A PARALLEL LINE CLIPPING ALGORITHM AND ITS IMPLEMENTATION

Fuhua Cheng
Department of Computer Science, University of Kentucky, Lexington, Kentucky 40506

Yue-Kwo Yen
Chung-Shang Institute of Science and Technology, Lungtang, Taiwan, R. O. C.

Abstract: A parallel line clipping algorithm and its implementation on a parallel clipping hardware environment are presented. We first develop a simple theory to show that parallel clipping is possible for all types of line segments. We then present the architecture of a hardware environment based on which parallel clipping is to be implemented. The parallel line clipping algorithm and its implementation on the parallel clipping hardware environment are presented finally. Based on our approach, only 141 cycles are required to clip a line segment. The corresponding figure for the famous J. Clark's Geometry Engine is 160 cycles.

Keywords: Clipping, parallel, pipeline, geometry processor, geometry system

1. INTRODUCTION

All the display devices have limited display areas. It is impossible, in general, to display all the objects defined by the user in a single screen. The graphics system has to be informed explicitly which portion of the scene is to be viewed. This visible region, specified by the user, is called the window (view volume, in 3D graphics). Data not contained in the visible region should be discarded to avoid overflow of the internal registers of the display device. The process of removing the portions of an image that lie outside the visible region is called clipping [5].

Two kinds of clipping process have been frequently used in graphics, namely, line clipping and polygon clipping. For images composed of straight line segments, only line clipping is involved. Each line segment of the image is clipped against the window (view volume, in 3D graphics), only the *visible segment*, i.e., the portion of the line segment that lies inside the window, is output for display. If an image comprises not only straight line segments but also polygons, then polygon clipping is required. In this case not only the visible segment of each edge of a polygon has to be output, sometimes additional vertices have to be output also to make the output polygon correct.

Several well known line clipping and polygon clipping algorithms have been proposed (see, e.g., [5],[7],[8],[10],[11]). Some of them have been implemented in hardware (see, e.g., [3],[9]). The basic idea used in [5], [10], and [11] ([3] and [9] as well) can be described as follows. If, say, 2D graphics is considered, then the given line segment or polygon is clipped against each boundary line of the window separately. For each boundary line of the window we first divide the plane by the boundary line into two sides, *visible* and *invisible*. The half plane which contains the window is called the visible side. The given line segment or polygon is clipped against this boundary line by discarding the portions of the line segment or polygon that lie in the invisible side of the boundary line using techniques such as *outcodes*, *midpoint subdivision* or *in-out* relationship. The remaining portions (if there are any) are then clipped against the other boundary lines of the window using the same technique. After the given line

segment or polygon has been clipped against all the boundary lines of the window then the remaining portions are the visible portions of the given line segment or polygon. They are then output for display. Since the clipping of a line segment or polygon against a boundary line has to be finished first before they can be clipped against another boundary line, this approach can actually be considered as clipping of a line segment or polygon against several lines in sequential order.*

The basic approach used in [7] and [8] is different from that of the above ones. The given line segment or each edge of the given polygon is expressed in parametric form first. Then parameters of the intersections of the straight line, defined by the parametric equation, with the boundary lines of the window are computed and compared together with 0 and 1 to determine if there is a visible segment. If there is a visible segment then its endpoints are computed. If polygon clipping is considered then sometimes turning vertices [8] also have to be found to make the output correct. Since the process of computing the parameters of the intersections does not depend on any particular order, they can be computed simutaneously. Therefore, this approach provides us with a possibility of clipping a line segment or polygon against several lines in parallel if the subsequent comparison process can be performed correctly.

In this paper we shall review the idea presented in [7], and develop a more general theory of line clipping for all types of line segments, such as vertical or horizontal line segments. We shall focus on 2D line clipping only. However, our approach can readily be extended to cover 3D line clipping too. We then present the architecture of a hardware environment based on which parallel line clipping is to be performed and present a parallel line clipping algorithm for this hardware environment. According to our algorithm, only 141 cycles are required to clip a line segment. The corresponding figure for the famous J. Clark's Geometry Engine is 160 cycles. Therefore, our approach outperforms J. Clark's system by about 15%. Our algorithm can be extended to include parallel polygon clipping as well [2]. However, it is not within the scope of this paper.

The remainder of this paper is organized as follows. In Section 2 we will review some basic concepts and develop a general theory of line clipping. The architecture of a hardware environment based on which parallel line clipping is to be implemented is presented in Section 3. In Section 4 we will present a parallel line clipping algorithm for the hardware environment. The computational complexity of the algorithm is shown in Section 5. Finally, in Section 6, we will make some Concluding remarks.

2. BASIC IDEA

Let (x_{left}, y_{bottom}) and (x_{right}, y_{top}) be the lower-left and upper-right corners of a window (Fig. 1). A line segment with endpoints (x_0, y_0) and (x_1, y_1) can be represented in parametric form as follows

$$\begin{cases} x = x_0 + \Delta x * t \\ y = y_0 + \Delta y * t \end{cases} \tag{2.1}$$

where $\Delta x = x_1 - x_0$, $\Delta y = y_1 - y_0$ and $0 \le t \le 1$. Any point of the line segment which is inside the window must satisfy the following inequalities

*The original idea of Weiler and Atherton [11] was to trace around the border of the polygon only once with respect to all the boundary lines of the window. However, it can also be classified into this category if finding the intersections of the polygon with the window is done for each of the boundary lines of the window separately.

Fig. 1. A window and a given line segment.

$$x_{left} \le x_0 + \Delta x * t \le x_{right}$$
$$y_{bottom} \le y_0 + \Delta y * t \le y_{top}$$

or, equivalently

$$-\Delta x * t \le x_0 - x_{left}, \qquad \Delta x * t \le x_{right} - x_0$$
$$-\Delta y * t \le y_0 - y_{bottom}, \qquad \Delta y * t \le y_{top} - y_0.$$

These inequalities can be written as

$$P_i * t \le Q_i, \qquad i = 1, 2, 3, 4 \tag{2.2}$$

where

$$\begin{aligned} P_1 &= -\Delta x; \quad & Q_1 &= x_0 - x_{left} \\ P_2 &= \Delta x; \quad & Q_2 &= x_{right} - x_0 \\ P_3 &= -\Delta y; \quad & Q_3 &= y_0 - y_{bottom} \\ P_4 &= \Delta y; \quad & Q_4 &= y_{top} - y_0. \end{aligned} \tag{2.3}$$

Now if $P_i \neq 0$ for all $1 \le i \le 4$ and if we define

$$t_i = \frac{Q_i}{P_i}, \qquad i = 1, 2, 3, 4 \tag{2.4}$$

then t_1 and t_2 represent the parameters of the intersections of the straight line defined by (2.1) ($-\infty < t < +\infty$) with the left boundary line $x = x_{left}$ and the right boundary line $x = x_{right}$ of the window, respectively, and t_3 and t_4 represent the parameters of the intersections of the straight line defined by (2.1) with the bottom boundary line $y = y_{bottom}$ and the top boundary line $y = y_{top}$ of the window, respectively (Fig. 1). Since the value of each t_i is independent of the other t_j's ($j \neq i$), the values of t_i's can be computed in parallel. It was shown in [7] that if $P_i \neq 0$ for all i then the necessary and sufficient condition for the given line segment to have a visible segment is

$$t_0' \leq t_1' \tag{2.5}$$

where

$$\begin{aligned} t_0' &= \max(\{t_i = \frac{Q_i}{P_i} \mid 1 \leq i \leq 4,\ P_i < 0\} \cup \{0\}) \\ t_1' &= \min(\{t_i = \frac{Q_i}{P_i} \mid 1 \leq i \leq 4,\ P_i > 0\} \cup \{1\}). \end{aligned} \tag{2.6}$$

If (2.5) is true then the coordinates of the endpoints of the visible segment are

$$\begin{aligned} x_0' &= x_0 + \Delta x * t_0'; \quad & y_0' &= y_0 + \Delta y * t_0' \\ x_1' &= x_0 + \Delta x * t_1'; \quad & y_1' &= y_0 + \Delta y * t_1'. \end{aligned} \tag{2.7}$$

Again, t_0' and t_1', and then x_0', x_1', y_0', and y_1' can all be evaluated in parallel. However, if $P_i = 0$ for some i, i.e., the given line segment is vertical or horizontal, then the above argument does not work any longer. In order to include vertical and horizontal line segments as well, we have to modify (2.3), (2.4) and (2.6) in the following way.

First, observe that if we define P_i' the following way

$$P_1' = P_2' = \Delta x; \quad P_3' = P_4' = \Delta y$$

then t_i defined in (2.4) can be written as

$$t_i = \frac{(-1)^i Q_i}{P_i'}, \quad i = 1, 2, 3, 4 \tag{2.8}$$

if $P_i' \neq 0$. In case $\Delta x = 0$ or $\Delta y = 0$, say $\Delta x = 0$, i.e., the given line segment is vertical, we first think of it as an oblique line segment by replacing the x-coordinate of the endpoint (x_0, y_0) by $x_0 - \varepsilon$, where ε is a small positive number (Fig.2). Since, in this case, $P_1' = P_2' = \varepsilon$, it follows from (2.8) that

$$t_1 = \frac{-Q_1}{\varepsilon}, \quad t_2 = \frac{Q_2}{\varepsilon} \tag{2.9}$$

and

Fig. 2. A vertical line segment and the oblique line segment generated from it.

$$t_3 = \frac{-Q_3}{P_3'}, \qquad t_4 = \frac{Q_4}{P_4'}.$$

Then simply take the limits of (2.9) when ε approachs 0 to get the corresponding t_1 and t_2 for the original line segment.

$$t_1 = \lim_{\varepsilon \to 0} \frac{-Q_1}{\varepsilon}, \qquad t_2 = \lim_{\varepsilon \to 0} \frac{Q_2}{\varepsilon}$$

This approach works when $\Delta y = 0$ also. In this case, we first replace the y-coordinate of the endpoint (x_0, y_0) by $y_0 - \varepsilon$ $(\varepsilon > 0)$ and then take the limits of $-Q_3/\varepsilon$ and Q_4/ε when ε approaches 0 to get the corresponding t_3 and t_4 for the given horizontal line segment, i.e.,

$$t_3 = \lim_{\varepsilon \to 0} \frac{-Q_3}{\varepsilon}, \qquad t_4 = \lim_{\varepsilon \to 0} \frac{Q_4}{\varepsilon}.$$

Therefore, if $P_i = 0$, t_i can simply be defined as

$$t_i = \lim_{\varepsilon \to 0} \frac{(-1)^i Q_i}{\varepsilon} \tag{2.10}$$

where ε is a positive number. Its value is either $+\infty$ or $-\infty$, depending on the sign of $(-1)^i Q_i$. Consequently, by combining (2.4) and (2.10), we get a general form for t_i as follows.

$$t_i = \begin{cases} Q_i/P_i, & P_i \neq 0 \\ -\infty, & P_i = 0 \text{ and } (-1)^i Q_i < 0 \\ +\infty, & P_i = 0 \text{ and } (-1)^i Q_i > 0. \end{cases} \tag{2.11}$$

Note that if the given line segment is vertical, i.e., $\Delta x = 0$, then P_1 is always considered having negative sign and P_2 having positive sign; if the given line segment is horizontal, i.e., $\Delta y = 0$, then P_3 is always considered having negative sign and P_4 having positive sign. Therefore, we can define a variable, $SIGN(i)$, representing the sign of P_i for each i as follows

$$SIGN(i) = \begin{cases} sign(P_i), & P_i \neq 0 \\ (-1)^i, & P_i = 0 \end{cases} \tag{2.12}$$

where

$$sign(x) = \begin{cases} +1, & x > 0 \\ -1, & x < 0 \end{cases}$$

and define t_0' and t_1' the following way

$$\begin{aligned} t_0' &= \max(\{\, t_i \mid 1 \le i \le 4,\ SIGN(i) < 0\} \cup \{0\}) \\ t_1' &= \min(\{\, t_i \mid 1 \le i \le 4,\ SIGN(i) > 0\} \cup \{1\}) \end{aligned} \tag{2.13}$$

where t_i is defined in (2.11) and $SIGN(i)$ is defined in (2.12). Then for any given line segment defined in (2.1), simply use (2.3), (2.11), (2.12) and (2.13) to find t_0' and t_1' and then use (2.4) to determine if it has a visible segment. Note that in this case (2.11), (2.12) and (2.13) can all be implemented in parallel too.

It should be pointed out that the above method can readily be extended to 3D line clipping if an orthogonal view volume is considered. This is done the following way. If (x_0, y_0, z_0) and (x_1, y_1, z_1) are the endpoints of a 3D line segment and an orthogonal view volume is defined by the six boundary planes:

$$x = x_{left},\ \ x = x_{right},\ \ y = y_{bottom},\ \ y = y_{top},\ \ z = z_{front},\ \ z = z_{back}$$

then except those defined in (2.3), we also need the following quantities

$$\begin{aligned} P_5 &= -\Delta z, & Q_5 &= z_0 - z_{front} \\ P_6 &= \ \ \Delta z, & Q_6 &= z_{back} - z_0 \end{aligned}$$

where $\Delta z = z_1 - z_0$. But then everything else is similar to the 2D case.

3. THE ARCHITECTURE OF THE HARDWARE ENVIRONMENT

The parallel clipping hardware environment which we are going to present here is actually a subsystem of a geometry system called PIPA [2]. PIPA is a PIpeline-architectured and PArallel-hardwired geometry system consists of three pipelined subsystems: Matrix

Subsystem, Clipping Subsystem and Scaling Subsystem. Each subsystem is composed of a certain number of Geometry Processors (GPs) [4] arranged in a way so that matrix multiplication, clipping and scaling can all be performed in parallel.

Basically, each GP is a four-component vector function unit implemented on a VLSI chip. It can be functionally divided into four subunits, as shown in Fig. 3.

(1) Microprogrammed Controller:
    □ Sequencer
    □ Microprogram Control Store
(2) Array Processing Unit:
    □ 4*ALU (Arithematic Logic Units)
(3) I/O Buffer
(4) Registers
    □ 4*LR (Local Registers)
    □ 4*TR (Temporary Registers)
    □ 4*CR (Curve Registers)

Fig. 3. Layout of a Geometry Processor.

The Array Processing Unit deserves special attention. It contains four parallel processing Arithematic Logic Units (ALUs). Each ALU can operate on either 32-bit floating points or 24-bit integers [6]. In floating point arithematic, the processing is divided into two parts, namely, exponent and mantissa. These four microprogrammed ALUs can do parallel addition, subtraction and two-variable operations on either the mantissa or the exponents, i.e., two-operand floating point operations.

The twelve registers in each GP are grouped into three categories: Local Registers, Temporary Registers and Curve Registers. Local Registers are used to store more sensitive values which will not be erased after each pipeline cycle. Curve Registers are used to store scaling factors or coordinates of the window or the viewport. Temporary Registers, as indicated by their name, are used to store temporary intermediate values in each operation.

The number of GPs required in each subsystem depends on the function of the subsystem. The Matrix Subsystem, performing parallel matrix multiplication and forward curve generation, requires four GPs. The Clipping Subsystem, performing parallel line clipping and polygon clipping, requires four GPs for 2D clipping and requires another two GPs if 3D clipping is needed also. The Scaling Subsystem, performing window-to-viewport mapping and generating monographic or stereographic views, requires only one GP. The operation within

each subsystem is controlled by the Preprocessor and Controller. Pipelining between subsystems is controlled by the Controller. Fig. 4 shows the layout and data flow of a PIPA.

Fig. 4. Layout and data flow of a PIPA.

We shall use only four GPs in this paper to illustrate the implemenation of the parallel line clipping algorithm. The four GPs in the Clipping Subsystem are numbered GP5, GP6, GP7 and GP8. For each GPi, $5 \le i \le 8$, contained in the Clipping Subsystem, its Local Registers, Temporary Registers and Curve Registers will be named $LR_{i,j}$, $TR_{i,j}$ and $CR_{i,j}$ ($1 \le j \le 4$), respectively. For instance, $LR_{5,3}$ represents the third Local Register of $GP5$. We shall assume that the contents of these registers within different GPs of the Clipping Subsystem can be exchanged during each pipeline cycle.

Data to be input from the Matrix Subsystem to the Clipping Subsystem are stored in the four I/O Buffers $Buf_{i,1}$, $1 \le i \le 4$, of the Matrix Subsystem. Since the matrix multiplication executed in the Matrix Subsystem is performed in the homogeneous coordinate system, four coordinates are presumably to be output to the Clipping Subsystem. However, in order to fix the coordinates of the window or view volume and, consequently, simplify the work of Clipping Subsystem, coordinates output from Matrix Subsystem will be normalized by the Matrix Subsystem first before they are put into the I/O Buffers. This means that coordinates in the Normalized Viewing Coordinate System (NVCS) are actually used in the Clipping Subsystem. Therefore, only the contents of $Buf_{i,1}$, $1 \le i \le 3$, will be needed for the Clipping Subsystem if 3D clipping is desired. In our case, since only 2D clipping is considered, only the contents of $Buf_{1,1}$ and $Buf_{2,1}$ will be needed for the Clipping Subsystem.

4. THE ALGORITHM

The parallel algorithm to be implemented on the above Clipping Subsystem will be presented in this section. In this algorithm, a block of instructions will be called a *parallel step* if it is bounded above by the key word **PARDO** and bounded below by the key word **DOPAR.** Any instructions contained in a parallel step are intended to be executed in parallel. A parallel

step is said to be at the GP level if the instructions contained in this step are expected to be executed for each GP. It is said to be at the register level if the instructions within this step are designated to certain registers of a particular GP. If a parallel step is expected to be executed at the GP level then the statement "for processor GP5,GP6,GP7,GP8" is usually attached to the key word **PARDO.** For instance, the following parallel step indicates : for each GPi, $5 \le i \le 8$, load register $LR_{i,2}$ with the contents of $LR_{i,1}$.

PARDO for processor GP5,GP6,GP7,GP8
    $LR_{i,2} = LR_{i,1}$
DOPAR

If it can be clearly inferred from the context that a parallel step is expected to be executed at the GP level then the statement "for processor GP5,GP6,GP7,GP8 " is omitted. For instance, the following parallel step is at the GP level. It indicates : load registers $LR_{5,1}$ and $LR_{6,1}$ with the contents of $Buf_{1,1}$ and load registers $LR_{7,1}$ and $LR_{8,1}$ with the contents of $Buf_{2,1}$, simultaneously.

PARDO
    $LR_{5,1}$, $LR_{6,1} = Buf_{1,1}$
    $LR_{7,1}$, $LR_{8,1} = Buf_{2,1}$
DOPAR

In a parallel step, if the operations to be executed by each GP are different then we list the instructions designated to a particular GPi under the statement "[for processor GPi]" as follows and any instructions within this block are then considered at the register level for this particular GP.

PARDO for processor GP5,GP6,GP7,GP8
    [for processor GP5]
        .....

    [for processor GP6]
        .....

    [for processor GP7]
        .....

    [for processor GP8]
        .....

DOPAR

Any instructions not enclosed by the key words **PARDO** and **DOPAR** are assumed to be executed in sequential order. The algorithm is given as follows.

Algorithm PLC: Parallel Line Clipping

1. [Parallel loading of the x- and y-coordinates]

PARDO

$LR_{5,1}$, $LR_{6,1} = Buf_{1,1}$ {x-coordinate in NVCS}

$LR_{7,1}$, $LR_{8,1} = Buf_{2,1}$ {y-coordinate in NVCS}

DOPAR

2. [Compute Δx and Δy and then t_i for each i]

PARDO for processor $GP5, GP6, GP7, GP8$

2-1. [for processor $GP5$]

2-1.1.

PARDO

$LR_{5,3} = LR_{5,1} - LR_{5,2}$ {Compute Δx}

$LR_{5,4} = LR_{5,2} - x_{left}$ {Compute Q_1}

DOPAR

2-1.2.

if $(LR_{5,3} = 0)$ then {Vertical line segment}

begin

if $(LR_{5,4} > 0)$ then

$TR_{5,1} = -\infty$

else

$TR_{5,1} = +\infty$

$sign(LR_{5,3}) = +$

end

else

$TR_{5,1} = -(LR_{5,4}/LR_{5,3})$ {$t_1 = -Q_1/\Delta x$}

2-2. [for processor GP6]

2-2.1.

PARDO

$LR_{6,3} = LR_{6,1} - LR_{6,2}$ {Compute Δx}

$LR_{6,4} = x_{right} - LR_{6,2}$ {Compute Q_2}

DOPAR

2-2.2.

if $(LR_{6,3} = 0)$ then {Vertical line segment}

if $(LR_{6,4} > 0)$ then

$TR_{6,1} = +\infty$

else

$TR_{6,1} = -\infty$

else

$TR_{6,1} = -(LR_{6,4}/LR_{6,3})$ {$t_2 = Q_1/\Delta x$}

2-3. [for processor GP7]

2-3.1.

PARDO

$LR_{7,3} = LR_{7,1} - LR_{7,2}$ {Compute Δy}

$LR_{7,4} = LR_{7,2} - y_{bottom}$ {Compute Q_3}

DOPAR

2-3.2.

if $(LR_{7,3} = 0)$ then {Horizontal line segment}

begin

if $(LR_{7,4} > 0)$ then

$TR_{7,1} = -\infty$

else

$TR_{7,1} = +\infty$

$sign(LR_{7,3}) = +$

end

else

$TR_{7,1} = -(LR_{7,4}/LR_{7,3})$ $\{t_3 = -Q_3/\Delta y\}$

2-4. [for processor GP8]

2-4.1.

PARDO

$LR_{8,3} = LR_{8,1} - LR_{8,2}$ {Compute Δy}

$LR_{8,4} = y_{top} - LR_{8,2}$ {Compute Q_4}

DOPAR

2-4.2.

if $(LR_{8,3} = 0)$ then {Horizontal line segment}

if $(LR_{8,4} > 0)$ then

$TR_{8,1} = +\infty$

else

$TR_{8,1} = -\infty$

else

$TR_{8,1} = -(LR_{8,4}/LR_{8,3})$ $\{t_4 = Q_4/\Delta y\}$

DOPAR

3. [Rearrange P_i's so that the signs of P_i's appear in $-+-+$ form]

PARDO

if $(sign(LR_{5,3}) = -)$ then

$TR_{5,1} \longleftrightarrow TR_{6,1}$ {Exchange $TR_{5,1}$ and $TR_{6,1}$}

if $(sign(LR_{7,3}) = -)$ then

$TR_{7,1} \longleftrightarrow TR_{8,1}$ {Exchange $TR_{7,1}$ and $TR_{8,1}$}

DOPAR

4. [Find the values of t_0' and t_1']

4.1 **PARDO**

$TR_{5,2} = TR_{7,1}$
$TR_{6,2} = TR_{8,1}$
$TR_{7,2} = TR_{5,1}$
$TR_{8,2} = TR_{6,1}$
DOPAR

4.2 **PARDO**

$TR_{5,4} = \min (TR_{5,1}, TR_{5,2}, 0)$ {Put t_0' in $TR_{5,4}$}
$TR_{7,4} = \min (TR_{7,1}, TR_{7,2}, 0)$ {Put t_0' in $TR_{7,4}$}
$TR_{6,4} = \max (TR_{6,1}, TR_{6,2}, 1)$ {Put t_1' in $TR_{6,3}$}
$TR_{8,4} = \max (TR_{8,1}, TR_{8,2}, 1)$ {Put t_1' in $TR_{8,4}$}
DOPAR

5. [If $t_0' < t_1'$ then compute endpoints of the visible segment]
if $(TR_{5,4} < TR_{6,4})$ then
PARDO
$LR_{5,4} = LR_{5,2} + LR_{5,3} * TR_{5,4}$ {Compute x_0'}
$LR_{6,4} = LR_{6,2} + LR_{6,3} * TR_{6,4}$ {Compute x_1'}
$LR_{7,4} = LR_{7,2} + LR_{7,3} * TR_{7,4}$ {Compute y_0'}
$LR_{8,4} = LR_{8,2} + LR_{8,3} * TR_{8,4}$ {Compute y_1'}
DOPAR

6. [Shift operation]
PARDO for processor GP5,GP6,GP7,GP8
$LR_{i,2} = LR_{i,1}$
output $LR_{i,4}$
DOPAR

{End of the algorithm}

5. COMPUTATIONAL COMPLEXITY OF THE ALGORITHM

Since the six steps in the algorithm are all parallel steps at the GP level and the instructions involved in each parallel step are essentially the same, it is sufficient to evaluate the computational complexity of this algorithm simply by evaluating its computational complexity on GP5. Implementation of Algorithm PLC on GP5 is shown in Fig. 5.

It is easy to see that scalar operations are required in Step 2 and 5 only. In Step 2, we need one parallel subtraction for Step 2-1.1, one parallel division for Step 2-1.2 (the operations done in Step 2-1.2 are dominated by the division). Therefore, one subtraction and one division are required for Step 2. In Step 5, we need one multiplication and one addition to compute the coordinates of the end points of a visible segment. Therefore, totally four scalar operations are required in our algortihm. Actually, by noticing that Step 1 and Step 6 can be performed in parallel and the fact that a multiplication (division) needs 51 cycles, an addition (subtraction) needs 9 cycles, a data transfer (and I/O as well) needs 1 cycle, it can be shown

Fig. 5. Implementation of Algorithm PLC on GP5.

that our Clipping Subsystem will require only 141 cycles to clip a line segment (1 cycle for

Step 1, 65 cycles for Step 2, 4 cycles for step 3, 7 cycles for Step 4, and 64 cycles for Step 5). Since J. Clark's system will require 160 cycles [6] to clip a line segment, our clipping process is indeed more efficient.

6. CONCLUSIONS

The architecture of a parallel clipping hardware environment and the associated algorithm are presented. We first show that how should a general theory of line clipping be developed so that parallel line clipping can eventually become possible. We then present the architecture of a hardware environment based on which the process of line clipping is to be performed. The associated parallel line clipping algorithm is then given. Using this approach, only four scalar operations: one subtraction, one division, one multiplication, and one addition, are required to clip a line segment against a window or an orthogonal view volume.

Our system has been simulated. The simulation is done by describing the system in ISPS language [1], and then runing a system simulation in Run Time Module in ISPS. By assuming a 4 mega clock rate, our system can process approximately 3800 line segments every 1/30 second.

Since at most two parallel arithmetic operations are required within each step of algorithm **PLC**, only two ALUs are actually needed within each GP. In fact, it should be relatively easy to implement the entire Clipping Subsystem on a single chip to make it more cost-effective and more efficient.

ACKNOWLEDGEMENT

We would like to thank H. C. Fu for helpful discussion during the preparation of this work.

REFERENCES

1. M.R. Barbacci, *The ISPS computer description language*, Technical Report No. CMU-CS-79-137, Department of Computer Science, Carnegie-Mellon University, 1979.
2. F. Cheng and Y.K. Yen, *PIPA: A pipeline-architectured and parallel-hardwired geometry system for graphics*, in preparation.
3 J.H. Clark, The Geometry Engine : A VLSI geometry system for graphics, *Computer Graphics* 16 3 (July 1982), 127-133.
4. J.H. Clark, A VLSI geometry processor for graphics, *IEEEComputer* 12 7 (July 1980), 59-68.
5. J.D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Mass., 1982.
6. H.C. Fu, S.C. Lee and C.H. Bao, *The design of a real time geometry system for graphics*, Technical Report No. NCTU-ERSO Project Report RCGS-84-2, Institute of Computer Engineering, National Chiao Tung Univ., 1984.
7. Y.D. Liang and B.A. Barsky, A new concept and method for line clipping, *ACM Trans. on Graphics* 3 1 (Jan. 1984), 1-22.
8. Y.D. Liang and B.A. Barsky, An analysis and algorithm for polygon clipping, *CACM* 26 11 (Nov. 1983), 868-877.
9. R.F. Sproull and I.E. Sutherland, A clipping divider, Proc. 1968 AFIPS FJCC, Vol. 33, AFIPS Press, Montvale, N.J., 757-764.
10. I.E. Sutherland and G.W. Hodgman, Reentrant polygon clipping, *CACM* 17 1 (Jan 1974), 32-43.

11. K. Weiler and P. Atherton, Hidden surface removal using polygon area sorting, *Computer Graphics*, 11 2 (Summer 1977), 214.