRIS (http://www.gris.uni-tuebingen.de), 2000.

[2] S. Bischoff, L.P. Kobbelt, and H-P. Seidel. Towards Hardware Implementation of Loop Subdivision. In *SIGGRAPH/Eurographics Hardware Workshop*, pages 41–50, 2000.

[3] J. Cohen, M. Olano, and D. Manocha. Appearance-Preserving Simplification. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 115–122. ACM SIGGRAPH, 1998.

[4] M. Doggett and J. Hirche. Adaptive View Dependent Tessellation of Displacement Maps. In *SIGGRAPH/Eurographics Hardware Workshop*, pages 59–66, 2000.

[5] S. Gumhold and W. Strasser. Real Time Compression of Triangle Mesh Connectivity. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 133–140. ACM SIGGRAPH, 1998.

[6] I. Guskov, K. Vidimče, W. Sweldens, and P. Schröder. Normal Meshes. In *SIGGRAPH 00 Conference Proceedings*, Annual Conference Series, pages 95–102. ACM SIGGRAPH, 2000.

[7] V. Krishnamurthy and M. Levoy. Fitting Smooth Surfaces to Dense Polygon Meshes. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 313–324. ACM SIGGRAPH, 1996.

[8] A. Lee, H. Moreton, and H. Hoppe. Displaced Subdivision Surfaces. In *SIGGRAPH 00 Conference Proceedings*, Annual Conference Series, pages 85–94. ACM SIGGRAPH, 2000.

[9] C. Loop. *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, University of Utah, Dept. of Mathematics, 1987.

[10] P. Schröder and D. Zorin (Organizers). Subdivision for Modeling and Animation. In *SIGGRAPH 00 Course Notes*. ACM SIGGRAPH, 2000.

[11] D. Zorin, P. Schröder, and W. Sweldens. Interpolating Subdivision for Meshes with Arbitrary Topology. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 189–192. ACM SIGGRAPH, 1996.