Triangle Similarity based Line Clipping

Saurabh Kumar, Aditya Agnihotri , Bhuvnesh Dadhich

***Abstract*—A new algorithm for clipping a line segment in two dimension against a rectangular window is presented in this paper. This algorithm uses an efficient method using triangle similarity for finding intersection point of the clip line against the clipping windows. This greatly reduces the amount of calculations needed to find the final clipped line.**

***Keywords*—computer graphics, line clipping, triangle similarity, algorithm**

I. Introduction

Line clipping is a very important task in computer graphics that is performed regularly in order to compute the final output that must be projected on the device window. Generally, the window is a rectangle or a general polygon.

One of the most popular and classical algorithm for line clipping is Cohen-Sutherland algorithm. This algorithm divides the the clipping window into regions and then tries to cut the lines till they become trivially solvable. The major advantage of this algorithm is that it is easy to implement, but it is a sequential algorithm and does not allow for much parallelism in its operations. This prevents it from taking advantage of multi threading that is available in most modern processors.

This problem was greatly alleviated by the Liang-Barsky algorithm that treats the line as a parametric equation and tries to find all the possible intersections of the line with the clipping window. There can be up to 4 possible intersections and the algorithm then filters out the invalid points to get the points of intersection. Linag-Barsky allows greater degree of parallelism, making it much more efficient than Cohen-Sutherland.

In this paper we will improve upon the classical Cohen-Sutherland algorithm by extending it to point level parallelism. We will also introduce an efficient method for finding the point of intersection between the clip line and the clipping window using triangle similarity.

II. Triangle Similarity

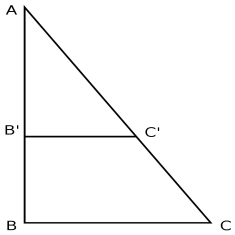

Fig. 1. ABC and AB'C' are two similar triangles

Given 2 triangles ABC and AB'C' (figure 1), we can say that the 2 triangles are similar as we can achieve either by performing simple uniform scaling transformation on the other one. Similar triangles hold an interesting, that they maintain the ratio between the corresponding sides of the triangles.

$$\frac{AB}{BC} = \frac{AB'}{B'C'}$$

We will exploit this property in our algorithm to quickly find the point of intersection.

III. Algorithm Details

Before we solve for intersections we compute the region in which the point lies. Like Cohen-Sutherland the point can lie in 5 possible regions.

1) Top
2) Bottom
3) Left
4) Right
5) Inside

Calculation of the region is needed in order to know where the perpendicular of the triangle should be dropped.

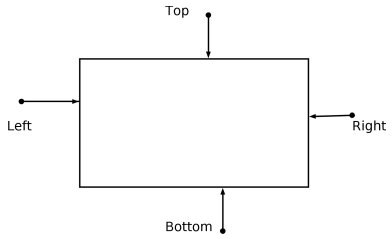

Fig. 2. Projections of the perpendicular from the different regions

When a point is inside the window it is considered as the intersection point itself and no further calculation is done on it. We will also ignore any line which does not intersect with the clipping window, i.e the logical and of the both end points is not equal to 0.

Next we need to calculate the values of dx and dy which is the absolute difference between the coordinates of the clip line. Suppose the two points are (x_0, y_0) and (x_1, y_1), then the values of dx and dy will be.

$$dx = |x_0 - x_1|$$
$$dy = |y_0 - y_1|$$

This is where our algorithm deviates from Cohen-Sutherland. While Cohen-Sutherland will now proceed to sequentially calculate the points of intersection between the end-points and the clipping window, we will use the pre-computed values of the region, dx and dy to allow each point to perform calculations independent of the other endpoint. This allows for great boost in performance as we now have point level parallelism.

A. Finding the point of intersection

Cohen-Sutherland uses the slope intercept form of line in order to calculate the point of intersection. This requires a multiple floating point division in order to achieve decent amount accuracy. This is can cause problems in low performance machines which do not have high floating point registers.

Our methods requires a only one integer division in order to calculate the point of intersection. This is done by exploiting the property of similar triangles.

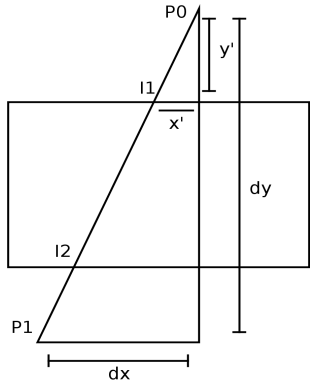

Fig. 3. Similar triangles for finding intersection

For each point we know the height and base of the larger triangle which will always either be dx and dy (figure 3). We also know the value of y' which can be computer as shown below. This is the perpendicular factor.

$$y' = |y_0 - y_{min}|$$

We can now compute the value of x' using all the values we have pre-computed by applying the below formula, we call this value the shift factor. The shift factor will be different for each side.

$$x' = \frac{y' \times dx}{dy}$$

Hence the point of intersection $I1$ is:

$$y_{I1} = y'$$

$$x_{I1} = x_0 - x'$$

Similar calculations can be performed on points lying any of the regions of the clipping window. The above division can be an integer division without loss of accuracy.

Once the point of intersection has been calculated it needs to checked if it lies within the clipping window, if it does not then it is trivially ignored.

IV. Algorithm

The algorithm can be performed in the following steps.

1) Calculate the region of the point.
2) Calculate the values of dx and dy.
3) Ignore trivially solvable points, i.e points that are inside the region and points that dont intersect with the region.
4) Parallel compute the following steps for both the points.
5) find the length of the perpendicular from the point to the respective side of the regions.
   a) Depending on the region we may need to calculate up to two such perpendiculars.
   b) If the point is completely in one region then we only drop one perpendicular.
   c) If the point lies in more than one region (corner) then we need to drop two perpendiculars.
6) Calculate the shift factor using all the pre-computed values.
7) Perform the shifting to get the point of intersection.
8) Check if the point is inside the clipping window, if not then ignore it.

V. Generalization

In this section we will discus how the algorithm generalizes to the 4 possible conditions of line clipping.

A. No intersection, Both points are inside

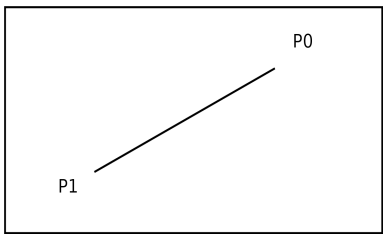

Fig. 4. No intersection, Both points inside

In this case both the points will be ignored and no calculation will be performed by the algorithm.

B. No intersection, Both points are outside

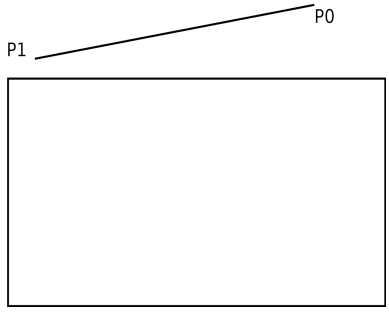

Fig. 5. No intersection, Both points outside

In this case both points will be ignored as the line does not intersect with the clipping windows and no calculations will be performed.

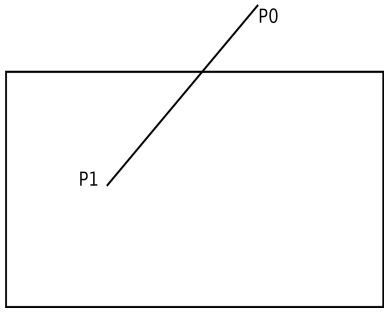

Fig. 6. One point of intersection

C. One point of intersection

In this case the algorithm will ignore point P_1 since it is inside the region and only perform calculation on point P_0. point P_0 can have up to two possible perpendiculars to the clipping region. Out of the two possible intersections one will be trivially ignored as it wont lie with in the clipping region.

D. Two point of intersection

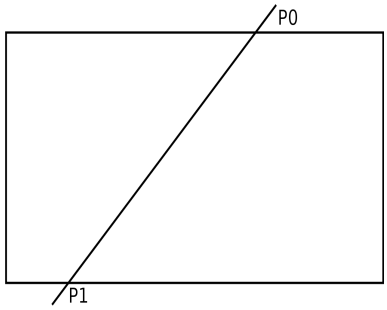

Fig. 7. Two point of intersection

Both points will be taken into consideration, up to 4 calculations need to be made on both the points.

VI. Analysis

The algorithm is closes to complexity to Liang-Barsky than it is to Cohen-Sutherland so we will compare our algorithm with Liang-Barsky.

In Liang-Barsky algorithm we always need to perform 4 operations on each of the extremes of the clipping polygon. This is constant regardless of the orientation of the points relative to the clipping window. Out of these 4 points 2 are always ignored.

In our algorithm since we perform point level operations, we use the information of the region which allows us to filter out unnecessary operations in most cases. When the point is completely in one region then we know exactly where to drop the perpendicular and can compute the intersection with just one operation. So in best case we only perform two calculations as compared to the constant 4 of Liang-Barsky.

Also our algorithm is compatible with integer division, allowing it work on restricted hardware. More over it is allows for point level parallelism making it highly efficient on modern hardware.

VII. Conclusion

The algorithm is an improvement over the classical Cohen-Sutherland algorithm which boosts its performance by allowing for greater parallelism using a much simpler, triangle similarity based technique for finding the intersection of the clip line and clipping window.

The algorithm works at the same theoretical complexity of Liang-Barsky but improves upon it in many situations by performing fewer calculations than Liang-Barsky.

Acknowledgment

We would like to thank Prof. Rajesh Kanna Baskaran for giving us the opportunity to work on this problem and giving us guidance during every phase of this project.

References

[1] Cohen-Sutherland, Cohen-Sutherland Line Clip-ping algorithm ,Computer Graphics,C version ,2ndEd.1997.11,Prentice Hall,Inc.Tsinghua university press,p.226.

[2] You-dong Liang,Brian A.Barsky, Liang-Barsky Line Clipping algorithm,Computer Graphics,C version ,2nd Ed.1997.11,Prentice Hall,Inc.Tsinghua university press .p230.