REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1990	presentation/paper

4. TITLE AND SUBTITLE

PARALLEL COMPUTER GRAPHICS ALGORITHMS FOR THE CONNECTION MACHINE

5. FUNDING NUMBERS

In-house

6. AUTHOR(S)

J. F. Richardson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Many of the classes of computer graphics algorithms and polygon storage schemes can be adapted for parallel execution on various parallel architectures. The connection machine is one such architecture that should be thought of as a multiprocessor grid that can be reconfigured into standard 2-dimensional mesh and n-dimensional hypercube architectures. The classes of algorithms considered in this paper are SPLINES; POLYGON STORAGE; TRIANGULARIZATION; and SYMBOLIC INPUT.

Published in *Proceedings of BRL-CAD Symposium 89*, October 1989.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

PARALLEL COMPUTER GRAPHICS ALGORITHMS FOR THE CONNECTION MACHINE

JOHN F. RICHARDSON

NAVAL OCEANS SYSTEMS CENTER

1. OVERVIEW

Many of the classes of computer graphics algorithms and polygon storage schemes can be adapted for parallel execution on various parallel architectures. The connection machine is one such architecture that should be thought of as a multiprocessor grid that can be reconfigured into standard 2-dimensional mesh and n-dimensional hypercube architectures. The classes of algorithms considered in this paper are SPLINES; POLYGON STORAGE; TRIANGULARIZATION; and SYMBOLIC INPUT.

2. ARCHITECTURE OVERVIEW

The target Connection Machine (hearafter designated as CM) for the algorithms of this paper has 8192 physical processors. Each physical processor has 8 kilobytes of local memory plus an arithmetic-logic unit. All processors can communicate with any other processor through a router. Thus this CM has a shared memory of 64 megabytes when used as a standard multiprocessor (MIMD) architecture. In addition, the CM interconnection structure can simulate a 2-dimensional mesh and n-dimensional hypercube (SIMD) architecture with the mesh being the default architecture. The front end for the CM is a Symbolics and the high level language is LISP or FORTRAN.

What are the advantages of the CM for computer graphics algorithms? The primary advantage is the reconfigurability of the CM. This reconfigurability is under program control and can be initiated by LISP program statements. Within a LISP program interprocessor communication can be mixed within a program block, allowing SIMD and MIMD execution within the same algorithm. Thus the best algorithm/architecture combination can be redily optimized for a wide variety of computer graphics problems. For FORTRAN the interprocessor communication mixture is at a lower level than LISP and is mostly related to the layout of array storage, which is where the mixture takes place.

3. SPLINE ALGORITHMS

The following algorithms relate to the class of problems dealing with curve and surface interpolation. The interpolation functions discussed are the B-SPLINE functions. These functions are calculated using recurrence relations. The recurrence relations, in turn, use two parameters for curve interpolation and three parameters for surface interpolation. The recurrence relations produce what are called blending functions. These blending functions can be thought of as weight functions that indicate percentage contribution to the final interpolation value of the designated points that are being interopolated. The inverse to the interpolation problem is the production of the designated points from function values. The function values for the inverse problem can be input data values or data values derived from an algebraic equation. The designated points are called control pointes. The B-SPLINE functions have several properties. The most important property for parallel evaluation of the function values is the propoerty of local control. Essentially, each control point affects the values of the interpolating B-SPLINE within a narrow range of blending function parameter values.

3.1 B-SPLINE CONVENTIONS

NOMENCLATURE:

o = order of the B-SPLINE curve or surface.

s = number of parameter steps between integer values of u and v.

S = increment step value of parameter = 1/s.

n = number of control points to be interpolated by a 3-d curve using the blending function parameter u.

n_u = number of control points to be interpolated by a 3-d surface using the blending function parameter u.

n_v = number of control points to be interpolated by a 3-d surface using the blending function parameter v.

$N_{i,o}(u)$ = B-SPLINE blending function for the 3-d curve or surface using the blending function parameter u. (order o)

$N_{j,o}(v)$ = B-SPLINE blending function for the 3-d surface using the blending function parameter v. (order o)

t_i = knot values relating the blending function parameters u and v to the control points of the curve or surface.

$\mathbb{B}(u)$ = B-SPLINE 3-d curve.

$\mathbb{B}(u,v)$ = B-SPLINE 3-d surface.

$\mathbf{P}_i$ = control point i for the 3-d curve $\mathbb{B}(u)$.

$\mathbf{P}_{i,j}$ = control point (i,j) for the 3-d surface $\mathbb{B}(u,v)$.

RANGES OF THE PARAMETERS:

The parameter u (3-d curve): $0 \le u \le n-o+2$

The parameter u (3-d surface): $0 \le u \le n_u-o+2$

The parameter v (3-d surface): $0 \le v \le n_v-o+2$

Knots t_i ($0 \le i \le n+o$, $0 \le i \le n_u+o$, $0 \le i \le n_v+o$):

$t_i = 0$ if $i<o$

$t_i = i-o+1$ if $o \le i \le n$ or $o \le i \le n_u$ or $o \le i \le n_v$

$t_i = n-o+2$ if $i>n$

n_u-o+2 if $i>n_u$

n_v-o+2 if $i>n_v$

B-SPLINE RECURRENCE RELATIONS:

(I) $\mathbb{N}_{i,1}(u) = 1.0$ if $t_i \le u < t_{i+1}$

$= 0.0$ otherwise

(II) $\mathbb{M}_{j,1}(v) = 1.0$ if $t_j \le v < t_{j+1}$

$= 0.0$ otherwise

$$\text{(III)} \quad \mathbb{N}_{i,k}(u) = \frac{(u-t_i) * \mathbb{N}_{i,k-1}(u)}{t_{i+k-1} - t_i} + \frac{(t_{i+k}-u) * \mathbb{N}_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}}$$

$$\text{(IV)} \quad \mathbb{M}_{j,k}(v) = \frac{(v-t_j) * \mathbb{M}_{j,k-1}(v)}{} + \frac{(t_{j+k}-v) * \mathbb{M}_{j+1,k-1}(v)}{}$$

$t_{j+k-1} - t_j$ $t_{j+k} - t_{j+1}$

$$(V) \quad B(u) = \sum_{i=0}^{n} P_i * N_{i,o}(u)$$

$$(VI) \quad B(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} * N_{i,o}(u) * M_{j,o}(v)$$

3.2 SEQUENTIAL SPLINE ALGORITHMS

Here is a sequential algorithm for calculating a B-SPLINE curve:

$$A = \frac{(u-t_i) * N_{i,k-1}(u)}{t_{i+k-1} - t_i} \qquad B = \frac{(t_{i+k}-u) * N_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}}$$

```
1.  FOR 0≤u≤n-o+1  step 1/s do
2.        Find i such that t_i≤u≤t_{i+1}
3.      N_{i,1} (u) = 1.0
4.      FOR 2≤k≤o do
5.          FOR 1≤j≤k do
6.              l = i+1-j
7.              IF ( t_{i+k-1} - t_i) and ( t_{i+k} - t_{i+1}) = 0
8.                N_{i,k} (u) = 0
9.              ELSE IF ( t_{i+k-1} - t_i) and ( t_{i+k} - t_{i+1}) <> 0
10.                N_{i,k} (u) = A + B
11.             ELSE IF ( t_{i+k-1} - t_i) = 0
12.                N_{i,k} (u) = B
13.             ELSE IF ( t_{i+k} - t_{i+1}) = 0
14.                N_{i,k} (u) = A
15.             END IF
16.         END FOR
```

17. END FOR
18. FOR i+1-o≤sum≤i do
19. $\mathbb{B}(u) = \mathbb{B}(u) + \mathbf{P}_{sum} * \mathbb{N}_{sum,o}(u)$
20. END FOR
21. END FOR

The loop in step 1 is executed C = (n-o+1) * s + 1 times, where s is the number of steps between integer increments in the parameter u. C is the number of curve points calculated by the algorithm. Since there are 3 rules for knot values, n control points and 2 knot values to evaluate, step 2 is O(n). Calculating the blending function in steps 4 and 5 is O((o-1)*o). The final summation to produce the curve value is O(o). It should be noted that the value of o is for all practical purposes 3 or 4 in most computer graphics applications. Since n >> o and n >> s in general, the complexity of the sequential algorithm is O(n). The low value of o is due to considerations of the stability of interpolation algorithms in general. The value of s depends upon the number of points or polygons to be plotted. Thus, if the value of s is greater than n, then the complexity of the algorithm is $O(n^2)$ or greater.

3.3 PARALLEL SPLINE ALGORITHM DISCUSSION

The most important property of splines that facilitates parallel algorithm development is the local control property. Only a finite subset of control points affect the value of the B-SPLINE curve or surface. This property is reflected in steps 2 and 3 of the sequential algorithm. The effect of this property of local control is to reduce the number of processors needed by the parallel algorithm. Examination of steps 2 and 3 in the sequential algorithm indicate that most of the blending functions for a particular parameter u will be zero. The number of nonzero blending functions is equal to the order of the B-SPLINE curve. Blending functions $\mathbb{N}_{i,o}(u)$, $\mathbb{N}_{i-1,o}(u)$,...., $\mathbb{N}_{i-o+1,o}(u)$ are nonzero where i is the value found in step 2 of the sequential algorithm. In addition, the combination of this property and the structure of the recurrence relation for the blending functions allows for the development of a set of recurrence relations relating the final nonzero set of blending functions of order o to the blending function $\mathbb{N}_{i,1}(u)$ calculated in step 3 of the sequential algorithm. This set of recurrence relations allows for the parallel calculation of all of the blending functions for all parameter values in 1 operation. Thus, the loop in step 1 is eliminated. The loop that calculates the final sum, step 18 in the sequential algorithm, can be executed in parallel in O(o) steps. Step 2 of the sequential algorithm can be executed on the CM using 1 parallel FORTRAN or LISP statement for all values of the parameters u. In

order to reduce the number of processors required, o sets of variables are needed. The use of these extra variable sets reduces the processor requirement from O(o*s*(n+o)) to O(s*(n+o)) in the worst case. In other words, the number of processors required is approximately equal to the number of points to be plotted by the algorithms computer graphics application. The parameters u and v have no intuitive graphics function and only serve to generate points.

The following discussion relates to the development of the parallel B-SPLINE recurrence relation set. This set is developed for the case of B-SPLINE functions of order 3 which is a suitable spline order for most computer graphics applications. The B-SPLINE recurrence is a 2nd order linear recurrence. Consider the above recurrence equations III and IV. These recurrence equations are of the form:

$$N_{l,k}(u) = x^{*}N_{l,k-1}(u) + y^{*}N_{l+1,k-1}(u)$$

Now substitute for $N_{l+1,k-1}(u)$, resulting in the following:

$$N_{l+1,k-1}(u) = x_1{}^{*}N_{l+1,k-2}(u) + y_1{}^{*}N_{l+2,k-2}(u)$$

and finally

$$N_{l,k}(u) = x_2{}^{*}N_{l,k-1}(u) + y_2{}^{*}N_{l+1,k-2}(u)$$

This process could be continued until k-2 < 1 at which point the blending function subscript would be meaningless. For the B-SPLINE of order 3 only 1 iteration of this process is necessary. Due to the local control property the blending function of order k-1 is really a function of order k-2 only since it is itself a combination of blending functions of order k-1 and k-2 with the order k-1 function equal to zero. Also due to the local control property only blending functions $N_{i,o}(u)$, $N_{i-1,o}(u)$,....., $N_{i-o+1,o}(u)$ need to be calculated. The result is a set of recurrence relations of the form:

(VII) $N_{i,o}(u) = z_1{}^{*}N_{i,o-2}(u)$

(VIII) $N_{i-1,o}(u) = z_2{}^{*}N_{i,o-2}(u) + z_3{}^{*}N_{i,o-2}(u)$

(IX) $N_{i-2,o}(u) = z_4{}^{*}N_{i,o-2}(u)$

where

$$z_1 = \frac{(u - t_i)}{(t_{i+o-1} - t_i)} * \frac{(u - t_i)}{(t_{i+o-2} - t_i)}$$

$$z_2 = \frac{(t_{i-2+o} - u)}{\qquad\qquad} * \frac{(u - t_{i-1})}{\qquad\qquad}$$

$$z_3 = \frac{(u - t_i)}{(t_{i+1} - t_i)} * \frac{(t_{i-1+o} - u)}{(t_{i-1+o} - t_i)}$$

$$z_4 = \frac{(t_{i-2+o} - u)}{(t_{i-2+o} - t_{i-1})} * \frac{(t_{i-2+o} - u)}{(t_{i-2+o} - t_i)} .$$

From step 2 of the sequential algorithm, the blending function $N_{i,o-2}(u)$ is $N_{i,1}(u)$ and is equal to 1. It is obvious that the above set of recurrences for the parallel B-SPLINE algorithm does not involve blending functions of lower order and so all of the values of $N_{i,o}(u)$ can be calculated simultaneously for all values of u. Thus, the value of the B-SPLINE curve for a particular value of u can be calculated using 3 processors (for B-SPLINES of order 4 it would take 4 processors).

3.4 PARALLEL SPLINE CM ALGORITHM

First consider processor interconnection and geometry for the parallel spline 3-d curve algorithm. The geometry is a rectangular grid of dimension n+o columns by s rows. The constant s is generally smaller than n but can be arbitrarily large, although limited by the resolution of the display, which is in reality, the closest distance between displayable points.

Each processor in a column of the rectangular grid contains the blending function variables $N_{col(i),o}(u)$, $N^1_{col(i),o}(u)$ and $N^2_{col(i),o}(u)$. Three different variables are used to reduce processor usage for the following reason. The index i of the nonzero blending function set is calculated from the integer valued knot functions. Due to the use of s rows, each column of the grid is calculating the blending functions for only o integer values of u in a particular row. Therefore only o different blending variables need to be stored per processor. Consider the case of n = 7 control points and blending functions of order 3. For $0 \le u < 1$, i = 3 and only the first 3 columns participate in the calculation. For $1 \le u < 2$, i = 4 and only columns 2,3 and 4 participate in the calculation. For $2 \le u < 3$, i = 5 and only columns 3,4 and 5 participate in the calculation. For $3 \le u < 4$, i = 6 and only columns 4,5 and 5 participate in the calculation. For $4 \le u$ none of

the first 3 columns participate in the calculation. Thus each processor in a particular row and column calculates the blending function set for 3 distinct values of u.

Each processor contains a variable set $u_{j,k}$, $u^1_{j,k}$, $u^2_{j,k}$. For each processor at grid position (j,k):

$u_{j,k} = 0 + (j-1)*S$ if $k \leq o$ $\quad u_{j,k} = k-o + (j-1)*S$ if $k>o$

$u^1_{j,k} = 1 + (j-1)*S$ if $k \leq o$ $\quad u^1_{j,k} = k-o +1 + (j-1)*S$ if $k>o$

$u^2_{j,k} = 2 + (j-1)*S$ if $k \leq o$ $\quad u^2_{j,k} = k-o + 2 + (j-1)*S$ if $k>o$

plus an initialization condition to match the values to the pattern of Figure1.

For o = 4 there would be 4 variables in the variable set for u. The reason for the above distributation of $u_{j,k}$'s among the processors is similar to the explanation for the $\mathbb{N}_{col(i),o}$'s.

Each of the processors in the rectangular grid contains a variable $t_{j,k}$ where $t_{j,k}$ assumes the following values:

$t_{j,k} = 0$ if $k < o + 1$

$t_{j,k} = k - (o+1) + 1$ if $o + 1 \leq k \leq n$

$t_{j,k} = n - (o + 1) + 2$ if $n < k$

Here is the 3-d parallel spline curve algorithm for the CM:

```
0) FORALL P(j,k) , 1 ≤ j ≤ s , 1 ≤ k ≤ n DO
        INPUT (P(j,k))
   END FOR ALL
1) FORALL P(j,k) , 1 ≤ j ≤ s , 1 ≤ k ≤ n DO
        IF k ≤ o
           u(j,k) = 0 + (j-1)*S
           u1(j,k) = 1 + (j-1)*S
           u2(j,k) = 2 + (j-1)*S
        ELSE
           u(j,k) = k-o + (j-1)*S
           u1(j,k) = k-o +1 + (j-1)*S
           u2(j,k) = k-o + 2 + (j-1)*S
         IF k = 2
           u1(j,k) = 0
```

$u^2_{j,k} = 1$

IF $k = 1$

$u^1_{j,k} = 0$

$u^2_{j,k} = 0$

END FORALL

2) FORALL $\mathbb{P}_{j,k}$, $1 \le j \le s$, $1 \le k \le n+o$ DO

IF $k < o + 1$

$t_{j,k} = 0$

IF $o + 1 \le k \le n$

$t_{j,k} = k - (o+1) + 1$

IF $n < k$

$t_{j,k} = n - (o+1) + 2$

END FORALL

3) FORALL $\mathbb{P}_{j,k}$, $1 \le j \le s$, $1 \le k \le n$ DO

IF $k \le 3$ THEN $i = 3$

ELSE $i = k$

$t_1 \Leftarrow t_{j,i}$

$t_2 \Leftarrow t_{j,i+1}$

$t_3 \Leftarrow t_{j,i+o-1}$

$t_4 \Leftarrow t_{j,i+(o-1)-1}$

$t_5 \Leftarrow t_{j,i-2+o}$

$t_6 \Leftarrow t_{j,i-1}$

IF $(t_3 - t_1)$ OR $(t_4 - t_1) = 0$

$\mathbb{N}_{j,k} (u_{j,k}) = 0$

ELSE

$$\mathbb{N}_{j,k} (u_{j,k}) = \frac{(u_{j,k} - t_1)}{(t_3 - t_1)} * \frac{(u_{j,k} - t_1)}{(t_4 - t_1)}$$

IF $(t_4 - t_1)$ OR $((t_4 - t_6)$ AND $(t_2 - t_1)) = 0$

$N^1_{j,k}(u^1_{j,k}) = 0$

ELSE IF $((t_4 - t_1)$ AND $(t_4 - t_6)$ AND $(t_2 - t_1)) <> 0$

$$N^1_{j,k}(u^1_{j,k}) = \frac{(t_4 - u^1_{j,k})}{(t_4 - t_1)} * \frac{(u^1_{j,k} - t_6)}{(t_4 - t_6)} + \frac{(t_3 - u^1_{j,k})}{(t_3 - t_1)} * \frac{(u^1_{j,k} - t_1)}{(t_2 - t_1)}$$

ELSE IF $((t_3 - t_1)$ OR $(t_2 - t_1)) = 0$

$$N^1_{j,k}(u^1_{j,k}) = \frac{(u^1_{j,k} - t_6)}{(t_4 - t_6)} * \frac{(t_4 - u^1_{j,k})}{(t_4 - t_1)}$$

ELSE IF $((t_4 - t_1)$ OR $(t_4 - t_6)) = 0$

$$N^1_{j,k}(u^1_{j,k}) = \frac{(t_3 - u^1_{j,k})}{(t_3 - t_1)} * \frac{(u^1_{j,k} - t_1)}{(t_2 - t_1)}$$

IF $((t_5 - t_6)$ OR $(t_5 - t_1)) = 0$

$N^2_{j,k}(u^2_{j,k}) = 0$

ELSE

$$N^2_{j,k}(u^2_{j,k}) = \frac{(t_5 - u^2_{j,k})}{(t_5 - t_6)} * \frac{(t_5 - u^2_{j,k})}{(t_5 - t_1)}$$

END FORALL

4) FORALL $P_{j,k}$, $1 \le j \le s$, $3 \le k \le n$ DO

$C_{j,k} = P_{j,k}$

$C^1_{j,k} \Leftarrow P_{j,k-1}$

$C^2_{j,k} \Leftarrow P_{j,k-2}$

$W_{j,k} = C_{j,k} * N_{j,k}(u_{j,k})$

$W^1_{j,k} = C^1_{j,k} * N^1_{j,k}(u^1_{j,k})$

$W^2_{j,k} = C^2_{j,k} * N^2_{j,k}(u^2_{j,k})$

END FORALL

5) FORALL $\mathbb{P}_{j,k}$, $1 \le j \le s$, $3 \le k \le n$ DO

$$W^1_{j,k} \Leftarrow W^1_{j,k-1}$$

$$W^2_{j,k} \Leftarrow W^2_{j,k-2}$$

$$\mathbb{B}_{j,k} = W_{j,k} + W^1_{j,k} + W^2_{j,k}$$

END FORALL

NOTES:

1) The $\Leftarrow$ symbol indicates communication over the NEWS interconnection network of the CM. The = symbol is the standard assigment operator. The $\mathbb{P}_{j,k}$ symbol denotes the processor at position (j,k) in the rectangular grid. Also the points P_{ij} for each row index are equal (i.e. $P_{24} = P_{34}$, etc.)

2) The **FORALL** statement is in pseudocode in the algorithm body. The CM **FORALL** syntax for line 0 is:

0) FORALL (j = 1:s, k = 1:n).

The **FORALL** statement is not yet supported for the CM FORTRAN version 5.1-0.5 (June 1989). Thus the **WHILE** statement must be used with a mask instead of the subscripts. This CM mask is an array mask so an extra index array is needed. The syntax for line 0 is:

0) WHERE (1 <= J <= S . AND. 1 <= K <= n).

Note that J and K are mask arrays initialized with the rectangular grid indicies. Figure 3 gives an illustration of the grid. The J and K mask arrays can be set up using the FORTRAN DATA declaration statement.

3) All of the pseudocode variables in the above algorithm are really CM arrays when programmed in FORTRAN. The actual procedure is to create a 2-D virtual processor set. For the small examples in the paper each virtual processor is a physical processor. The variables are declared as arrays.

4) Only a n*s subgrid is computing. The extra 3 columns are used to hold the knot values for i=n.

3.5 OPERATION OF THE CM ALGORITHM

Consider the case of n = 7 and s = 4. The symbol $S_{l,o}$ indicates the blending functions developed in section 3.3. Consider the following figures. Figure 1 shows the arrangement of the values of the u parameter set variables for the first row of the processor grid. Notice that the values in the following set varaiables are equal: $u_{j,k}$, $u^1_{j,k-1}$ and $u^2_{j,k-2}$. Consider figure 2 which shows the blending function calculations for the

first row of the processor grid. Row 1 is illustrated since it serves to illustrate the principles behind the algorithm structure. These figures illustrate the rational for the program statements in step 3 of the parallel algorithm. Figure 3 shows an overview of the processor array from a theoretical standpoint. Consider the calculation of $\mathbf{B}_{24}$. Now the value of u is 1.5. The blending function set that is nozero for this parameter value consists of $S_{l,o}$, $S_{l-1,o}$, and $S_{l-2,o}$ where l = 4 and o = 3. Now, 3 processors are being used to calculate $\mathbf{B}_{24}$. Now FIGURE 1, with 2 substituted for the processor and variable subscripts, and with 1/2 added to all of the values would illustrate the parameter distribution for row 2 calculations. Remember that $\mathbb{P}_{24}$ is calculating portions of the functions required for $\mathbf{B}_{24}$, $\mathbf{B}_{25}$ and $\mathbf{B}_{26}$. The $\mathbf{B}_{24}$ calculation is involving the variable $\mathbb{N}_{24}$ (u_{24}) in $\mathbb{P}_{24}$ which corresponds to the recurrence relation (VII). It also involves the variable $\mathbb{N}^1{}_{23}$ $(u^1{}_{23})$ which corresponds to the recurrence relation (VIII) which is in $\mathbb{P}_{23}$ and the variable $\mathbb{N}^2{}_{22}$ $(u^2{}_{22})$ which corresponds to recurrence relation (IX) which is in $\mathbb{P}_{22}$. FIGURE 2 helps to illustrate the mapping between the recurrence relations (VII) , (VIII) , (IX) and the $\mathbb{N}$'s (the recurrence relation equivalent is denoted by an $\mathbb{S}$). Note that this scheme does not calculate the last u parameter value. This value can be calculated and stored in any one of the processors after by executing the above parallel algorithm with minor modifications. This last value may be needed for continuity of spline surfaces 'stiched' together. It should also be noted that the NEWS interconnection facility of the CM can not be used for the parallel curve algorithm, but that the arrays can be declared with weights using the LAYOUT directive to optimize communications between a processor and it's neighbors for a distance of 3 processors, which is the maximum needed to fetch the knot values.

$\mathbb{P}_{11}$	$\mathbb{P}_{12}$	$\mathbb{P}_{13}$	$\mathbb{P}_{14}$	$\mathbb{P}_{15}$	$\mathbb{P}_{16}$	$\mathbb{P}_{17}$
$u_{11}= 0$	$u_{12}= 0$	$u_{13}= 0$	$u_{14}= 1$	$u_{15}= 2$	$u_{16}= 3$	$u_{17}= 4$
$u^1{}_{11}= 0$	$u^1{}_{12}= 0$	$u^1{}_{13}= 1$	$u^1{}_{14}= 2$	$u^1{}_{15}= 3$	$u^1{}_{16}= 4$	$u^1{}_{17}= 5$
$u^2{}_{11}= 0$	$u^2{}_{12}= 1$	$u^2{}_{13}= 2$	$u^2{}_{14}= 3$	$u^2{}_{15}= 4$	$u^2{}_{16}= 5$	$u^2{}_{17}= 6$

FIGURE 1

$\mathbb{P}_{11}$	$\mathbb{P}_{12}$	$\mathbb{P}_{13}$	$\mathbb{P}_{14}$ .....
		$\mathbb{S}_{33}(u_{13})$	$S_{43}(u_{14})$

$\mathbb{N}_{11}(u_{11}) = z_1$ $\quad \mathbb{N}_{12}(u_{12}) = z_1$ $\quad \mathbb{N}_{13}(u_{13}) = z_1$ $\quad \mathbb{N}_{14}(u_{14}) = z_1$

$\mathbb{S}_{23}(u^1_{12})$ $\quad \mathbb{S}_{33}(u^1_{13})$

$\mathbb{N}^1_{11}(u^1_{11}) = z_2 + z_3$ $\quad \mathbb{N}^1_{12}(u^1_{12}) = z_2 + z_3$ $\quad \mathbb{N}^1_{13}(u^1_{13}) = z_2 + z_3$ $\quad \mathbb{N}^1_{14}(u^1_{14}) = z_2 + z_3$

$\mathbb{S}_{13}(u^2_{13})$ $\quad \mathbb{S}_{23}(u^2_{12})$

$\mathbb{N}^2_{11}(u^2_{11}) = z_4$ $\quad \mathbb{N}^2_{12}(u^2_{12}) = z_4$ $\quad \mathbb{N}^2_{13}(u^2_{13}) = z_4$ $\quad \mathbb{N}^2_{14}(u^2_{14}) = z_4$

$$\mathbb{B}(u_{13}) = P_{13} * \mathbb{N}_1(u_{13}) + P_{12} * \mathbb{N}^1_{12}(u^1_{12}) + P_{11} * \mathbb{N}^2_{11}(u^2_{11})$$

$$B(u_{14}) = P_{14} * \mathbb{N}_{14}(u_{14}) + P_{13} * \mathbb{N}^1_{13}(u^1_{13}) + P_{12} * \mathbb{N}^2_{12}(u^2_{12})$$

ETC.

FIGURE 2

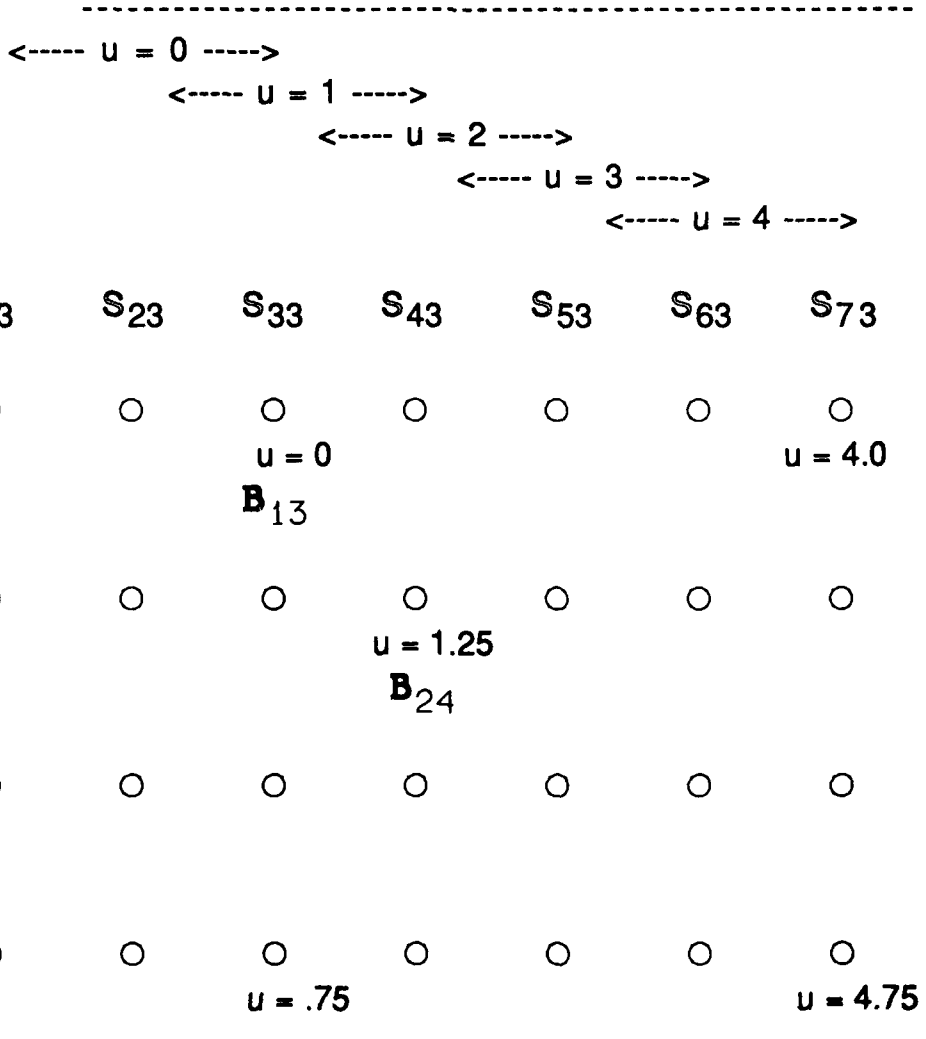


FIGURE 3 (S=.25)
GENERAL OVERVIEW

3.6 DISCUSSION OF THE SURFACE ALGORITHM

The strategy for the parallel calculation of a spline surface involves the transformation of the variables in the parallel curve algorithm into vectors. Thus, all of the required blending functions for all u and v parameter values can still be calculated simultaneously. A subset of the control point net equal to n_u x n_v will contribute to the final value of the spline surface point for a particular pair of u and v parameter values. Similarly, as in the curve parallel spline algorithm, steps 4 and 5 need to use the general CM router for communication between arbitrary processors in the grid. The control points that contribute will live in the $\mathbf{P_{j,i}}$, $\mathbf{P_{j,i-1}}$ and $\mathbf{P_{j,i-2}}$ vectors. Their positions in the vectors will be at positions f, f-1 and f-2. The i value of the vector is the index of the nonzero blending functions for the parameter n_u. The f value in the vector is the index of the nonzero blending functions for the parameter n_v. Note that vectors can be used to reduce communication needs between processors in the grid at the expense of increased storage.

Note that the length of the vectors is m (given an nxm control point grid) for the $\mathbf{P_{i,j}}$'s and 2 for the $u_{j,k}$'s and the $N_{i,j}$'s. Thus you have to place vectors of this length in each processor in the grid for the control points. For steps 4 and 5, two sets of arrays are used to store the $u_{j,k}$'s and the $N_{i,j}$ vectors. The CM allows for variables that consist of vectors to be distributed throughout the processor grid. Note that the arrays for the curve algorithm are 1 dimensional arrays set up using the CM LAYOUT compiler directive to declare the arrays parallel. For the surface algorithm the LAYOUT directive is used to develop 2 dimensional arrays. The first dimension (or AXIS) is declered SERIAL while the 2nd dimension is declared parallel (declared SEND or NEWS). For the example of the processor grid in this paper, the nxm control point grid is really nxn. Note that any array with a serial dimension is a vector allocated to each processor in the processor set.

Basically, steps 0,1,2 and 3 of the surface algorithm are similar to steps 0,1,2 and 3 of the curve algorithm in section 3.4. The main difference is in steps 4 and 5. Steps 0,1,2 and 3 just require the correct 2

dimensional LAYOUT dirrective for the $P_{i,j}$'s. Note that the u parameter value is the first index of the serial AXIS for each of the $u_{j,k}$'s, and the $\mathbb{N}_{i,j}$'s. Steps 0 thru 3 can have the assignment statements replaced with a vector assignment for each processor in the grid. Also note that the variables t_1 thru t_6 are used for both the u parameters and the v parameters. Note that since the serial dimension is 2 one can use CM arrays designated as v and $\mathbb{M}$ and duplicate the blocks containing the formulas in step 3, replacing the $u_{j,k}$'s with $v_{j,k}$'s and the $\mathbb{N}_{j,k}$'s with $\mathbb{M}_{j,k}$'s. For a control point grid of unequal dimensions, the processor grid dimensions would be based upon the larger of the control point grid dimensions. The CM ALLIGN directive would be needed to line up the $\mathbb{N}_{j,k}$'s with the $\mathbb{M}_{j,k}$'s. Different values of the parameter increment variable S could be used. For the unequal grid sizes or variable increment steps the surface algorithn would have to be modified slightly.

For steps 4 and 5, the final summations to produce the result in (VI) require 1 sequential loop. Basically, each of the $\mathbb{N}_{j,k}$'s in the processor set of three processors, is combined with 3 of the $\mathbb{M}_{j,k}$'s. Inside the loop each processor calculates a $\mathbb{B}$ (u,v) value for a fixed value of v. The parameter v is the sequential loop parameter. Here is what steps 4 and 5 look like. The variables c and d are the indexes of the processor set containing the v parameter. The v parameter progresses from 0.0 to n -o +1 in increments of 1/s. This progression is in the downward dirrection in figure 3. This produces an O(n) algorithm for the surface spline.

E = 1
FOR R = 1 to (n*s) DO
    FORALL $\mathbb{P}_{j,k}$, $1 \le j \le s$, $3 \le k \le n$ DO
        c = R mod s
        IF (c = 0) c = s
        d = R - (E)*S-1 + 2
        $T^2_{j,k} \Leftarrow \mathbb{M}^2_{c,d-2}\ (v^2_{c,d-2})$
        $T^1_{j,k} \Leftarrow \mathbb{M}^1_{c,d-1}\ (v^1_{c,d-1})$
        $T_{j,k} \Leftarrow \mathbb{M}_{c,d}\ (v_{c,d})$
        $G_{j,k} \Leftarrow P_{d,j,k}$
        $H_{j,k} \Leftarrow P_{d-1,j,k-1}$
        $I_{j,k} \Leftarrow P_{d-2,j,k-2}$
        $a_1 = I_{j,k} * \mathbb{N}^2_{j,k-2}(u^2_{j,k-2}) * T^2_{j,k}$
        $a_2 = I_{j,k} * \mathbb{N}^2_{j,k-2}(u^2_{j,k-2}) * T^1_{j,k}$

$a_3 = I_{j,k} * \mathbb{N}^2_{j,k-2}(u^2_{j,k-2}) * T_{j,k}$

$b_1 = H_{j,k} * \mathbb{N}^2_{j,k-1}(u^2_{j,k-1}) * T^2_{j,k}$

$b_2 = H_{j,k} * \mathbb{N}^2_{j,k-1}(u^2_{j,k-1}) * T^1_{j,k}$

$b_3 = H_{j,k} * \mathbb{N}^2_{j,k-1}(u^2_{j,k-1}) * T_{j,k}$

$c_1 = G_{j,k} * \mathbb{N}^2_{j,k}(u^2_{j,k}) * T^2_{j,k}$

$c_2 = G_{j,k} * \mathbb{N}^2_{j,k}(u^2_{j,k}) * T^1_{j,k}$

$c_3 = G_{j,k} * \mathbb{N}^2_{j,k}(u^2_{j,k}) * T_{j,k}$

$\mathbb{B}_{j,k} = a_1 + a_2 + a_3 + b_1 + b_2 + b_3 + c_1 + c_2 + c_3$

WRITE($\mathbb{B}_{j,k}$)

END FORALL

IF (R mod s = 0) E = E + 1

END FOR

Notes:

1. The d subscript in the control point vector variable $\mathbf{P}_{d,j,k}$ is the serial dimension. The vectors are repeated in each row of the processor grid ($\mathbf{P}_{d,j,k} = \mathbf{P}_{d,j+1,k} = \mathbf{P}_{d,j+2,k}$ .... $= \mathbf{P}_{d,s,k}$. The d subscript is called f in the above discussion. Since the $\mathbb{M}_{j,k}$'s are laid out similar to the $\mathbb{N}_{j,k}$'s the index is in the same relative position in the grid as the index for the $\mathbb{N}_{j,k}$'s.

4.0 POLYGON STORAGE AND TRIANGULARIZATION

In this section we discuss polygon storage schemes that lead to a natural and automatic triangularization of CSG primatives and also for splines. By using the CM to store 1 point of a polygon per processor, triangularization patterns can be generated for most of the polygonal CSG primatives and splines. In addition, by using spline surfaces to cover nonpolygonal CSG solids, triangularization patterns can be generated for cone, cylinder and elipse CSG primatives. For the nonpolygonal CSG primatives you just cut the spline 'skin' and unwrap it just like you unwrap the earth when rendering it on the pages of an atlas. For polygonal CSG primatives consisting of 4 to 8 points storage and triangularization requires 10 processors or less in a 2 by 5 rectangular grid. Since the triangularization patterns and information relating to adjacency of triangles in the pattern can be stored in square grid subsets of the rectangular grid, the CM NEWS interconnection structure can be used to reduce processor interconnection times. In the case of spline surfaces or the spline 'skins' of the nonpolygonal CSG primatives, storage of the points and triangularization patterns requires a C_u by C_v rectangular grid. C_u is the number of points calculated using parameter u and equals

$(n_U\text{-}o+1)*s+1$ C_V is the number of points calculated using parameter v and equals $(n_V\text{-}o+1)*s+2$(1 extra to account for the vertex duplication in the pattern to utilize the NEWS network). The construction of complex nonpolygonal or spline solids by concatenation can be accomplished by the concatenation of the spline or nonpolygonal primatives spline 'skin' and the storage grids can be placed adjacent to one another. Note that the adjacency structure of the CM grids can define a super grid for a complex solid. CM instructions that return masks or 'in use' patterns can be useful in bounding boxes type calculations for solids or groups of solids. Note that the vertex duplication is a case of trading processors for time.

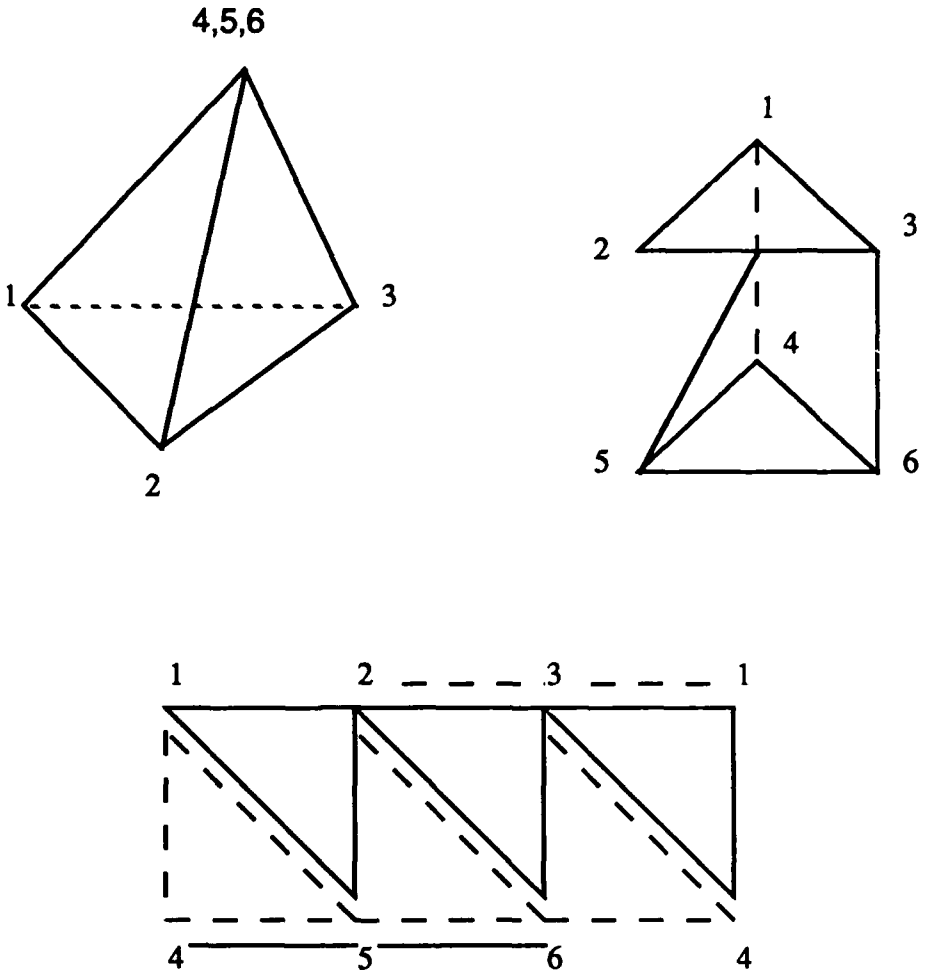
4.1 POLYGON STORAGE AND TRIANGULARIZATION ALGORITHMS

The CM can be used to automatically decompose CSG primatives and spline surfaces into triangle or rectangle components without any further additional operations other that those operations required to input the solids into the CM processors from the front end or those operations required to transfer or expand a solid from one rectangular grid to another rectangular grid. This automatic triangularization is based upon regular storage schemes for the various CSG and spline primatives which produce identifiable processor patterns within the rectangular grids which make up the primatives discussed in this paper. The primary type of complex solid is composed of spline surfaces or combinations of spline surfaces where the control points are coincident at the points of combination. Each processor in this discussion has for each polygonal solid allocated to the processor the following variables. $\mathbf{P}_{i,j}$ which is the point coordinate for a vertex of primative k. $\mathbf{S}_{i,j}$ contains k. $\mathbf{E}_{i,j}$ is an end of polygon flag used for operations on complex solids. $\mathbf{X}_{i,j}$ and $\mathbf{Y}_{i,j}$ are offsets from $\mathbf{P}_{1,1}$ and identify the i and j index of the processor at the upper left corner of the grid corresponding to primative or solid k. $\mathbf{P}_{i,j}$ contains vertex 1 of the primative. $\mathbf{X}_{i,j}$ and $\mathbf{Y}_{i,j}$ are just to flag the location of a primative inside of a large processor set containing many primatives.

Consider the following algorithm for the calculation of the plane equations of all the rectangular (equals two triangles) surface components of an n by m spline surface. Figure 10 illustrates the storage and triangularization pattern for an open spline surface of 4 by 4 surface points. All the variables are vectors. The id of the spline is k. The numbers in Figure 10 indicate the vertices of the spline rectangle components. The points of the spline triangle components are stored in $\mathbf{P}_{i,j}$, $\mathbf{P}_{i,j+1}$, $\mathbf{P}_{i,j+2}$, $\mathbf{P}_{i,j+3}$, $\mathbf{P}_{i,j+4}$ and $\mathbf{P}_{i+1,j}$, $\mathbf{P}_{i+1,j+1}$, $\mathbf{P}_{i+1,j+2}$, $\mathbf{P}_{i+1,j+3}$, $\mathbf{P}_{i+1,j+4}$. Note that the triangles can be accessed from the processors in the grid containing the vertexes connected by dashed and solid lines along the top

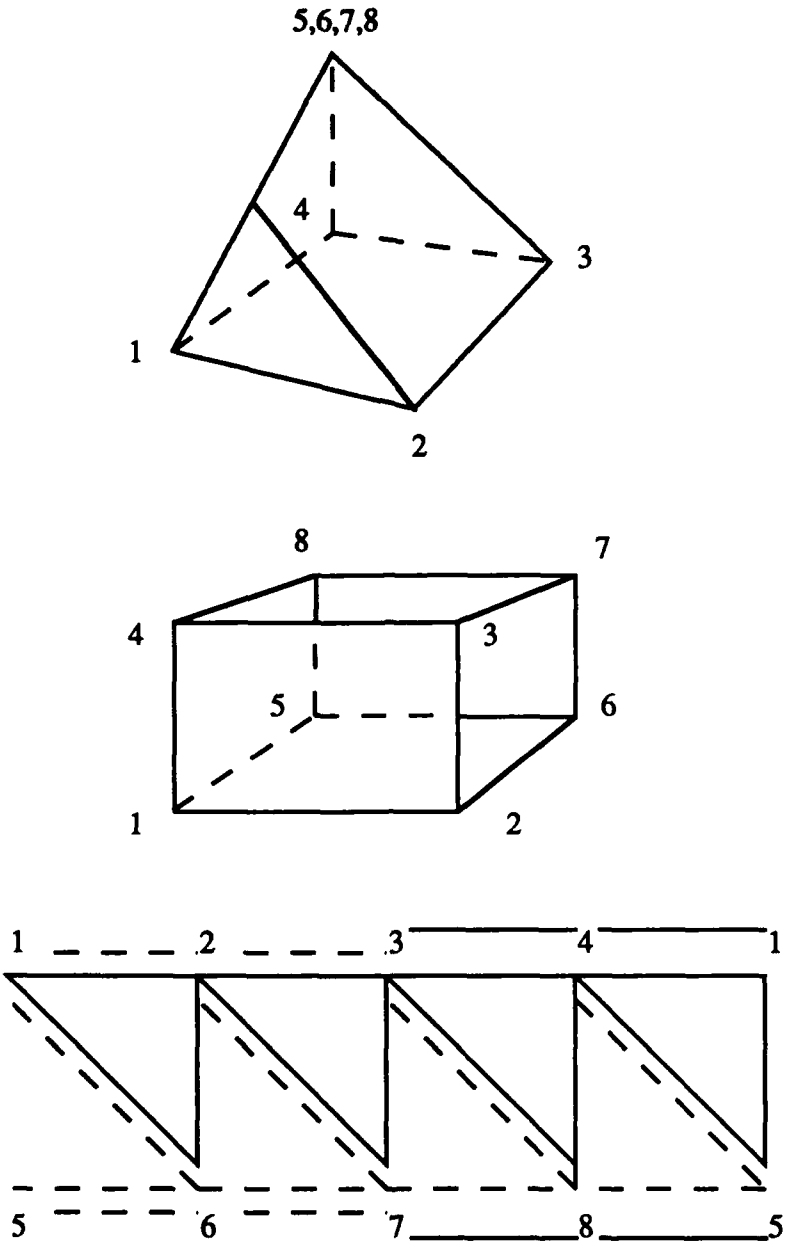
and bottom of the grid in figure 4. In this applications algorithm steps 3,4 and 5 place a triangle in each of the processors $P_{i,j+1}$, $P_{i,j+2}$, $P_{i,j+3}$, $P_{i,j+4}$ and $P_{i+1,j}$, $P_{i+1,j+1}$, $P_{i+1,j+2}$, $P_{i+1,j+3}$. It then calculates the equation of the plane containing the triangles. Note that $T^1_{i,j}$ is a vector with x,y,z coordinates of the point $P_{i,j}$, $T^2_{i,j}$ is a vector with x,y,z coordinates of the point $P_{i+1,j}$, $T^3_{i,j}$ is a vector with x,y,z coordinates of the point $P_{i,j-1}$. It is an example of the use of the NEWS network to take advantage of the automatic triangularization of a spline CSG primative. The plane equation is $Ax + By + Cz + D = 0$. $R^y_{i,j}$ is the y coordinate of the vector $R_{i,j}$. The situation is similar for $Q_{i,j}$. In general, the triangle points for the various figures will reside in the processors that are at the right angle vertex of the patterns in the grids that are associated to a particular CSG primative or complex solid. Steps 3,4 and 5 of the algorithm show the use of the NEWS network to store the triangle vectors in the processors connected with the solid lines in the grid in figure 10. Note that the processors along the sides of the grid have a bogus triangle due to the nature of the FOR ALL statement used in the algorithm. Steps 6 to 11 calculate the coefficients of the plane equation for all of the triangles of the CSG 4-point primative simultaneously. Steps 8 to 10 are calculations of the minors of the cordinate determinants. Steps 6 and 7 are straight vector operations that can take advantage of the CM vector operations instructions. Remember that the vectors are just arrays with a SERIAL dimension. The pseudocode below calculates plane equations for the solid triangles only. The calculation of the plane equations of the lower dashed triangles is similar and only one triangle is needed for the rectangle plane equation for each of the rectangle components of the surface.

1) FOR ALL $P_{i,j}$ where $S_{i,j} = k$ DO
2) $T^1_{i,j} \Leftarrow P_{i,j}$
3) $T^2_{i,j} \Leftarrow P_{i+1,j}$
4) $T^3_{i,j} \Leftarrow P_{i,j-1}$
5) $R_{i,j} = T^2_{i,j} - T^1_{i,j}$
6) $Q_{i,j} = T^3_{i,j} - T^1_{i,j}$
7) $A_{i,j} = R^y_{i,j} * Q^z_{i,j} - R^z_{i,j} * Q^y_{i,j}$
8) $B_{i,j} = -(R^x_{i,j} * Q^z_{i,j} - R^z_{i,j} * Q^x_{i,j})$
9) $C_{i,j} = R^x_{i,j} * Q^y_{i,j} - R^y_{i,j} * Q^x_{i,j}$
10) $D_{i,j} = -(A_{i,j} * Tx^1_{i,j}) + (B_{i,j} * Ty^1_{i,j}) - (C_{i,j} * Tz^1_{i,j})$
11) END FOR ALL



FOUR AND SIX POINT POLYGON STORAGE
AND TRIANGULARIZATION PATTERNS
FIGURE 4

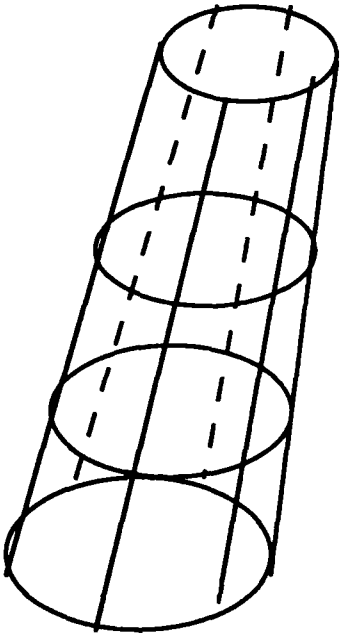
Consider the case of a five or eight point CSG primative of the form depicted in figure 5. These primatives are stored in a 2 by 5 processor grid. The points of the primative are stored in $P_{i,j}$, $P_{i,j+1}$, $P_{i,j+2}$, $P_{i,j+3}$, $P_{i,j+4}$ and $P_{i+1,j}$, $P_{i+1,j+1}$, $P_{i+1,j+2}$, $P_{i+1,j+3}$, $P_{i+1,j+4}$. Note that the triangles can be accessed from the processors in the grid containing the vertexes connected by dashed and solid lines along the top and bottom of the grid in figure 5. The above algorithm for the plane equations is applicable to this case. Figure 5 illustrates the storage pattern and triangularization pattern for the 5 and 8-point CSG primatives. Observe the grid pattern in figure 5. Figure 4 is an illustration of the triangularization patterns for four and six point CSG primatives.



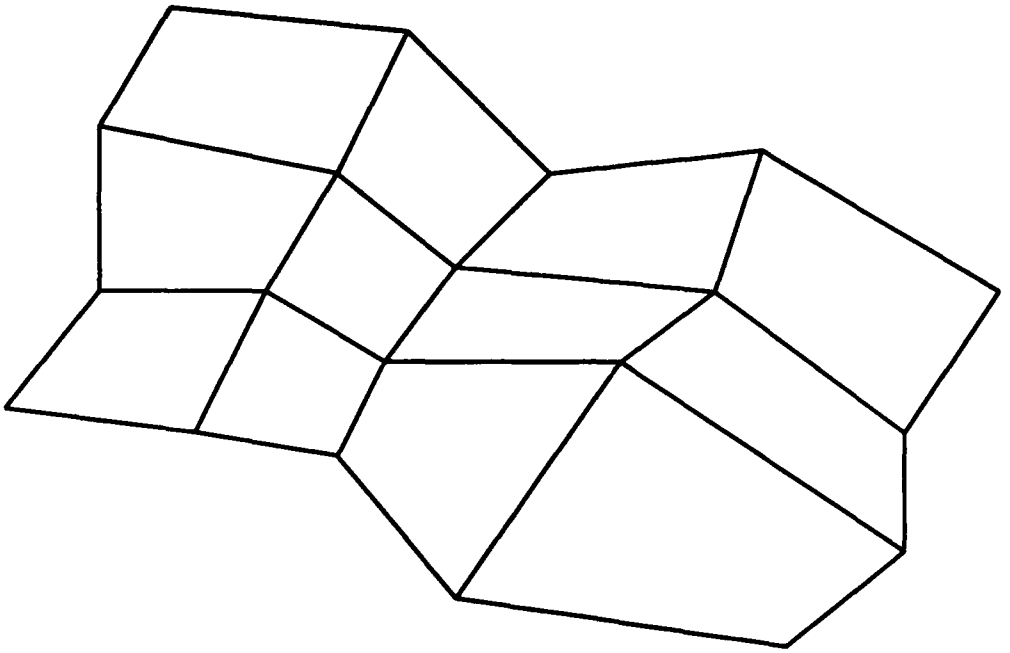
FIVE AND EIGHT POINT POLYGON STORAGE
AND TRIANGULARIZATION PATTERNS
FIGURE 5

If you unwrap the pyramid by cutting along edge containing verticies 1 and 5 you get a pattern related to the solid line triangle pattern in figure 5. If you unwrap the rectangular solid then you get a pattern related to both the solid and dashed line patterns in figure 5. Thus you can think of the CSG primatives as consisting of a spline surface. Remember that sharp angles between faces (less than 100 degrees) can be simulated by multiple control points in the patterns in figures 4 and 5. Figure 10 is an example of a closed spline surface consisting of a 4 by 4 point surface. Thus the general patterns of figures 4 and 5 can be generalized to splines of arbitrary size and shape. Figure 6 is an example of a nonpolygonal CSG

primative whose storage and triangularization pattern has been transformed to the spline storage and triangularization pattern. Thus the algorithm for plane equation calculation can be generalized to all of the CSG primatives and complex spline solids. Figure 7 is an illustration of a part of an open spline surface. When visualized as a spline surface then figure 10 is an illustration of the storage and triangularization grid for the arbitrary spline surface.



CONE WITH SIX BY FOUR CONTROL
POINT MESH OVERLAIN
FIGURE 6



FIVE BY FOUR CONTROL POINT MESH
FOR ARBITRARY B-SPLINE
FIGURE 7

Consider the combination of complex solids from simpler CSG primatives. If the components are stored according to the parallel scheme outlined above then storage of the combination is impractical within the CM processor grid. The only practical complex primative that can be stored by the above parallel scheme is the spline surface. An open spline storage and triangularization scheme is easy and is just an extension of figure 10 to include larger grid dimensions. For an open spline surface consisting of n by m points in a grid then the processor grid size required within the CM is of dimension n+1 by m. The above algorithm for the plane equation calculation is immediatelly applicable to an open spline surface. Thus any open spline can be triangularized with a resulting regular pattern to the triangularization. This pattern can be exploited by any algorithm that does calculations based upon the triangle components of a surface. A closed spline surface can utilize this storage and triangularization scheme by dissecting it into a series of open spline surfaces. Each open spline component will retain the solid identification value k. For applications which require calculations from triangle members of two or more components a parallel compare of the $T^1_{i,j}$, $T^2_{i,j}$ and $T^3_{i,j}$ values can identify the necessary processors containing the required triangles.

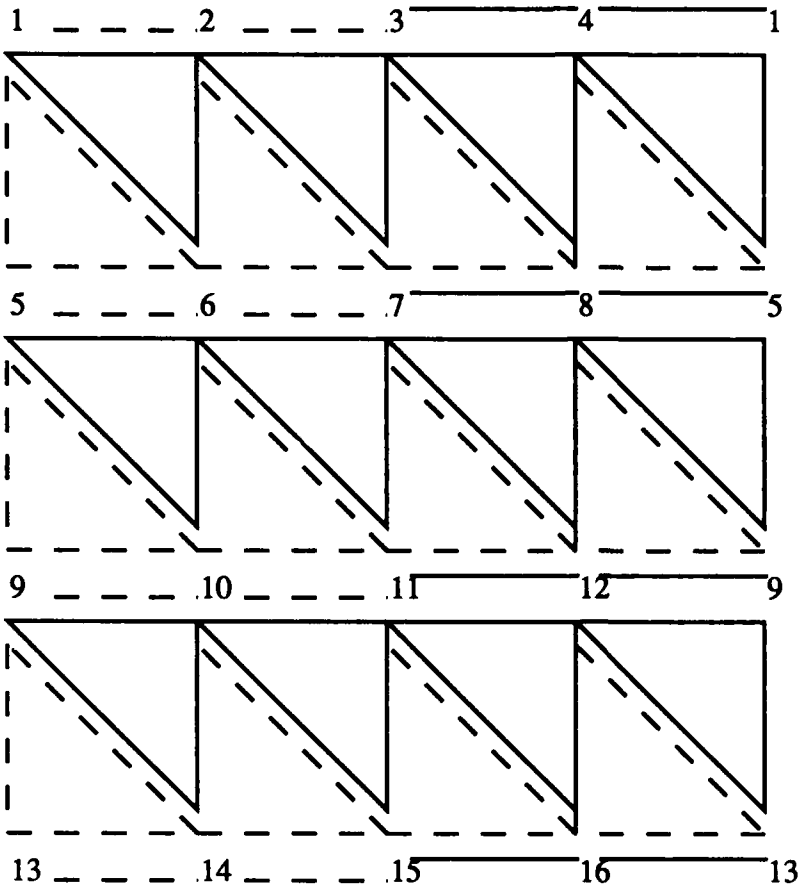


FIGURE 10

5.0 SYMBOLIC INPUT

Symbolic input is closly related to the inverse spline problem in computer graphics. Given an algebraic equation as input, the problem is to parse the equation, calculate the surface or curve points and then calculate the control point net for the algebraic surface of interest. The corresponding sequential algorithm for the inverse spline problem is from a paper by Rodgers, Satterfield, and Rodriguez[1]. A brief description of the algorithmis is as follows. Let **Q** and **B** be column matricies (vectors). Let **C** be a r by t matrix where t equals n*m and r is equal to the number of surface points. The number of control points relating to the u parameter is equal to n and the number of control points relating to the v parameter is m. **Q** contains the surface points and **B** contains the control points. Thus **Q** = **C** * **B** is the equation of the spline surface. Therefore solving for **B** we obtain:

$$(X) \quad \mathbf{B} = ((\mathbf{C}^T * \mathbf{B})^{-1} * \mathbf{C}) * \mathbf{C}^T * \mathbf{Q}.$$

Where

$$(XI) \quad \mathbf{C}_{i,j} = \sum_{j=1}^{n_v} \mathbf{N}_{i,k}(u_{i,j}) * \mathbf{M}_{j,k}(v_{i,j}).$$

The $u_{i,j}$ and $v_{i,j}$ parameters for the **C** matrix entries are calculated as follows. The surface points form a (n-o+1) by (m-o+1) grid. If stored in column major order then each collumn of points was produced by a fixed value of u and varying values of v. similarly, each row of points was produced by a fixed value of v and varying values of u. Thus, $u_{i,j}$ and $v_{i,j}$ are calculated by taking the ratios of i to n and j to m and calculating the equivalent ratios of $u_{i,j}$ and $v_{i,j}$ to the maximum values of u and v. Then the values of $u_{i,j}$ and $v_{i,j}$ can be distributed in the pattern required for a similar parallel spline algorithm for surfaces and the blending functions calculated. There exist standard algorithms for matrix multiplication which are of O(logn). Matrix transposition algorithms exist which are of O(n) for a n by n matrix. An algorithm exists [2] for the solution of a block diagonal system that is of O(nlogm). The CM has instructions for matrix multiplication and matrix transposition. Using the algorithm of [2] and the CM matrix instructions, **B** can be calculated in O(nlogm). There exists an algorithm for the calculation of the control point net for a m by n bicubic B-spline patch of O(logm + logn)[3]. The O(nlogm) estimate assumes that the CM instructions are of the same order as the software algorithms reported in the literature.

For the problem of parsing the original arithmetic input to produce the surface values input to the inverse spline algorithm there are plenty of algorithms for parallel parsing based upon the standard sequential infix to postfix algorithm of Horowitz and Sahni. Most of these algorithms are of time complexity O(logn), with n equal to the length of the infix expression. The number of processors required is of the order O(n).

REFERENCES:

1) Rodgers, D. F., and Satterfield, S. G., and Rodriguez, F. A. Ship Hulls, B-splines and CAD/CAD. IEEE Computer Graphics and Applications. December 1983.

2) Cheng, F., and Goshtasby, A. B-spline surface interpolation using SLOR method with parallel relaxation. Tech. Rep. 96-87, Dept. of Computer Science, University of Kentucky, Lexington, 1987.

3) Cheng, F. and Goshtasby, A. A parallel B-spline surface fitting algorithm. ACM Transactions on Graphics. January 1989.