Two Parallel Methods for Polygon Clipping

Theoharis Theoharis[*]
and Ian Page[‡]

Abstract

A control parallel and a novel data parallel implementation of the Sutherland-Hodgman polygon clipping algorithm are presented. The two implementations are based on the INMOS transputer and the AMT Distributed Array Processor respectively; both of these machines are general purpose parallel processors. Performance Figures are reported and implications for further work are discussed.

Keywords and phrases: polygon clipping, Multiple Instruction Multiple Data stream (MIMD) processor, Single Instruction Multiple Data stream (SIMD) processor, transputer, DAP

1. Introduction: The Transputer and the DAP

The INMOS transputer[1] and the AMT Distributed Array Processor (DAP)[2] are two very different examples of a general purpose parallel processor. They lie at opposite ends of the spectrum that defines the distribution of processing power and the organisation of control; the former offers coarse grain control parallelism and the latter fine grain data parallelism. The two machines are typical examples of a Multiple Instruction Multiple Data stream (MIMD) and a Single Instruction Multiple Data stream (SIMD) parallel processor. Although special purpose parallel architectures can provide very efficient support for some restricted class of applications, the wider applicability of a general purpose machine often makes it more cost-effective.

The transputer is a 32-bit microprocessor with on-chip local memory and 4 bidirectional bit-serial links which enable it to be connected to 4 other transputers in order to construct a MIMD system. The user physically wires transputers together using their links in order to construct networks of various topologies. The topology of a transputer network is easily modifiable; it is only a matter of specifying a **configuration** and changing the physical link connections between the transputers. Pipelines, trees, arrays and other topologies can easily be constructed out of the same transputer network according to the requirements of the task at hand.

The transputer efficiently supports the concurrent programming language Occam[3,4] which is derived from Hoare's Communicating Sequential Processes (CSP)[5]. An Occam program consists of a number of *processes*. Processes can be explicitly combined to run in parallel by means of the PAR statement. Concurrent processes can communicate via *channels*. A channel provides unidirectional communication between a pair of processes. Concurrent processes may be executed by one or more transputers and Occam allows the allocation of processes to transputers to be performed with ease. "Software" channels provide communication between pairs of processes running on the same transputer while physical transputer links implement the channels of processes running on different transputers.

Processor arrays consist of $N \times N$ *processing elements* (PE's) operating in SIMD mode and are probably the most widespread form of parallel processor. The Illiac IV, Burroughs PEPE, Goodyear Aerospace STARAN and more recently the ICL DAP, CLIP (University College, London), Burroughs BSP and MPP are representative examples of the large class of processor arrays that have been designed over the past 25 years[6]. The DAP is an $N \times N$ array of bit-serial PE's. Each PE has its own, local, bit-wide memory and is connected to its four nearest neighbours. A full adder and 3 registers are the main components of a PE (all are bit-wide). One of the registers, the activity register, has special significance; an instruction is only executed by those PE's whose activity registers contain the value TRUE. It is thus possible to control which PE's execute any one instruction by setting the activity registers as necessary, a process called masking.

The DAP is programmed in DAP Fortran[7,8] which is an extension of Fortran that offers two new data types; vector and matrix. Vector objects consist of N scalar elements while matrix objects are made up of $N \times N$ scalar elements. The dimensions of objects of the above two data types are implicit in declarations (N refers to the $N \times N$ PE's of the DAP). For example:

[*] St. Catharine's College
Cambridge CB2 1RL
U.K.

[‡] Oxford University Computing Laboratory
8-11 Keble Road
Oxford OX1 3QD
U.K.

The authors would like to express their gratitude to INMOS Ltd and Sigmex Ltd who have provided financial support for the doctoral studies of Theoharis.

```
REAL *4 A(), B(,), C(,,10)
```

declares A as a vector of N 4-byte real elements, B as a matrix of $N \times N$ real elements and C as an array of 10 matrices of $N \times N$ real elements each. The usual arithmetic operators are used for arithmetic operations between vector or matrix objects; the operations are performed in parallel for all scalar elements of the vectors or matrices. A special kind of assignment, the conditional assignment, uses a logical vector or matrix expression as a mask which enables the assignment to take place only in those elements of the l.h.s. vector or matrix for which the corresponding mask element is TRUE. For example, the statement:

```
B (B .LT. 0) = 0
```

sets all the negative elements of B to 0.

2. The Sutherland-Hodgman Polygon Clipping Algorithm

The Sutherland-Hodgman polygon clipping algorithm is best described in their paper[9]; in this section we demonstrate the key concepts involved in the algorithm which are necessary for the explanation of the parallel approaches to its implementation presented in sections 3 and 4.

The polygon to be clipped is represented as a sequence of vertices which occur in the order dictated by an anticlockwise traversal around the polygon. The polygon is clipped against (the *clipping plane* defined by) the first face of the clipping pyramid and a new sequence of vertices is produced which represents the polygon clipped against the first face of the pyramid. The process is repeated for each of the six faces of the clipping pyramid. The sequence of vertices that are produced by the last clipping stage, represents the polygon clipped against the clipping pyramid.

Clipping against a single clipping plane is performed by considering the polygon's vertices in pairs (v_i, v_{i+1}). For each such pair 0, 1 or 2 vertices are appended to the clipped polygon sequence, depending on the relationship of the vertex pair to the clipping plane. The 4 possible cases are demonstrated in Figure 1. Note that the first vertex of the polygon must also be repeated as the last.

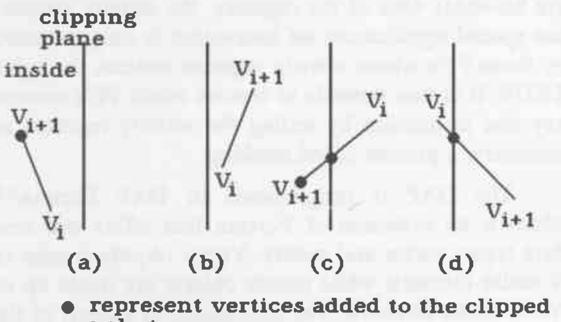


Figure 1. Clipping against a single clipping plane.

The next subsection describes how the key calculations of determining the relationship between a vertex and a clipping plane and calculating the intersection of a clipping plane with a line segment can be performed efficiently.

2.1. Key Clipping Calculations

Clipping is performed in the eye coordinate system. Let us assume that eye coordinates are homogeneous i.e. the point (x_e, y_e, z_e) is represented as a quadruplet $(w_e x_e, w_e y_e, w_e z_e, w_e)$ for $w_e \neq 0$, see §7.2 of Foley and van Dam[10]. If the viewing transformation calculates the w_e coordinate of a point (x_e, y_e, z_e) as $w_e = (S/D) z_e$, where S is half the screen height / width and D is the distance from the viewpoint to the screen plane, then the tests for a point being inside each of the six clipping planes are the following:

inside	if
top	$y_e \leqslant w_e$
bottom	$y_e \geqslant -w_e$
left	$x_e \geqslant -w_e$
right	$x_e \leqslant w_e$
hither	$z_e \geqslant z_{\mathrm{HITHER}}$
yon	$z_e \leqslant z_{\mathrm{YON}}$

where $z = z_{\mathrm{HITHER}}$ and $z = z_{\mathrm{YON}}$ are the hither and yon clipping planes respectively.

We also need to be able to determine the intersection of a clipping plane with a line segment whose two endpoints lie on either side of the clipping plane. We use the method suggested by Sutherland et al.[9]. Consider the points $P_1 = (x_{e1}, y_{e1}, z_{e1})$ and $P_2 = (x_{e2}, y_{e2}, z_{e2})$ lying on either side of the top clipping plane. The parametric equations of the line segment from P_1 to P_2 can be written as:

$$x_e = x_{e1} + \alpha(x_{e2} - x_{e1})$$
$$y_e = y_{e1} + \alpha(y_{e2} - y_{e1})$$
$$z_e = z_{e1} + \alpha(z_{e2} - z_{e1})$$

where $0 \leqslant \alpha \leqslant 1$. The value of α at the intersection of the line segment P_1 -P_2 with the top clipping plane is:

$$\alpha = (y_{e1} - w_{e1}) / ((y_{e1} - w_{e1}) - (y_{e2} - w_{e2}))$$

and if this value is substituted into the parametric equation of the line segment, we get the coordinates of the intersection. The denominator in the expression for α is guaranteed to be non-zero by the fact that P_1 and P_2 are on opposite sides of the top clipping plane. Intersections with other clipping planes can be found in a similar manner.

2.2. Scope for Parallelism

Clipping may not be the performance bottleneck of the graphics output pipeline but it requires a significant amount of computation which necessitates parallel processing especially when complex scenes have to be displayed in real-time or near real-time.

The only restriction that the Sutherland-Hodgman algorithm imposes on the order of clipping, is that a polygon must be clipped sequentially against the 6 clipping planes. In other words a polygon can not be clipped against more than one clipping plane at the same time. There are three main ways of introducing parallelism into the algorithm:

A. Different polygons can be clipped against different clipping planes at the same time, or

B. A large number of polygons can be clipped against the same clipping plane at the same time, or

C. Both A and B.

The first method (A) results in control parallelism and requires a pipeline of 6 clippers, each clipping against a different clipping plane, see §3. The second method (B) results in data parallelism and can be implemented on a SIMD processor array as shown in §4. We are not aware of any previous SIMD implementations of the Sutherland-Hodgman or of any other polygon clipping algorithm. The last method (C) involves both control and data parallelism and could be implemented on a pipeline of SIMD processors! This method will not be considered any further.

3. A Control Parallel Implementation of the Sutherland-Hodgman Algorithm

The following control parallel implementation of the Sutherland-Hodgman algorithm is not novel; Clark[11] constructed a pipeline of purpose built VLSI chips (the Geometry Engine) for this purpose and a transputer implementation of the pipeline is given in a report by Theoharis[12]. The pipeline implementation of the Sutherland-Hodgman algorithm is presented here for comparison with the data parallel implementation of §4.

3.1. Description of the Control Parallel Implementation

The Sutherland-Hodgman algorithm can easily be distributed on a pipeline of 6 processors. Each processor is assigned the task of clipping against one of the 6 clipping planes that constitute the clipping pyramid. Processor i (i: 1..6) receives as input polygons that have been clipped against clipping planes $1..i-1$ and clips them against clipping plane i (the first processor receives the unclipped polygons). The resulting polygons, now clipped against clipping planes $1..i$, are output to processor $i+1$ (the last processor outputs polygons which have been clipped against all 6 clipping planes). Figure 2 illustrates the pipeline.

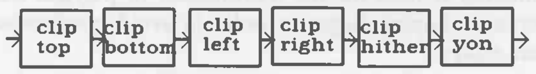


Figure 2. A clipping pipeline

We shall next describe the operation of a single clipping stage, the top one. Two sets of variables are used to store the coordinates of succesive vertices v_1 and v_2:

```
REAL32 x1, y1, z1, w1, x2, y2, z2, w2:
BOOL insidev1, insidev2:
```

The two Booleans are used to store the relationship of v_1 and v_2 to the clipping plane. For each successive pair of vertices v_1 and v_2, the following actions take place:

1. Determine the relationship of v_1 and v_2 to the (top) clipping plane:

```
insidev1 := ($y_1 \leq w_1$)
insidev2 := ($y_2 \leq w_2$)
```

Notice that only one of the above two comparisons need take place in each loop iteration.

2. If both vertices are inside the clipping plane, output v_2 to the next clipping stage:

```
IF
  (insidev1 AND insidev2)
    ... output $v_2$
```

3. If only v_1 is inside the clipping plane, output the intersection of the clipping plane with the edge $v_1 - v_2$ to the next clipping stage:

```
IF
  (insidev1 AND (NOT insidev2))
    SEQ
      ... calculate intersection of clipping
          plane with $v_1 - v_2$
      ... output intersection
```

4. If only v_2 is inside the clipping plane, output the intersection of the clipping plane with the edge $v_1 - v_2$ followed by v_2 to the next clipping stage:

```
IF
  ((NOT insidev1) AND insidev2)
    SEQ
      ··· calculate intersection of clipping
          plane with $v_1 - v_2$
      ··· output intersection
      ··· output $v_2$
```

5. Finally if neither vertex is inside the clipping plane, no action need take place.

Buffering is used for the transmission of polygon data between clipping stages in order to avoid synchronisation delays.

The 5 other clipping stages operate in a similar manner. A process is defined for each of the 6 stages:

```
PROC clip.plane (CHAN in, out)
  ··· body
```

where *in* and *out* are the channels connecting the stage to its predecessor and its successor in the pipeline respectively. The Occam PAR statement and the appropriate channel connections are used to construct the clipping pipeline:

```
CHAN $c_1, c_2, c_3, c_4, c_5, c_6, c_7$:
PAR
  clip.top ($c_1, c_2$)
  clip.bottom ($c_2, c_3$)
  clip.left ($c_3, c_4$)
  clip.right ($c_4, c_5$)
  clip.hither ($c_5, c_6$)
  clip.yon ($c_6, c_7$)
```

(a variant of the PAR statement, the PLACED PAR, is used to place the processes on different transputers).

3.2. Performance of the Control Parallel Implementation

A large number of the polygons that constitute our model will usually lie outside the clipping pyramid. It is therefore reasonable to assume that each of the clipping stages will have to deal with fewer polygons than its predecessors in the pipeline; thus the first clipping stage will determine the rate of flow of polygons through the pipeline i.e. the rate at which polygons are clipped. Let $t_{\text{CLIP.ONE.PLANE}}$ represent the cost of clipping one polygon against one clipping plane. The time to clip P polygons against the 6 clipping planes is:

$$T_{\text{CLIP}} = (P+6) * t_{\text{CLIP.ONE.PLANE}}$$

where $6 * t_{\text{CLIP.ONE.PLANE}}$ is the amount of time taken to "fill" the pipeline of 6 clipping stages and it can be ignored if P is sufficiently large. Our INMOS T414 transputer implementation resulted in $t_{\text{CLIP.ONE.PLANE}}$ = 0.5ms; about 2,000 polygons can therefore be clipped per second. Notice that the bandwidth of the transputer links is not the performance limiting factor provided that buffering is used between stages to avoid synchronisation delays. Assuming that a polygon requires 100 bytes (each vertex consists of four 4-byte coordinates), the 2,000 polygons/s require a data rate of 200 Kbytes/s which is significantly less than the bandwidth of a transputer link.

The T800 floating point transputer should provide better performance since real arithmetic calculations are used in clipping.

4. A Data Parallel Implementation of the Sutherland-Hodgman Algorithm

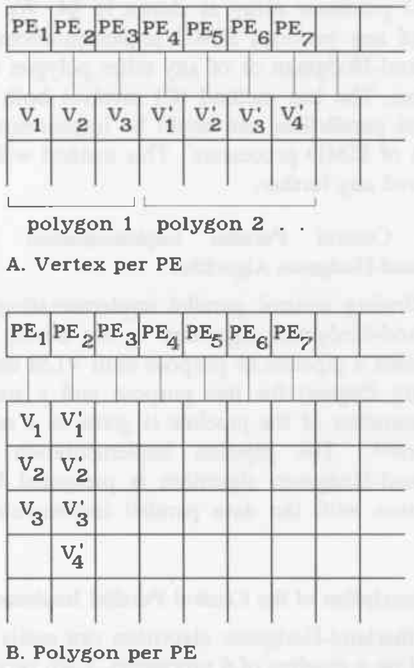
In this section we shall describe how the Sutherland-Hodgman polygon clipping algorithm can be implemented on a SIMD processor array. Our particular implementation is based on the DAP processor array.

4.1. Distribution of the Polygon Vertices

There are two basic ways of distributing the vertices of the polygons to be clipped among the PE's of the processor array:

1. Assign one vertex to each PE.
2. Assign the vertices of one polygon to each PE.

The two methods are demonstrated in Figure 3.



PE_1 PE_2 PE_3 PE_4 PE_5 PE_6 PE_7
V_1 V_2 V_3 V'_1 V'_2 V'_3 V'_4
polygon 1
polygon 2 . .
A. Vertex per PE
PE_1 PE_2 PE_3 PE_4 PE_5 PE_6 PE_7
V_1 V'_1
V_2 V'_2
V_3 V'_3
V'_4
B. Polygon per PE

Figure 3. Distributing the vertex data among the PE's.

The first method (1) has two disadvantages; first it requires interprocessor communication in order to access the adjacent vertex data necessary to clip an edge in the Sutherland-Hodgman algorithm. Second, every time a vertex is deleted or a new vertex is created during clipping, it is necessary to shift the whole array of vertices in the appropriate direction in order to close the gap or create space for the new vertex. It is also necessary to perform shifts when the second method is used, but the shifts for a whole "row" of vertices can be performed in parallel. Furthermore, as the number of vertices can increase as well as decrease during clipping, one should start with fewer vertices than the number of PE's if the first method is used. In the worst conceivable case, every edge of every polygon will intersect two clipping planes and the number of vertices will be at least doubled after clipping against all 6 clipping planes, see Figure 4. Thus we should start with less than $1/2\ N^2$ vertices, where N^2 is the number of PE's, and this would result in low processor utilisation because clipping usually reduces, rather than increases the number of vertices.

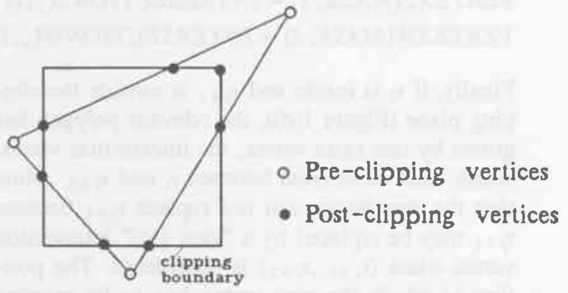


Figure 4. Clipping can increase the number of vertices (2D example)

For the above reasons we decided to use the second method of assigning vertices to PE's i.e. assign all the vertices of a polygon to a single PE. A disadvantage of this method is that if there is a wide variance on the number of vertices per polygon, some PE's will be underutilised.

4.2. Description of the Data Parallel Implementation

The data parallel polygon clipper was implemented on the 64×64 ICL DAP processor array using DAP Fortran. Before describing the details of the implementation we shall give a high level pseudo-code description in order to clarify the algorithm. In the following piece of pseudo-code variable names are in lower case. The PAR statement signifies parallel execution:

```
FOR c := 0 TO 5 {There are 6 clipping planes}
  FOR v := 0 TO
      max_number_of_vertices_in_any_polygon - 1
    PAR p := 0 TO $N^2$ - 1
          {There are $N^2$ polygons (one per PE)}
      insidev1(p) := INSIDE (clipping_plane (c),
            vertex (p,v))
      insidev2(p) := INSIDE (clipping_plane (c),
            vertex (p,v+1))
      intersection (p) :=
        FIND_INTERSECTION (clipping_plane(c),
            EDGE (vertex (p,v), vertex(p,v+1)))
      IF (NOT insidev1 (p)) AND (NOT insidev2 (p))
        THEN DELETE (vertex (p,v))
      IF (NOT insidev1 (p)) AND insidev2 (p)
        THEN vertex (p,v) := intersection (p)
      IF insidev1 (p) AND (NOT insidev2 (p))
        THEN INSERT(intersection (p),
            BETWEEN (vertex(p,v), vertex(p,v+1)))
```

Insertions and deletions do not actually take place immediately but are batched up and processed between successive executions of the outer loop (which steps through the clipping planes). We shall next describe the DAP Fortran implementation. The vertices of the N^2 polygons to be clipped are stored in 4 real matrix arrays (one for each of the 4 coordinates). An integer matrix is used to store the number of vertices in each polygon:

```
REAL *4 VERTEXX(,,10), VERTEXY(,,10),
        VERTEXZ(,,10), VERTEXW(,,10)
INTEGER *4 N(,)
```

(It is assumed that polygons can have up to 10 vertices). Four arrays of real matrices are necessary for the temporary storage of intersection vertices created during clipping and an array of logical matrices is used to indicate the positions at which new vertices must be inserted (logical is the DAP Fortran name for Boolean). Two logical matrices are needed for the storage of the relationship between pairs of vertices from each of the N^2 polygons and the clipping plane:

```
REAL *4 INTERSECTIONX(,,10),
        INTERSECTIONY(,,10),
        INTERSECTIONZ(,,10),
        INTERSECTIONW(,,10),
        ALPHA(,)
LOGICAL EXTRAVERTEX(,,10),
        INSIDEV1(,), INSIDEV2(,),
        MASK(,)
```

Initially, the first vertex of every polygon is duplicated as its last vertex and the following 3 steps take place for every I: 0..*maxvertex*−1 where *maxvertex* represents the number of vertices in the polygon with the largest number of vertices:

1. Determine the relationship between vertices v_i and v_{i+1} of every polygon and the (top) clipping plane:

   INSIDEV1 =
   VERTEXY(,, *I*).LE.*VERTEXW*(,, *I*)
   INSIDEV2 =
   VERTEXY(,, *I*+1).LE.*VERTEXW*(,, *I*+1)

   This calculation is shown in §2.1. Notice that the same operation is performed for all N^2 polygons in parallel. Also note that only one of the above two comparisons need take place in each loop iteration.

2. Calculate the intersection of the edge $v_i - v_{i+1}$ of every polygon with the (top) clipping plane:

   ALPHA = (*VERTEXY*(,, *I*) − *VERTEXW*(,, *I*)) /
   ((*VERTEXY*(,, *I*) − *VERTEXW*(,, *I*)) −
   (*VERTEXY*(,, *I*+1) − *VERTEXW*(,, *I*+1)))
   INTERSECTIONX(,, *I*) = *VERTEXX*(,, *I*) +
   (*ALPHA* * (*VERTEXX*(,, *I*+1) −
   VERTEXX(,, *I*)))
   INTERSECTIONY(,, *I*) = *VERTEXY*(,, *I*) +
   (*ALPHA* * (*VERTEXY*(,, *I*+1) −
   VERTEXY(,, *I*)))
   INTERSECTIONZ(,, *I*) = *VERTEXZ*(,, *I*) +
   (*ALPHA* * (*VERTEXZ*(,, *I*+1) −
   VERTEXZ(,, *I*)))
   INTERSECTIONW(,, *I*) =
   INTERSECTIONZ(,, *I*) * MAT (*S*/*D*)

   where *S* and *D* are the scalar constants described in §2.1 and MAT is a DAP Fortran operator which converts a scalar into a matrix. The intersection calculation is performed in parallel for the N^2 polygons although it is not useful for all of them. The denominator in the expression for *ALPHA* may be zero for some elements but the error produced in these elements should be ignored because the result of the intersection calculation will not be used in these elements; if the v_i and v_{i+1} vertices of a polygon are on opposite sides of the clipping plane, then the value of the denominator in the expression for *ALPHA* will not be zero in the relevant element.

3. Each of the N^2 vertex pairs (v_i, v_{i+1}) must now be classified according to one of the 4 cases of Figure 1. The classification produces logical matrices (masks) which are used in order to take a different action for the vertex pairs that belong to each of the 4 categories. Each of the 4 actions is implemented in parallel for all the vertex pairs of the respective category. The simplest action is taken if both v_i and v_{i+1} are inside the clipping plane (Figure 1(a)); nothing is done in this case! If both v_i and v_{i+1} are outside the clipping plane (Figure 1(b)), vertex v_i can be deleted. To this effect, one of the coordinates of v_i is replaced by a special value:

   MASK = (.NOT. *INSIDEV1*) .AND.
   (.NOT. *INSIDEV2*)
   VERTEXX(*MASK*, *I*) = MAT(*DELETE*)

   Two more cases remain to be dealt with. If v_i is outside and v_{i+1} is inside the clipping plane (Figure 1(c)), then v_i can be replaced by the intersection of edge $v_i - v_{i+1}$ and the clipping plane:

   MASK = (.NOT. *INSIDEV1*) .AND. *INSIDEV2*
   VERTEXX(*MASK*, *I*) = *INTERSECTIONX*(,, *I*)
   VERTEXY(*MASK*, *I*) = *INTERSECTIONY*(,, *I*)
   VERTEXZ(*MASK*, *I*) = *INTERSECTIONZ*(,, *I*)
   VERTEXW(*MASK*, *I*) = *INTERSECTIONW*(,, *I*)

   Finally, if v_i is inside and v_{i+1} is outside the clipping plane (Figure 1(d)), the relevant polygon has grown by one extra vertex, the intersection vertex, which must be inserted between v_i and v_{i+1}. Note that the new vertex can not replace v_{i+1} because v_{i+1} may be replaced by a "case 1(c)" intersection vertex when (v_{i+1} ,v_{i+2}) is considered. The position at which the new vertex has to be inserted must therefore be remembered for later use:

   EXTRAVERTEX(,, *I*) =
   INSIDEV1 .AND. (.NOT. *INSIDEV2*)

After performing the above 3 steps for I: 0..*maxvertex* −1, all the edges of all the N^2 polygons have been clipped against the (top) clipping plane. Before clipping against the next clipping plane it is necessary to create space for the new vertices produced in "case 1(d)" and remove the vertices marked for deletion. *EXTRAVERTEX*(,, *I*) is a logical matrix that indicates which "columns" of the vertex arrays must be shifted by one place starting from "row" $i+1$ in order to create space for the insertion of new vertices (column here refers to the vertices of a single polygon which reside within the memory of a single PE while row refers to a slice consisting of the ith vertex of every polygon, see Figure 5b). Note that the data shifting is performed in parallel for all the columns indicated by *EXTRAVERTEX*(,, *I*). Removal of the vertices marked for deletion is carried out in a similar manner. Figure 5b illustrates graphically the action of the data parallel polygon clipper for the polygons of Figure 5a.

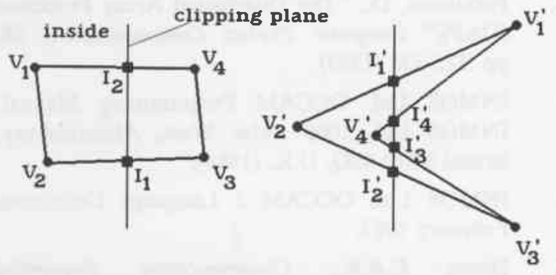


Figure 5a. Data parallel polygon clipping - example polygons

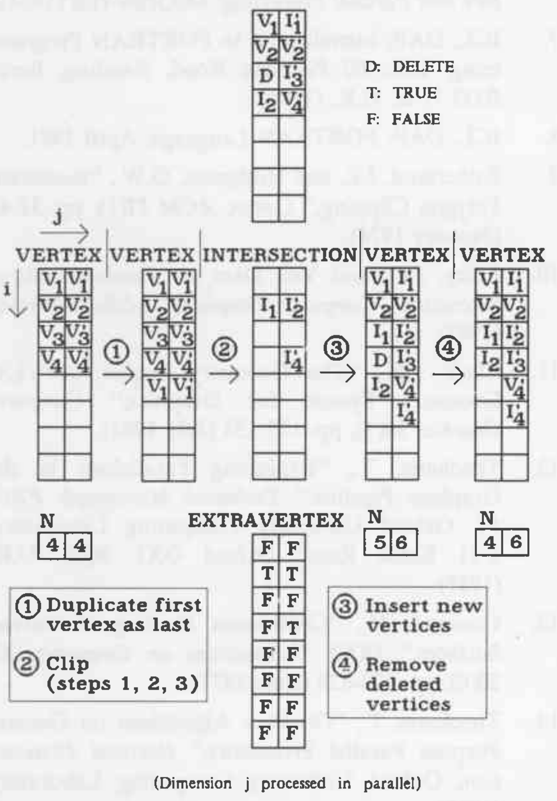


Figure 5b. Data parallel polygon clipping - example execution

4.3. Performance of the Data Parallel Implementation

The data parallel clipper described in the previous section can clip N^2 polygons against one clipping plane in parallel; let $t'_{\text{CLIP.ONE.PLANE}}$ stand for the amount of time taken to execute it. In 3D clipping we usually have to clip against 6 clipping planes (the truncated pyramid) and thus the clipper has to be executed 6 times in succession. The resulting vertex arrays from each execution will be the input to the next. The time to clip P polygons against 6 clipping planes will therefore be:

$$T'_{\text{CLIP}} = 6 * \lceil P/N^2 \rceil * t'_{\text{CLIP.ONE.PLANE}} \ .$$

Our implementation on the 64×64 ICL DAP processor array gave $t'_{\text{CLIP.ONE.PLANE}} = 25\text{ms}$. The performance of the implementation can be improved by programming the DAP in assembler rather than DAP Fortran. Also the DAP used was built in the late 1970's and has a cycle time of 200ns. Nevertheless, about 25,000 polygons can be clipped per second using the current implementation.

The data parallel clipper described above uses polygon data that has been distributed to the memories of the DAP PE's. The cost of distributing and retrieving polygon data from the DAP store can be ignored if the clipping operation is part of a data parallel implementation of the graphics output pipeline.

4.4. Usefulness of the Data Parallel Implementation

The data parallel clipper can be used in conjunction with data parallel implementations of coordinate transformations and shading calculations in order to provide an efficient data parallel implementation of the non-rendering stages of the graphics output pipeline. The viewing and perspective transformations can trivially be implemented on a SIMD processor array if the vertex data are distributed among the PE's in the manner described in §4.1 (polygon per PE). Shading calculations involve the computation of the normal to each polygon, the surface normal, the averaging of the surface normals for all polygons that have a common vertex in order to compute vertex normals and the use of a shading model to calculate vertex intensities from the vertex normals (if the Gouraud shading model is used[13]). The surface normal calculation requires the coordinates of the polygons' vertices as input and can be done data parallel since the calculation is the same for all polygons. Similarly the vertex intensity calculation can be performed data parallel once the vertex normals are known. However the calculation of the vertex normals by averaging adjacent surface normals will require the transmission of each polygon's surface normal to the PE's where its neighbouring polygons (in the polygonal model) reside. Alternatively, for rigid objects, the vertex normals can be precomputed and subjected to the same coordinate transformations as the corresponding vertices.

5. Contrasting the Two Parallel Implementations

The code fragments for the implementation of a single clipping stage on a transputer (§3.1) and on the DAP (§4.2) are quite similar. However in the case of the transputer pipeline, similar code is run on 6 different transputers each clipping a different polygon against a different clipping plane whereas in the case of the DAP, the same code is executed by all N^2 PE's in order to clip N^2 polygons against a single clipping plane in parallel. The DAP can therefore exploit more parallelism since the number of polygons that can be clipped in parallel (P) is potentially much larger than the number of clipping planes that can be processed in parallel (6). Of course it is possible to exploit data parallelism in a MIMD implementation by having a number of clipping pipelines working in parallel or by distributing the polygon data among the transputers each one of which performs all of the 6 clipping stages to its polygons.

Each of the two approaches assumes a certain environment for its effective use. In the case of the clipping pipeline, it is assumed that it is part of a larger pipeline which contains stages for coordinate transformations, rendering etc. In the case of the data parallel clipper it is assumed that it forms only one stage of a data parallel implementation of the graphics output pipeline; otherwise the polygon data distribution and collection costs are excessive. The polygon data has to be distributed among the DAP PE's anyway but it may be possible to do this once and for all by dividing the polygon data base among the memories of the PE's.

We have given some performance Figures for the implementations of the polygon clipper on the transputer and the DAP in §3.2 and §4.3 respectively. Although tempting, it is dangerous to make any direct comparisons between these Figures because of large differences in the size, vintage and cost of the two machines.

6. Conclusion

A control parallel and a novel data parallel implementation of a polygon clipping algorithm on transputers and the DAP respectively were presented and contrasted. It would be interesting to expand our work in order to include more parallel implementation strategies (e.g. data parallelism on transputers) as well as a wider range of parallel machines.

References

1. INMOS Ltd. Transputer Reference Manual, October 1986.
2. Parkinson, D., "The Distributed Array Processor (DAP)," *Computer Physics Communications* **28**, pp. 325-336 (1983).
3. INMOS Ltd. OCCAM Programming Manual, INMOS Ltd, 1000 Aztec West, Almondsbury, Bristol BS12 4SQ, U.K. (1984).
4. INMOS Ltd. OCCAM 2 Language Definition, February 1987.
5. Hoare C.A.R., *Communicating Sequential Processes,* Prentice-Hall (1985).
6. Hwang, K. and Briggs, F.A., *Computer Architecture and Parallel Processing,* McGraw-Hill (1984).
7. ICL. DAP: Introduction to FORTRAN Programming, ICL, 60 Portman Road, Reading, Berks RG3 1NR, U.K. (1980).
8. ICL. DAP: FORTRAN Language, April 1981.
9. Sutherland, I.E. and Hodgman, G.W., "Reentrant Polygon Clipping," *Comm. ACM* **17**(1), pp. 32-42 (January 1974).
10. Foley, J.D. and Van Dam, *A. Fundamentals of Interactive Computer Graphics,* Addison-Wesley (1983).
11. Clark, J.H., "The Geometry Engine: A VLSI Geometry System for Graphics," *Computer Graphics* **16**(3), pp. 127-133 (July 1982).
12. Theoharis, T., "Exploiting Parallelism in the Graphics Pipeline," *Technical Monograph PRG-54*, Oxford University Computing Laboratory, 8-11 Keble Road, Oxford OX1 3QD, U.K. (1985).
13. Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Transactions on Computers* **C-20**(6), pp. 623-628 (June 1971).
14. Theoharis, T., "Graphics Algorithms on General Purpose Parallel Processors," *Doctoral Dissertation*, Oxford University Computing Laboratory (1988).