Accelerating Polygon Clipping

Bengt-Olaf Schneider [1]

ABSTRACT

Polygon clipping is a central part of image generation and image visualization systems. In spite of its algorithmic simplicity it consumes a considerable amount of hardware or software resources. Polygon clipping performance is dominated by two processes: intersection calculations and data transfers.

The paper analyzes the prevalent Sutherland-Hodgman algorithm for polygon clipping and identifies cases for which this algorithm performs inefficiently. Such cases are characterized by subsequent vertices in the input polygon that share a common region, e. g. a common halfspace.

The paper will present new techniques that detect such constellations and simplify the input polygon such that the Sutherland-Hodgman algorithm runs more efficiently. Block diagrams and pseudo-code demonstrate that the new techniques are well suited for both hardware and software implementations.

Finally, the paper discusses the results of a prototype implementation of the presented techniques. The analysis compares the performance of the new techniques to the traditional Sutherland-Hodgman algorithm for different test scenes. The new techniques reduce the number data transfers by up to 90 % and the number of intersection calculations by up to 60 %.

CR Categories and Subject Descriptions:

B.4.2 [Input/Output and Data Communications]: Input/Output Devices — Image display. I.3.1 [Computer Graphics]: Hardware Architecture. I.3.3 [Computer Graphics]:Picture/Image Generation — Display algorithms.

Keywords:

Polygon Clipping, Sutherland-Hodgman Algorithm, Simplification, Polygon Preprocessing, Common Subspace, Common Halfspace.

[1]Author's address: IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598
Phone: (914) 784-6002, e-mail: bosch@watson.ibm.com

1.1 Introduction

Clipping is an important step in the process of displaying a computer-generated scene. It removes portions of objects lying outside a viewing area (window) or a viewing volume (frustum). In this paper we will restrict ourselves to polygonal objects and to convex, polyhedral clipping volumes which are typical for today's high-performance graphics workstations.

Several algorithms for line and polygon clipping have been published in the past [2, 3]. However, most of these algorithms suffer from serious drawbacks. Some of them do not handle all cases correctly, others are specifically tailored to two-dimensional clipping areas (windows) and do not easily scale to three (or more) dimensions. The Sutherland-Hodgman algorithm [4] is the only simple and general clipping algorithm, making it the prevalent clipping algorithm in graphics workstations.

Polygon clipping is a central step in the graphics pipeline and despite the simplicity of the algorithm a considerable amount of hardware (or CPU cycles) is dedicated to this step. Therefore, any improvement in the efficiency of the algorithm has a direct impact on the performance of the graphics system. This paper will present several techniques to improve the performance of the clipping process.

The outline of the paper is as follows. In section 1.2 we will analyze the Sutherland-Hodgman algorithm for polygon clipping and identify cases that it handles inefficiently. Section 1.3 presents several ways to overcome these inefficiencies. In section 1.4 we will show that the new method can be easily implemented in hardware. Finally, in section 1.5 we will compare the new algorithms to existing methods.

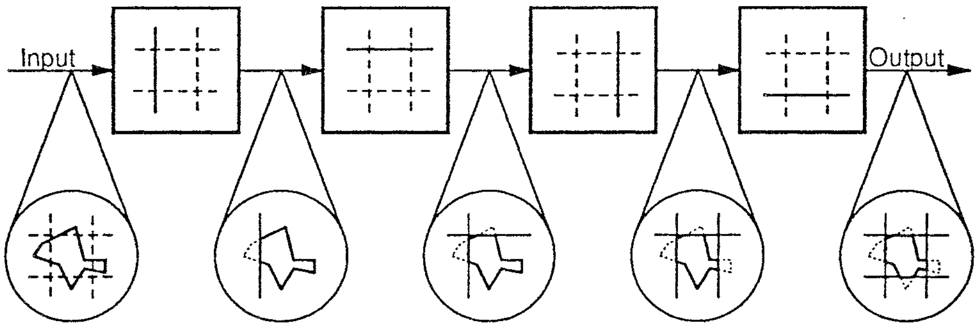


FIGURE 1.1. **Pipeline of Sutherland-Hodgman clipping stages.** The big circles show how each clipping applies its clipping plane to the respective input polygon.

1.2 The Sutherland-Hodgman Algorithm

In this section we will first give a short description of the Sutherland-Hodgman algorithm and highlight certain characteristics of this algorithm. It should be noted that all remarks and observations apply to both 2D and 3D cases and an arbitrary number of clipping planes.

1.2.1 Basic Algorithm

The input to the Sutherland-Hodgman algorithm is a set of clipping planes $c_1 \ldots c_\ell$ and an input polygon P described by its vertices $p_0 \ldots p_m$. The algorithm produces an output polygon $Q = q_1 \ldots q_n$ in the same format. Ideally, the output polygon is the intersection of the input polygon with the viewing volume. In practice however, the Sutherland-Hodgman algorithm can produce degeneracies which we will discuss later.

The algorithm can be best understood when described as a pipeline of clipping stages, each of which clips its input polygon against one of the clipping planes (fig. 1.1). However, the algorithm may also be implemented in software using only one routine that is called recursively, as it was originally described in [4].

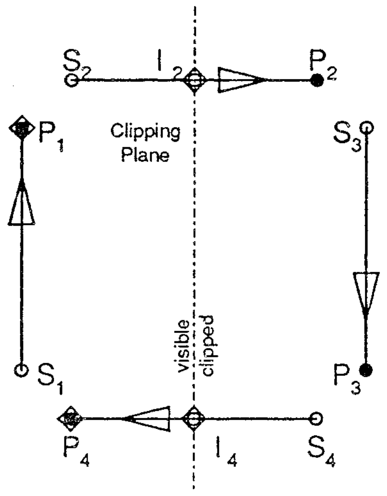


FIGURE 1.2. **Edge processing by the Sutherland-Hodgman algorithm.** The input vertex is shown as a solid disk. Output vertices are enclosed in a diamond marker.

Fig. 1.2 explains how the vertices of the output polygon are generated from the input vertices. The algorithm stores the last vertex as the vertex s (saved), the current vertex is called p. If the edge $s \rightarrow p$ intersects the clipping plane the algorithm computes and outputs the intersection point i. The vertex p is output if it is on the visible side of the clipping plane.

The first vertex of the input polygon is output if it is inside the clipping window and then stored both in s and the variable *start*. After the last vertex of the input polygon has been processed a special step is executed to close the output polygon by connecting the saved vertex s with the first input vertex *start*.

1.2.2 Characteristics of the Sutherland-Hodgman Algorithm

The Sutherland-Hodgman algorithm requires for each stage, i. e. for each clipping plane, local storage for two vertices: the start vertex *start* and the previous (saved) vertex s. Furthermore, each clipping stage maintains a flag *first* indicating whether the next vertex begins a new polygon.

For later use we will note the following properties of the Sutherland-Hodgman algorithm:

P1 Each clipping stage c_i initializes its local variable $start_i$ to the first vertex it receives.

Since each clipping stage forwards only visible vertices to the next stage all $start_1$ $\ldots start_j$ are initialized to p_0, where c_j is the first clipping plane that clips p_0.

P2 Similarly to property P1 we observe that the saved vertex s_i initially assumes the same value as $start_i$. Later, it always contains the last vertex received by clipping stage i. Assuming that a vertex p_k is visible for the first j clipping planes, $s_i = p_k$ for $i = 1 \ldots j$.

P3 It is known that the Sutherland-Hodgman algorithm can produce degenerate polyons [2], i. e. polygons with collinear edges, if the input polygon is concave. These degeneracies occur when the clipping plane splits the input polygon into several disjoint pieces and two or more of these pieces are on the visible side of the plane. Then, the algorithm connects these pieces with collinear edges. For some polygons it depends on the order in which the clipping planes are applied whether such degeneracies occur.

Degenerate output polygons are of no concern if they are processed further by a scan-conversion algorithm. Usually, these algorithm do not produce pixels for these zero-area portions of the polygon, thus preventing them from being displayed. However, if the output polygon is displayed as a wireframe by drawing its edges, these degeneracies have to be removed explicitly in a postprocessing step.

For a given number of input vertices, the execution time of the Sutherland-Hodgman algorithm is determined by several factors:

1. Calculations to determine the intersection of an edge with a clipping plane. These calculations are usually performed as floating point calculations and are therefore expensive.

2. Transferring vertices from one clipping stage to the next. These costs occur either in the form of physical data transfers between hardware components or as (recursive) procedure calls for software implementations.

3. Tests to determine on which side of the clipping plane a vertex lies. Although these tests can be reduced to simple bit tests by properly preprocessing the input vertices, these tests can amount to a considerable number of CPU cycles if the algorithm is implemented using software.

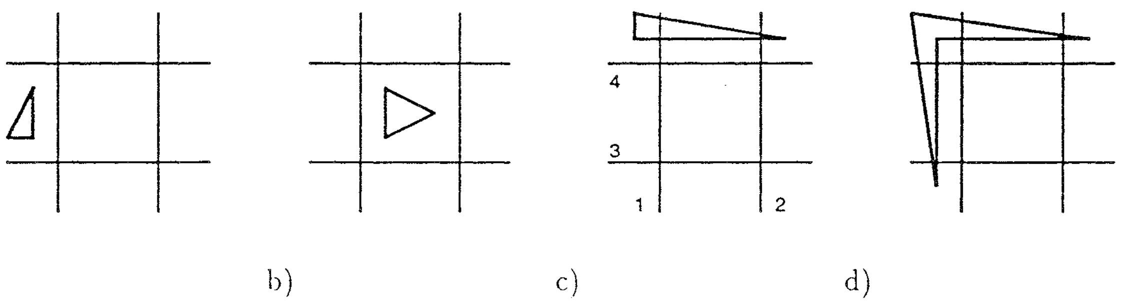


FIGURE 1.3. **Polygons that could be treated trivially.** The Sutherland-Hodgman algorithm processes for these cases inefficiently. In c) the numbers indicate the sequences in which the clipping planes are applied.

The Sutherland-Hodgman algorithm is not optimized for trivial cases, i. e. polygons that can be accepted or rejected, entirely or in part, without performing any expensive computations, tests, or communication. Fig. 1.3 shows some examples that the Sutherland-Hodgman algorithm handles in a suboptimal way. For example, the polygon in fig. 1.3c would be rejected only after up to four intersection calculations although it could be rejected immediately by checking against clipping plane 4.

1.3 Polygon Simplification

In this section we will briefly discuss a previously published technique for improving the efficiency of the Sutherland-Hodgman algorithm. We will then present two new techniques that offer significant performance advantages over this technique.

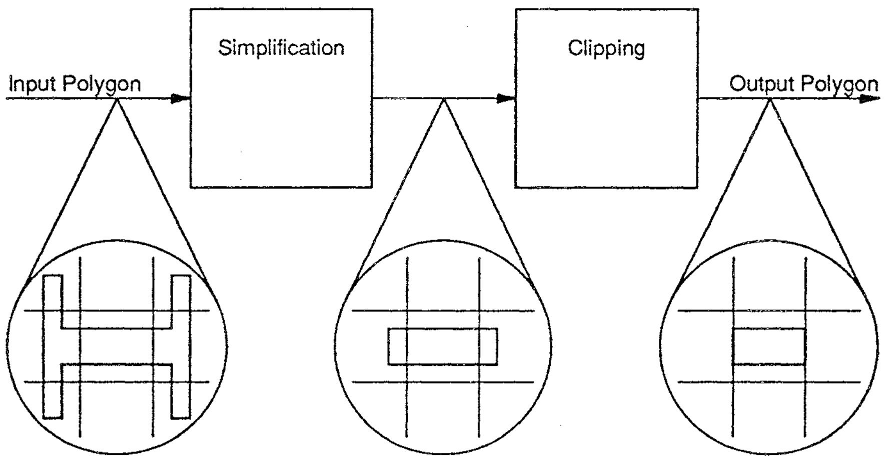


FIGURE 1.4. **Principle of polygon preprocessing for optimizing clipping performance.** The preprocessor simplifies the input polygon by removing all vertices/edges that can be handled trivially.

All techniques split the clipping process into a preprocessing step and the actual clipping step. During preprocessing the input polygon P is inspected for edge (vertex pair) sequences that can be accepted or rejected trivially. Such vertices are taken care of immediately and removed from the input polygon, thus producing a simplified polygon R that only contains non-trivial edges. The simplified polygon is then handed over to the standard Sutherland-Hodgman algorithm (fig. 1.4).

The following techniques differ in how many trivial cases are detected and processed during the preprocessing step. The first two methods only find trivial edge sequences at the beginning of the vertex list describing the input polygon. The third method is able to detect trivial case also in the middle and at the end of the input polygon.

1.3.1 Common Subspace Technique (CS)

This technique was first reported in [1] and is probably used in a number of Silicon Graphics workstations. The method operates in two modes: simplification mode and clipping mode. For each polygon the technique starts in simplification mode and checks for a sequence of vertices $p_0 \ldots p_k$ that are all in the same subspace.[2] Depending on the subspace, vertices are immediately rejected or accepted.

Once the technique encounters a vertex p_{k+1} in a different subspace, it switches to clipping mode and initializes the state variables of the Sutherland-Hodgman algorithm (s and *start*) according to properties P1 and P2:

$$\begin{aligned} start_i &= p_0 \\ s_i &= p_1 \\ \text{for } i &= 1 \ldots j \end{aligned}$$

where clipping plane c_j is the first clipping plane that rejects the vertices $p_0 \ldots p_k$. This initializes the clipping stages to the same state as if the vertices $p_0 \ldots p_k$ had been actually processed by the Sutherland-Hodgman clipping algorithm instead of being handled by the preprocessing stage.

Then, starting with the vertex p_{k+1} all incoming vertices $p_{k+1} \ldots p_m$ are processed using the standard Sutherland-Hodgman clipping algorithm. Fig. 1.5a) shows an example of the simplification performed by the CS method.

The preprocessor for the CS method must maintain a certain amount of state information:[3] It stores the first input vertex p_0 in the variable *START* and the last processed vertex in the variable *S*. Furthermore, it maintains three flags, indicating whether the method operates in simplification mode or in clipping mode (*MODE*), whether a vertex is the first vertex of a new polygon (*FIRST*), and whether vertices are trivially rejected or accepted (*REJECT*). The state variable *EDGE* is initially reset and is set as soon as at least two vertices were found in the common subspace. Then the Sutherland-Hodgman stages must be initialized to the last vertex in this subspace in order to process correctly the edge leaving the subspace. Finally, the algorithm uses the variable *REGION* to remember the common subspace.

We can make the following observations:

- A very small set of state information must be kept for preprocessing the input polygon.
- The efficiency of the method depends on the polygon's starting point. In the worst case the algorithm performs no simplification at all, although the polygon contains a series of vertices in a common subspace.
- The CS method can reduce the number of tests and data transfers (procedure calls), provided it simplifies the input polygon.
- It does not reduce the number of intersection calculations, because the standard Sutherland-Hodgman algorithm is started as soon as the common subspace is left, i. e. the polygon intersects a clipping plane.
- Even with an optimal choice for the starting vertex p_0 this method will only remove *one* sequence of vertices, although the input polygon might contain several sequences of vertices in common subspaces.

[2] A subspace is defined as the set of points in space that share the same in/out classification with respect to all clipping planes. In the 2D examples in this paper there are 9 subspaces: the clipping region and 8 regions around the clipping region. For the standard 3D viewing frustum there are 27 subspaces.

[3] In this paper we will use capitalized names for variables used by the preprocessor, and lower case names for variables used in the Sutherland-Hodgman clipping stages.

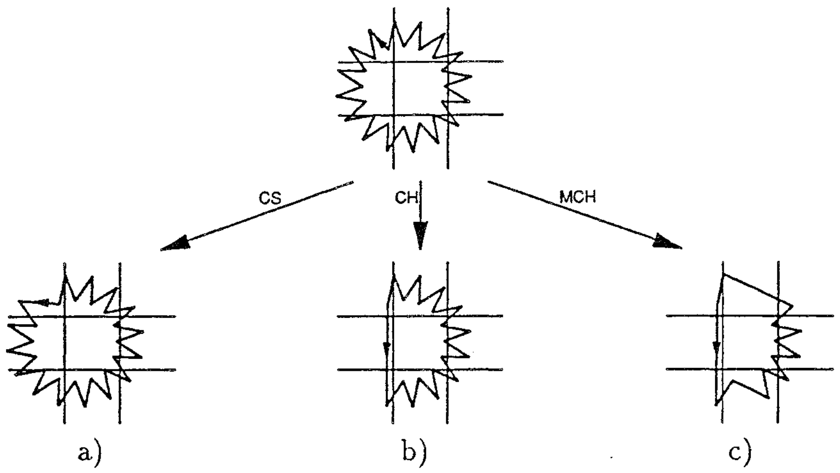


FIGURE 1.5. Comparison of the different simplification techniques. The polygons on the bottom show the results of the simplification step. These polygons are sent to the Sutherland-Hodgman algorithm.

1.3.2 Common Halfspace Technique (CH)

The following technique is similar to the CS technique. However, it has the potential for reducing the number of intersection calculations. It achieves this by checking for a sequence of vertices $p_0 \ldots p_k$ that are either in the viewing volume or in a common halfspace.[4] We will refer to the *common region* when we mean "the viewing volume or a common halfspace". The sequence of vertices $p_0 \ldots p_k$ sharing a common region is simplified to the edge $p_0 \rightarrow p_k$ and then processed further by the Sutherland-Hodgman algorithm. Fig. 1.5b) shows an example.

Special care must be taken when switching from simplification mode to clipping mode. If p_0 and p_k are in different subspaces we must compute the point(s) of intersection between the edge $p_0 \rightarrow p_k$ and the clipping plane(s) separating those subspaces. Therefore, we initialize as follows:

$$\begin{aligned} start_i &= p_0 \\ s_i &= \begin{cases} p_0 & \text{if } p_0 \text{ and } p_k \text{ are in} \\ & \text{different subspaces} \\ p_k & \text{if } p_0 \text{ and } p_k \text{ are} \\ & \text{the same subspace} \end{cases} \\ \text{for } i &= 1 \ldots j \end{aligned}$$

where c_j is the first clipping plane that rejects p_0. In the first case (different subspaces) the Sutherland-Hodgman clipping stages are initialized as if they had only processed vertex p_0 of the simplified polygon $R : p_0, p_k, p_{k+1} \ldots p_m$. Consequently, the vertices $p_k \ldots p_m$ are sent to the Sutherland-Hodgman clipping stages. On the other hand, if p_0 and p_k lie in the same subspace, the Sutherland-Hodgman clipping stage are set up as if they had already processed the vertices p_0 and p_k. Therefore, only the remaining vertices $p_{k+1} \ldots p_m$ need to be processed by the Sutherland-Hodgman algorithm.

The state information maintained by the preprocessing stage is the same as for the CS technique. However, instead of storing the common subspace the common region is stored.

The CH technique exhibits the following characteristic behavior:

[4] A halfspace is formed by all points in space that lie on the rejection side of a clipping plane. Note that points can lie in more than one halfspace.

- A very small set of state information must be kept for preprocessing the input polygon.
- As the CS technique, this technique also depends on the polygons starting point. However, the CH technique considers more points for trivial rejections than the CS technique, thus increasing the probability of simplifying the input polygon.
- Because the CH technique is more likely to simplify the input polygon, there is also an increased chance of reducing the number of clipping steps, and thus the number of tests and data transfers.
- The CH method can reduce the number of intersection calculations. It can remove points in different subspaces from the input polygon, thus eliminating edges that cross clipping planes.
- In some cases the CH method can reduce or eliminate the number of degenerate edges in the output polygon. This happens when the input polygon was simplified to a convex polygon or a polygon with less concave points.[5]
- Even with an optimal choice for the starting vertex p_0 this method will only remove *one* sequence of vertices, although the input polygon might contain several sequences of vertices in common regions.

1.3.3 Multiple Common Halfspaces Technique (MCH)

We will now discuss an extension to the CH technique that can remove several vertex sequences each of which lies in a common region. The preprocessor toggles between reject mode and accept mode. In each mode the input polygon is checked for consecutive vertices that lie within the acceptance region (accept mode) or share a common halfspace (reject mode). Such vertices are accepted or rejected immediately without starting the Sutherland-Hodgman algorithm. If a vertex is found that cannot be handled trivially, it is passed to the Sutherland-Hodgman clipping stages and, if necessary, the algorithm switches modes.

The MCH technique simplifies each of the vertex sequences $p_i \dots p_{i+k}$ in a common region to the edge $p_i \rightarrow p_{i+k}$. This edge is processed by the Sutherland-Hodgman clipping algorithm before the vertex p_{i+k+1} lying in the next halfspace (or in the viewing volume) is sent to the clipping stages.

The flag *EDGE* is set whenever the input polygon contains at least two vertices, i. e. an edge, in a common region. The value of *EDGE* controls whether the last vertex in the common region is sent to the Sutherland-Hodgman algorithm (*EDGE* set) or not (*EDGE* reset).

The variable *CLIPPED* is set if any input vertex was sent to the clipping stages, i. e. whether the common region was left. The final step in processing the input polygon is connecting the last vertex p_m with the first vertex p_0. If both vertices lie in a common region, we have to distinguish two cases: If all other vertices $p_1 \dots p_{m-1}$ were also contained in this region, the last vertex can be trivially accepted or rejected. On the other hand, if the input polygon is contained in more than one region (*CLIPPED* set), the edge $p_m \rightarrow p_0$ must be clipped explicitly, because, based on the available information, we cannot distinguish between polygons enclosing the clipping area and polygons wrapping

[5]At a concave point the adjacent edges enclose an angle greater than 180°.

around the clipping area. Fig. 1.5c) gives an example of a polygon clipped using the MCH technique.

The fact that the MCH technique is not restricted to simplifying only the first vertex sequence in a common region makes it the most powerful of the three techniques discussed in this paper:

- A small set of state information is kept for preprocessing the input polygon.
- The algorithm depends less on the choice of the starting vertex p_0 than the CS and CH techniques. There is still the chance that a bad choice of p_0 will prevent simplification. This happens when p_0 is the middle vertex of three vertices in a common region.
- The MCH technique has the potential for significant savings in clipping steps, i. e. tests and data transfers.
- The MCH technique has the largest potential for reducing the amount of intersection calculations.
- This technique inherits from the CH technique that it can reduce the number of degenerate edges in the output polygon.

1.3.4 Comparison

Table 1.1 summarizes qualitatively the amount of performance improvement achievable with each of the techniques.

Technique	Reduction of			
	Communication	Tests	Intersections	Degeneracies
CS	+	+	−	−
CH	+	+	++	+
MCH	++	++	++	+

TABLE 1.1. **Qualitative comparison of the different preprocessing techniques.** The columns show possible performance improvements for reducing the communication, the number of tests, the number of intersection calculations, and the number degeneracies in the output polygon. The techniques are ranked according to whether they provide no (−), some (+), or significant (++) reductions.

As expected both the CH and the MCH techniques are superior to the CS technique because they have the potential to reduce the number of intersection calculations and the number of degeneracies in the output polygon. Section 1.5 will confirm these assertions with experimental results.

1.4 Hardware Implementation

This section presents the overall architecture for implementing the three techniques presented in the previous section.

Fig 1.6 shows the block diagram of a circuit that implements the techniques CS and CH. The main components are registers for storing vertices and the common region. For the CS technique common regions are formed by subspaces, while for the CH technique common

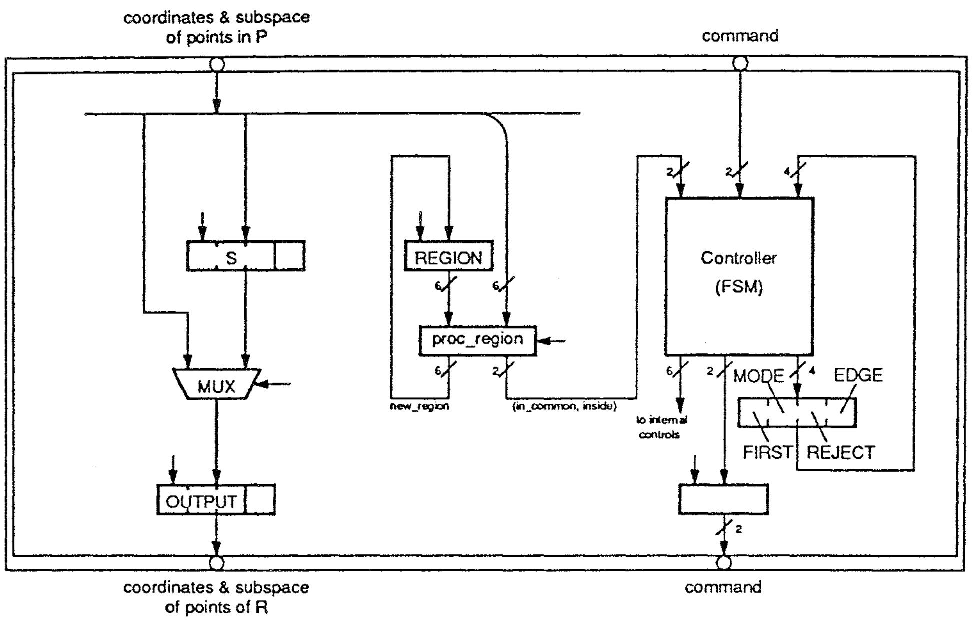


FIGURE 1.6. Block diagram of a preprocessor the CS/CH technique.

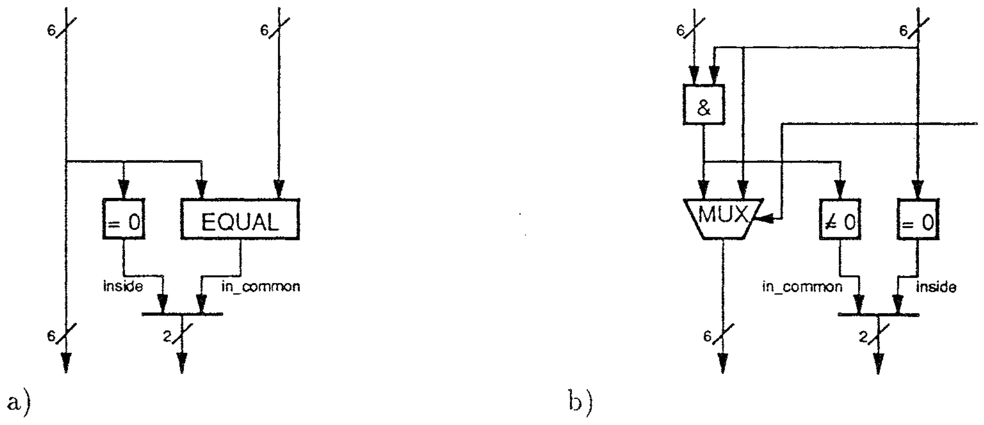


FIGURE 1.7. Details of the block proc_region. (a) for CS technique. (b) for CH/MCH technique. The AND-Gate provides a bit-wise AND function over its inputs.

regions are halfspaces. The block proc_region compares the subspace of the incoming vertex with the stored common region and reports this result to the Controller. The details of this block are different for the techniques CS and CH (see fig. 1.7). Depending on this result and the current state, the controller updates the vertex registers and propagates vertices to the output. The Controller PLA has 7 inputs, 13 outputs, and about 13 product terms.

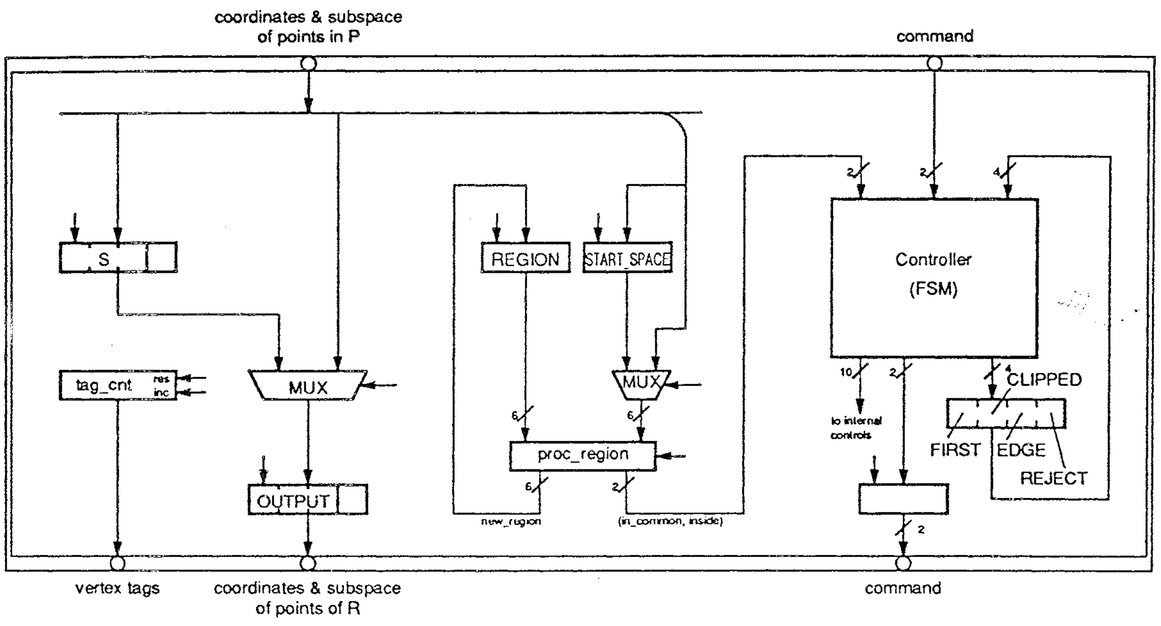


FIGURE 1.8. Block diagram of a preprocessor the MCH technique.

Fig. 1.8 outlines the architecture implementing the MCH technique. Extra state variables (*CLIPPED* and *EDGE*) and a more complex logic for maintaining the common region reflect the increased complexity of this technique. Consequently also the control PLA is larger than for the CH technique.

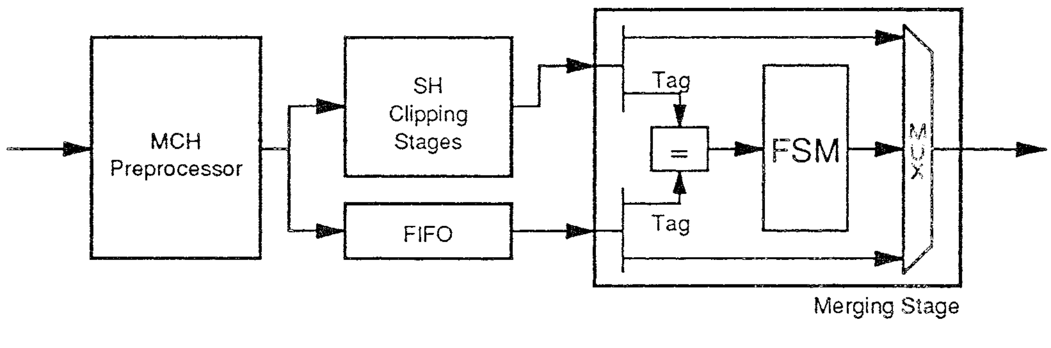


FIGURE 1.9. Merging stage for combining visible vertices from the preprocessing stage and the clipping stages.

The MCH technique is designed to toggle between reject mode and accept mode. Each time the algorithm switches to accept mode it generates a sequence of trivially accepted edges. These edges must be merged with the edges entering and leaving the clipping volume which were generated by the clipping stages. Each of these edges is described by two vertices, e. g. the entering edge is described by an intersection point with the clipping region and the first vertex inside the clipping region. Tags are used to distinguish between different accepted edge sequences. Each time the input polygon enters the clipping region, a new tag is generated by incrementing the tag counter tag_cnt. The tags are attached to all vertices inside the clipping region. Vertices being trivially accepted are stored in a FIFO. Once the entering edge has been processed by the clipping stages a merging stage first propagates the clipped entering edge and then reads all vertices from the FIFO that have the same tag as the vertices of the entering edge. Then, the vertices of the clipped leaving edge are sent (fig. 1.9). The hardware requirements for the merging stage are a multiplexer, a comparator, and a small control PLA which cycles through five principal [6] states:

1. Receive from the last clipping stage the point where the clipping volume is entered.
2. Receive from the last clipping stage the first vertex inside the clipping volume.
3. Read from the FIFO all vertices having the same tag as the vertices received from the clipping stage.
4. Receive from the last clipping stage the last vertex inside the clipping volume.
5. Receive from the last clipping stage the point where the clipping volume is left.

The merging stage can be easily integrated into a postprocessor that cleans the clipped polygon of degenerate edges.

Table 1.2 summarizes the hardware requirements for each of the preprocessing stages. For comparison the approximate (!) requirements for each Sutherland-Hodgman clipping stage are given in the last line of the table.

Technique	Vertex Registers	Space Codes	Flags	PLA (approx.) (in × products × out)	Intersection Unit	Merging Stage
CS	1	1	4	$7 \times 15 \times 12$	NO	NO
CH	1	1	4	$7 \times 15 \times 12$	NO	NO
MCH	1	2	4	$9 \times 17 \times 17$	NO	YES
SH	2	0	1	$6 \times 10 \times 8$	YES	NO

TABLE 1.2. **Hardware requirements for the various preprocessing techniques.**

Considering that the intersection unit executes several multiplications and division steps, it is obvious that each of the preprocessing algorithms can be implemented with less hardware than any of the clipping stages, thus only slightly increasing the hardware complexity. The small number of registers and the size of the PLAs allows to integrate each of the preprocessing circuits entirely on commercially available Field Programmable Gate Arrays (FPGA).

[6]Some extra care must be taken to handle polygons correctly whose first vertex is visible.

1.5 Experiments

1.5.1 Description of the Experiments

We have implemented the Sutherland-Hodgman algorithm and the three preprocessing techniques in software. The implementation minimizes the number of Sutherland-Hodgman clipping steps according to the properties P1 and P2 by initializing the status variables of the clipping stages before performing the first clipping steps. (The pseudo-code in the appendix details these initialization steps.)

We have measured the performance of the different methods by counting the number of (recursive) invocations of the Sutherland-Hodgman clipping procedure and the number of intersection calculations. All tests involved six clipping planes positioned at $xyz = \pm 0.5$ forming a unit cube centered at the origin.

We have conducted two series of tests. The first one tested the performance of the algorithms depending on the area of the input primitives. Where applicable, the number of vertices per primitive was set to 20. The second series of tests measured the performance in relation to the number of vertices per input polygon for a fixed size of the polygons.

We ran each series of tests for different sizes of the world volume as compared to the viewing volume. This gave different ratios of visible vertices to total vertices. For each scene, the objects' centers of gravity where randomly distributed in the world volume.

The first set of scenes contained triangles. They are representative for graphics applications that use triangulated scene descriptions for display. Also, most of today's graphics workstations are built on top of triangle rasterizers.

The next set of scenes was composed of ellipses which belong to the particularly well-behaved class of convex polygons. For such polygons we can expect a strong coherence in the in/out classification of adjacent vertices, which allows for drastic simplification by the discussed techniques.

Finally, we processed scenes of star-shaped polygons (fig. 1.5). We have chosen this primitive type to test the performance of the algorithms for input polygons with many concave vertices. For such polygons there is a high probability that only few adjacent vertices share a common subspace, thus forming a difficult case.

1.5.2 Results

A comparison of the different techniques is shown in fig. 1.10. The plots on the left compare the number of Sutherland-Hodgman clipping steps as a function of the objects size for the methods CS, CH, and MCH. The plots on the right compare the number of intersection calculations as a function of the objects size for the techniques CH and MCH. (Since the technique CS cannot reduce the number of intersection calculations, it is not shown in these plots.)

Fig. 1.11 compares the degree of simplification achieved by the various preprocessing techniques, i. e. the ratio of the number vertices in the input polygon to the number of vertices processed by the Sutherland-Hodgman algorithm.

The results in figs. 1.10 and 1.11 were obtained for a clipping volume that occupied 35 % of the entire world. Although not shown here, we have conducted the same experiments for other ratios between world size and clipping volume. Qualitatively, the results were similar, i. e. the curves had the same shape. The quantitative differences are characteristic for the particular algorithm and are discussed in the following subsections.

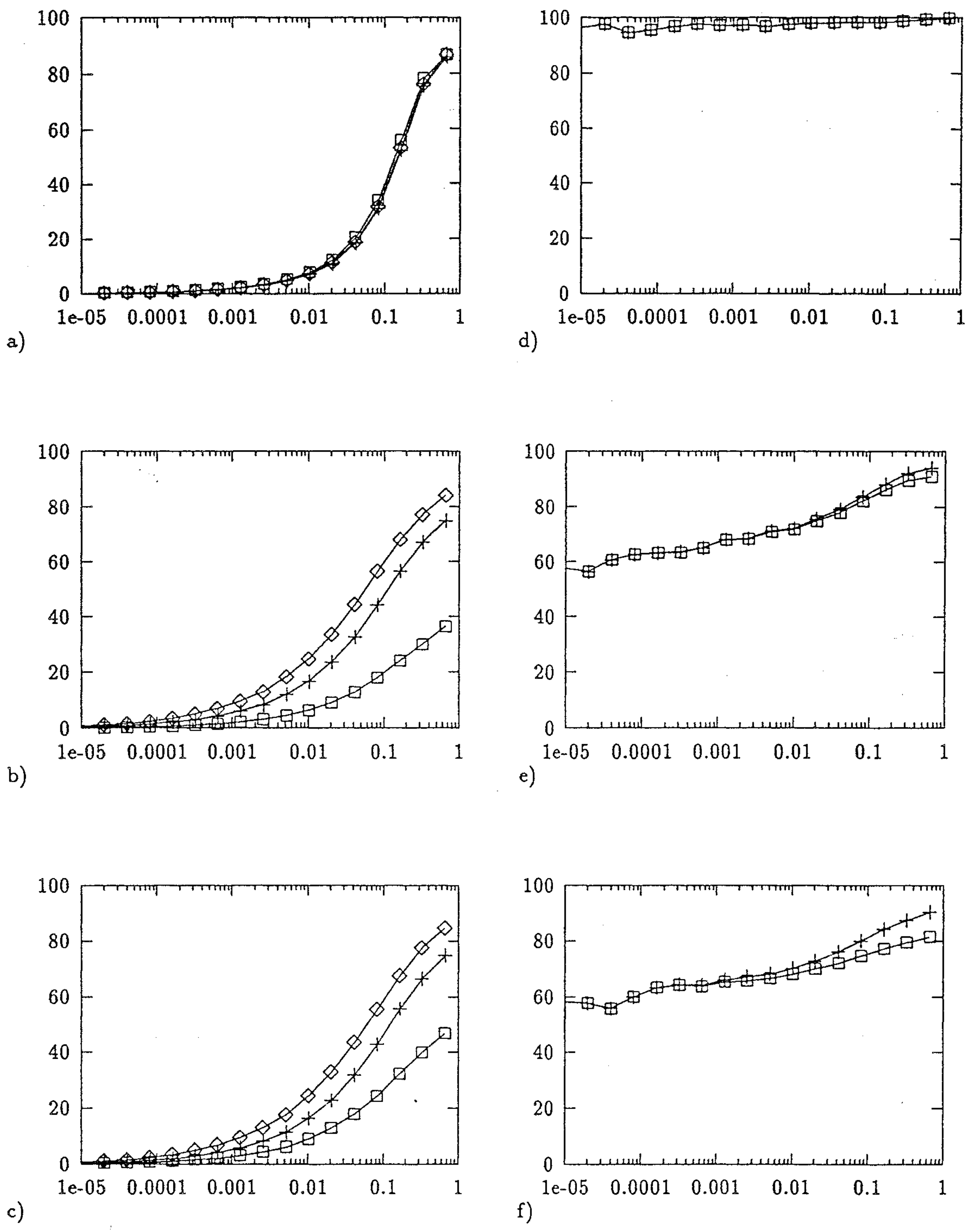


FIGURE 1.10. **Comparison of the techniques CS (◇), CH (+), MCH (□).** The clipping volume occupies 35 % of the universe. The x-axes show the size of the primitives in a unit cube. The y-axis are scaled in percent of steps/calculations performed by the Sutherland-Hodgman algorithm. a-c) Clipping steps for triangles, ellipses, and stars. d-f) Intersection calculations for triangles, ellipses, and stars.

Reducing the Number of Clipping Steps

We observed the expected direct relationship between the degree of simplification, i. e. the ratio between he number of vertices in the simplified polygon and in the input polygon, and reductions in the number of clipping steps. Comparing the plots on the left side of fig. 1.10 with the corresponding curves in fig. 1.11 shows that both sets of curves have exactly the same shape.

In general, all techniques are significantly more efficient for small primitives because the probability that consecutive vertices lie in the same region (either subspace of halfspace) increases for small primitives.

All techniques are performing almost identically for triangles because a triangle cannot be simplified — triangles can only be handled trivially by rejecting or accepting them completely. However, the techniques CH and MCH are slightly more efficient because they detect more cases for rejecting entire triangles. This effect becomes more significant as the percentage of primitives outside the clipping volume increases, i. e. for smaller viewing volumes.

The same effect can be observed for ellipses and stars as well. For clipping volumes only slightly smaller than the world the difference in performance between the CS and CH techniques is only marginal, because their criteria for trivial acceptance are the same. The MCH technique can also accept more edges trivially and is hence more effective even for a high percentage of primitives in the clipping volume. This ability however gets important only for polygons with many edges.

We found that the performance improvements were independent of the number of vertices used to describe the ellipses and star-shaped polygons.

Reducing the Number of Intersection Calculations

As mentioned before only the techniques CH and MCH are able to reduce the number of intersection calculations. We observed a a roughly linear dependency on the primitive size. The achievable reductions were far smaller than for the number of clipping steps, they lay in the range of 0 % to 60 %. For small primitives (< 0.001) there was no significant difference between the two techniques.

With increasing percentage of primitives outside the clipping region, the reductions were also growing. This effect is due to the effect that for a larger world size the probability increases that one clipping plane rejects the whole input polygon (or at least large portions) thus making intersection calculations with other clipping planes obsolete.

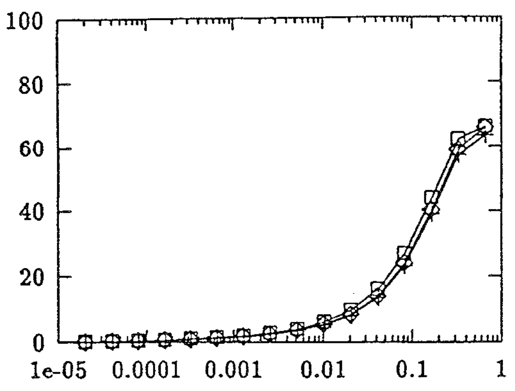
We observed slightly better performance for the star-shaped polygons than for the ellipses because concave polygons are more likely to intersect with the same clipping several times. If all these intersection calculations can be saved this results in larger savings.

For both ellipses and stars, the performance improvements were independent of the number of vertices used to describe these primitives.

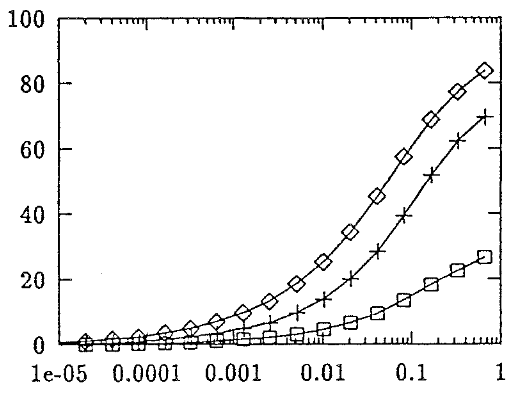
1.5.3 Evaluation

Our experiments show that preprocessing polygons before actually clipping them can result in significant reductions for both the number of clipping steps and the number of intersection calculations during the clipping steps. Recognizing that the CH technique always performs better than the CS technique without requiring more complex implementations, one only has to choose between the CH and the MCH technique.

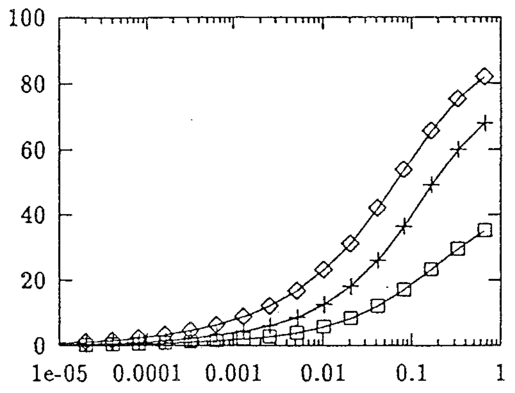
The choice of any particular technique depends on the implementation and on the work load. For a software implementation the MCH technique is the first choice because of the



a)



b)



c)

FIGURE 1.11. **Comparison of CS (◇), CH (+), MCH (□).** The fraction of vertices remaining after the simplification. The clipping volume occupies 35 % of the universe. The x-axes show the size of the primitives in a unit cube and the y-axes the percentage of the input vertices contained in the simplified polygon. a-c) Number of steps for triangles, ellipses and stars.

large savings in the number of clipping steps which directly translate into reductions in procedure calls and data accesses.

For a hardware implementation the differences in implementation cost change the trade-off between the techniques. The expected work load is the primary selection criterion. If the data consists mainly of small triangles, both methods are doing equally well. In this case the CH technique is sufficient. If on the other hand, a significant portion of the input consists of modestly large polygons with more than 4 vertices the performance improvements achieved by the MCH technique outweigh the additional costs incurred by more complex hardware.

1.6 Conclusion

We have presented two new techniques for improving the efficiency of polygon clipping implementations by preprocessing the input polygon. The preprocessor simplifies the polygon by removing edges from the polygon that can be accepted or rejected trivially. Two edges are replaced by only one edge if the original edges share a common region. A common region is either a halfspace outside of the clipping region or the clipping region itself. The first technique applies the simplification only to the beginning of the input polygon, while the second technique tries to simplify as much as possible by toggling between acceptance and rejection mode.

These techniques constitute notable advances over a previous technique that uses subspaces as the common region for simplifying the input polygon. We have shown that the first of the new techniques requires the same amount of hardware or software resources and performs much better than this technique.

We then described hardware and software implementations of the new techniques. The implementation of these techniques has verified that the efficiency of polygon clippers can be improved by up to 90 % in terms of number of clipping steps and by up to 60 % in terms of number of intersection calculations. These techniques provide a means to greatly improve the efficiency of polygon clipping in graphics workstations.

1.7 References

[1] Kurt Akeley and Carl P. Korobkin. Efficient Graphics Process for Clipping Polygons. U.S. Patent No. 5,051,737, Sep. 24, 1991. Silicon Graphics, Inc.

[2] John D. Foley and Andries van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley Publishing Company, 1982.

[3] Ari Rappoport. An efficient algorithm for line and polygon clipping. *The Visual Computer*, (7):19–28, 1991.

[4] I.E. Sutherland and G.W. Hodgman. Reentrant Polygon Clipping. *Communications of the ACM*, 17(1):32–42, January 1974.

Appendix: Pseudo-Code

The following C-like pseudo-code details the implementation of the CH and MCH techniques. As in the rest of the paper variables in capital letters, e. g. *FIRST*, are state variables of the preprocessing stages. State variables of the Sutherland-Hodgman algorithm use lower-case letters, e. g. *first*. The Sutherland-Hodgman clipping algorithm is invoked by *SH_clip(p)*, which takes as its argument a point *p*.

For each point the routines expect a code describing its subspace. This code is identical to the outcodes used in the Cohen-Sutherland line clipping algorithm [2]. One bit is assigned to each clipping plane. A bit is set if the point is on the rejection side of this plane. If the bit-wise logical AND of two outcodes contains at least one asserted bit, the two points belonging to these codes share a common halfspace. The simplification routines *CH_clip* and *MCH_clip* use these codes to describe common regions.

For clarity, the CH pseudo-code does not initialize the Sutherland-Hodgman stages directly as it was described in section 1.3.2. Instead, the Sutherland-Hodgman stages are simply initialized by sending the appropriate vertices. It should be apparent from the description of the algorithm where these initializations should happen; these places have been marked with the comment *Init SH* in the pseudo-code. Actually, this initialization may be preferrable for a hardware implementation because it does not require extra datapaths to access the internal registers of the Sutherland-Hodgman clipping stages.

A.1 CH Technique

```
typedef struct
          { float x, y, z ;
            int subspace ;
          } Point ;

/* State variables used by the CH algorithm */
static int FIRST ;   /* First vertex flag */
static int REJECT ;/* Reject/Accept flag */
static int EDGE ;    /* Skipped a vertex */
static int MODE ;    /* Simplify/Clip flag */

static int REGION ;/* Common halfspace */

static Point S ;        /* Saved point */


/* Initialization routine for the CH algorithm */
void CH_init (void)
{ FIRST = TRUE ;
  MODE = FALSE ;
  SH_init() ;
}


/* Simplification and invocation of SH algorithm */
void CH_clip (Point p)
{ int subspace ;

  subspace = p.subspace ;

  if (FIRST)
  { FIRST = FALSE ;
    EDGE = FALSE ;
    S = p ;
    if (subspace == 0)
      /* trivial accept */
      REJECT = FALSE ;
    else
    { /* trivial reject */
      REJECT = TRUE ;
      REGION = subspace ;
    }
    SH_clip (p) ;
  }
  else /* not FIRST */
  { if (MODE)
      SH_clip(p) ;
    else
      if (REJECT)
      { if (subspace & REGION)
        { S = p ;
          REGION &= subspace ;
          EDGE = TRUE ;
        }
        else
        { MODE = TRUE ;
          if (EDGE) SH_clip (S) ; /* Init SH */
          SH_clip (p) ;
        }
      }
      else /* not REJECT */
      { if (subspace == 0)
        { output(p) ;
          S = p ;
          EDGE = TRUE ;
        }
        else
        { MODE = TRUE ;
          if (EDGE) SH_clip (S) ; /* Init SH */
          SH_clip (p) ;
        }
      }
  } /* not FIRST */
}


void CH_close (void)
{ if (MODE)
    SH_close () ;
  else
    output("close") ;
}
```

A.2 MCH Technique

```
typedef struct
        { float x, y, z ;
          int subspace ;
        } Point ;

/* State variables used by the MCH algorithm */
static int FIRST ;      /* First vertex flag */
static int REJECT ;     /* Reject/Accept flag */
static int EDGE ;       /* Skipped a vertex */
static int CLIPPED ;    /* "Real" SH clipping was done */

static int REGION ;     /* Common halfspace */
static int START_SPACE ;

static Point S ;        /* Saved point */


/* Initialization routine for the MCH algorithm */
void MCH_init (void)
{ FIRST = TRUE ;
  SH_init() ;
}


/* Polygon simplification and invocation of SH algorithm */
void MCH_clip (Point p)
{
  int subspace ;

  subspace = p.subspace ;

  if (FIRST)
  { FIRST = FALSE ;
    EDGE = FALSE ;
    CLIPPED = FALSE ;
    START_SPACE = subspace ;
    S = p ;

    if (subspace == 0)
      /* start in trivial accept mode */
      REJECT = FALSE ;
    else
    { /* start in trivial reject mode */
      REJECT = TRUE ;
      REGION = subspace ;
    }
    SH_clip (p) ;
  }
  else /* not FIRST */
  { if (REJECT) /* previous out*/
    { if (subspace & REGION)
      { /* reject trivially */
        S = p ;
        REGION &= subspace ;
        EDGE = TRUE ;
      }
      else
      { /* clip and determine new mode */
        CLIPPED = TRUE ;
        if (EDGE) SH_clip (S) ;
        SH_clip (p) ;

        S = p ;
        REJECT = (subspace != 0) ;
        REGION = subspace ;
        EDGE = FALSE ;
      }
    }
    else /* previous in */
    { if (subspace == 0)
      { /* accept trivially */

        /* When inside the visible window,
         * we simplify the polygon sent to the
         * SH algorithm to a line connecting the
         * first and the last visible vertex.
         * Only these two vertices are
         * processed by the SH algorithm,
         * the rest is handled trivially.
         * Hence, these two vertices are included
         * in the output poly by the SH algorithm
         * and must not be output by "us".
         * We achieve this as follows:
         * 1. We output an vertex only iff
         *    its successor is also visible
         *    That is accomplished by sending the saved
         *    vertex 'S' to the output, if 'p' is visible.
         * 2. We do not output 'S' if it is the
         *    first vertex inside the window
         *    i.e. if it is equal to 'ENTERED'.
         *    However, to avoid the comparison,
         *    the variable 'EDGE' controls whether
         *    'S' is output.It is reset for the
         *    first visible vertex and set afterwards.
         *    This has the effect of skipping 'ENTERED'.
         */
        if (EDGE) output (S) ;
        S = p ;
        EDGE = TRUE ;
      }
      else
      { /* clip and goto trivial reject mode */
        CLIPPED = TRUE ;
        if (EDGE) SH_clip (S) ;
        SH_clip (p) ;
        S = p ;
        REJECT = TRUE ;
        REGION = subspace ;
        EDGE = FALSE ;
      }
    } /* previous in */
  } /* not FIRST */
}


void MCH_close (void)
{
  if (REJECT)
  { if (! CLIPPED)
      /* do nothing */
      output ("close") ;
    else
    { if (EDGE) SH_clip (S) ;
      SH_close () ;
    }
  }
  else
  { if (START_SPACE == 0)
    { if (EDGE) output (S) ;
      output ("close") ;
    }
    else
    { if (EDGE) SH_clip (S) ;
      SH_close () ;
    }
  }
}
```