AN EFFICIENT LINE CLIPPING ALGORITHM FOR CIRCULAR WINDOWS USING VECTOR CALCULUS AND PARALLELIZATION

Prastut Kumar[1] , Fenil Patel[1] and Rajesh Kanna[2]

[1&2] Department of Computing Science & Engineering, Vellore Institute of Technology, Chennai Campus

ABSTRACT

With the advent of digitization and growing abundance of graphic and image processing tools, use cases for clipping using circular windows have grown considerably. This paper presents an efficient clipping algorithm for line segments using geometrical features of circle and vector calculus. Building upon the research with rectangular windows, this method is proposed with the belief that computations are more expensive (heavy) than other computations. Execution time can be drastically improved if we replace expensive computations with cheaper computations. The cheaper computations can be computed even more efficiently using parallelization thus improving time complexity.

KEYWORDS

Line clipping, Circle boundary, vector calculus, parallelization

1. INTRODUCTION

In computer graphics, line clipping algorithms with respect to rectangular windows has seen its fair share of success and has become a basic operation in most graphic applications. Popular algorithms like Cohen-Sutherland [1], Liang-Barsky[2], midpoint division [3] algorithm suited for implementation purposes, Nicholl–Lee–Nicholl algorithm [4], FLC algorithm [5] have been used extensively in production. Most algorithms come into two types, namely the encoding approach and parametric approach.

Today in most graphic and image processing applications, the clipping window is no longer restricted to a simple regular rectangle. This paper describes an algorithm based on circular windows. In recent times, many line clipping algorithms for circular windows have been presented. The two papers [6,7] are substituting the line segment parametric equation into the algebraic equation of the circular window which is a very expensive computation in itself and an additional drawback of ignoring the case that the line does not intersect the circle. Cai et al. [8] uses position relation between the circular window and its outer tangent square, ruling out out the above edge case, and then clips the line segment using the standardized intersection tables. The lookup in searching tables is again expensive. The idea of Lu et al. [9] is applying multiple encoding techniques.

DOI : 10.5121/ijcga.2018.8201

Due to the advent of cheaply available GPU's, parallel processing has been well accepted by the scientific society, with many algorithms ported to support parallelization methods. This paper proposes a clipping algorithm for line segments against circular windows using geometrical features of circle and vector calculus. The method combines both encoding and parametric approaches as we use the parametric equation of line segments for vector calculus and the tangential property of the circle to clip the line. Since the algorithm works in the same fashion for both endpoints of a line segment, we parallelize this process to reduce the execution time.

2. PROPOSED METHODOLOGY

We illustrate our methodology by first discussing the primitives of circle and how we use them inside our algorithm, followed by the explanation of how our algorithm works using the very same primitives.

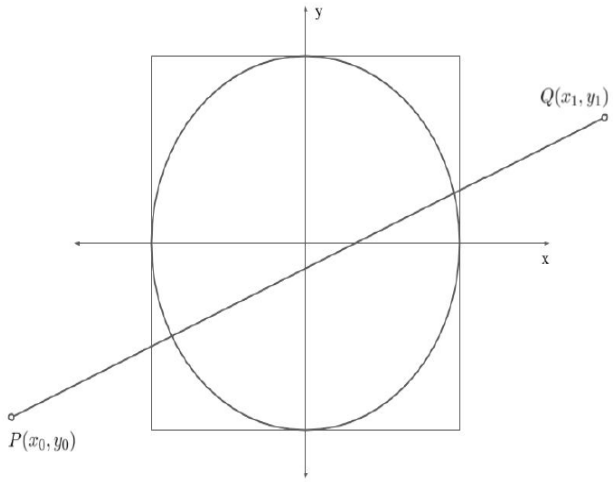


Figure 1. The clipping of line segments against circular window.

Assume that the center of the circular window is at the origin of the coordinate system, the equation of the circle is:

$$x^2 + y^2 = R^2 . \quad (1)$$

and the parametric equation of the line segment PQ is:

$$x = x_0 + t\Delta x,\ y = y_0 + t\Delta y \quad (2)$$

where the endpoints are given by $P(x0 , y0)$ and $Q(x1 , y1)$. [see Fig. 1]

2.1 Preprocessing using Liang-Barsky Algorithm

The tangential square acts as the the clipping window for Liang-Barsky Algorithm. The choice for Liang-Barsky was made since it's one of the most efficient line clipping algorithms in terms of implementation as well. A point on the line segment is in the clip window if

$$x_{min} \leq x_0 + t\Delta x \leq x_{max}$$
$$y_{min} \leq y_0 + t\Delta y \leq y_{max} \quad (3)$$

which can be expressed in terms of 4 inequalities

$$tp_i \leq q_i \ , \ i = 1, 2, 3, 4 \quad (4)$$

Where

$$\begin{aligned} p_1 &= -\Delta x \ , \ q_1 = x_0 - x_{min} \\ p_2 &= \Delta x \ , \ q_2 = x_{max} - x_0 \\ p_3 &= -\Delta y \ , \ q_3 = y_0 - y_{min} \\ p_4 &= \Delta y \ , \ q_4 = y_{max} - y_0 \end{aligned} \quad (5)$$

To bound the tangential circle to the circle, we take the diameter parallel to the x axis and use the left endpoint as x_{min} and the right endpoint as x_{max} and similarly use the diameter parallel to y axis to compute y_{min} and y_{max} . To clip the the line segment, we need to consider various cases:

1. A line parallel to a clipping window edge has $pi = 0$ for that boundary.

2. If for that i, $qi < 0$, then the line is completely outside and can be eliminated.

3. When $pi < 0$, the line proceeds outside to inside the clip window, and $pi > 0$, the line proceeds inside to outside.

4. For nonzero gives the pk , $u = qi / pi$ intersection point.

5. For each line, calculate $u1$ and $u2$. For $u1$, look at the boundaries for which $pi < 0$. For $u2$, look at the boundaries for which $pi > 0$.

6. Take $u1 = max \{0, qi / pi\}$ and $u2 = min \{1 , qi / pi\}$. If $u1 > u2$, the line is outside and therefore rejected.

After applying the above algorithm by putting $x_0 = P(x_0)$ & $y_0 = P(y_0)$, we will be able to compute the new endpoint $P'(x_0 , y_0)$. The same process is repeated with Q'. Finally, the new endpoints given after applying Lian-Barsky Algorithm are P'& Q' . [see Fig. 2]

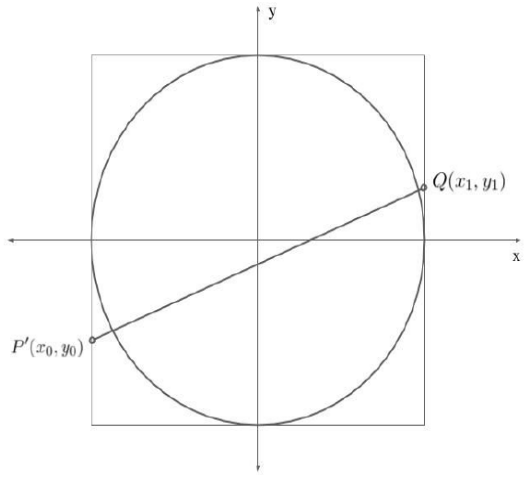
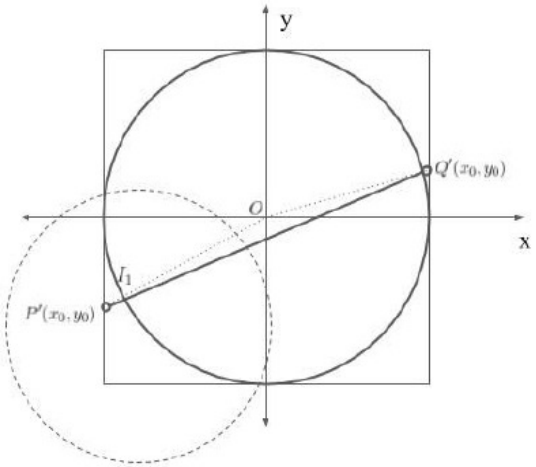


Fig. 2: Using the tangential square as clipping window, applying Lian-Barsky results in new line segment $P'Q'$.

2.2. Line Clipping using vector calculus

As mentioned before, the center of the circle is at origin. Let it be denoted by O . We connect the endpoints P'& Q' to the center resulting in line segments OP'& OQ' [See Fig. 3] . The line segment OP'& OQ' intersects the circle at points I_1 & I_2 respectively. To find intersection point I_1 we don't solve line equation with circle's because it's a very expensive computation since circle's equation in itself is a second degree equation. Instead we traverse a distance equal to the radius of the circle on OP'& OQ' to find I_1 & I_2 . We use vector calculus for traversing the distance and finding the intersection points.



a)

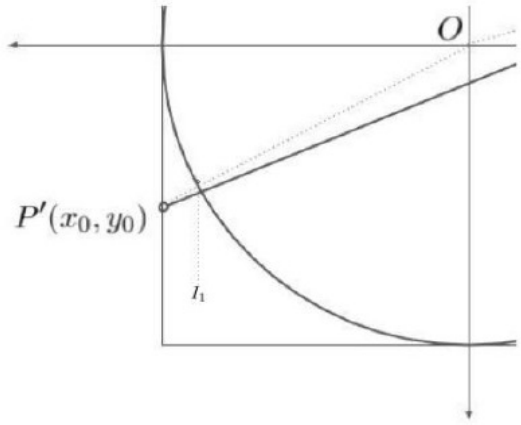


Fig. 3: a) Finding intersection point on circle of line segments OP'& OQ'. b) Zoom over endpoint P'. For brevity purposes, let us first concentrate on & . To find P' I_1 I_1 , we traverse a distance equal to radius r along line segment OP' . Let:

$$v = P'(x, y) - O(x, y) \tag{6}$$

$$u = v \, ||v|| \tag{7}$$

be the normalized vector parallel to OP' and pointing to P' . Therefore:

$$I1 = O(x, y) + r * u \tag{8}$$

On I_1 draw a tangent to the circle which would intersect our line segment $P'Q'$. The intersection point thus created on line segment $P'Q'$ is P'' . Thus this is the new endpoint and the new clipped line becomes $P''Q'$. [See Fig.4]

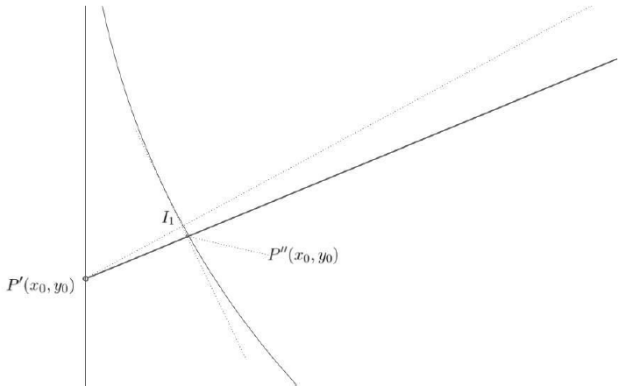


Fig. 4: Tangent drawn from I_1 intersecting OP' at P''.

In similar fashion, we find I_2 and then find Q'' on line segment OQ'. The new clipped line becomes $P''Q''$. This happens repeatedly until the endpoints we get lie on the circle. Thus the original line PQ gets clipped to become $P'Q'$.

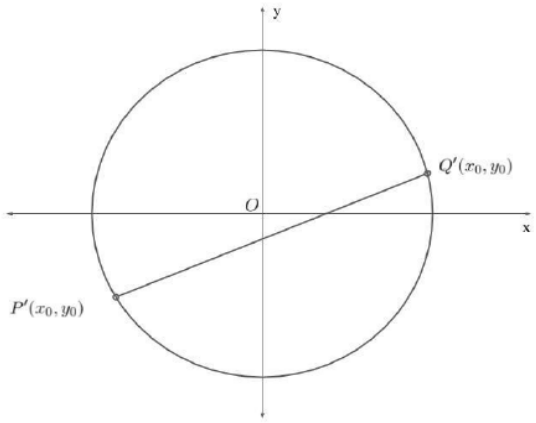


Fig. 5: End of algorithm

2.3. Parallelization

In our proposed algorithm, the new endpoints are calculated in isolation i.e they have no correlation in calculation with each other. Therefore we can parallelize endpoint calculation using multithreading. Only two threads would be required for computation which will be fed by starting points P & Q. Each thread execution produces a new endpoint which is again fed and the process is repeated until the endpoints lie on the circle.

2.4 Handling edge cases

(1) Both endpoints lie inside of the circle . This results in case where line clipping is not required at all. In the beginning of our method, substituting both endpoints in the equation of the circle, we will check if both the endpoints lie in the circle or not:

$$X2 + y2 - R2. \tag{9}$$

If the result of the above operation for both endpoints is ≤ 0 that means the line segment lies completely inside the circle and the algorithm exits. If not then we check for rest of the edge cases.

(2) One of the endpoint lies inside the circle . If either of the endpoint returns a result ≥ 0, we apply our algorithm for that endpoint only.

(3) Both endpoints lie outside of the circle . If both endpoints return a result ≥ 0 then both the end points lie outside of the circle. This results in 2 edge cases i.e if the line lies completely outside the tangential square or the line lies outside of the circle but lies inside the tangential square [see Fig. 6].

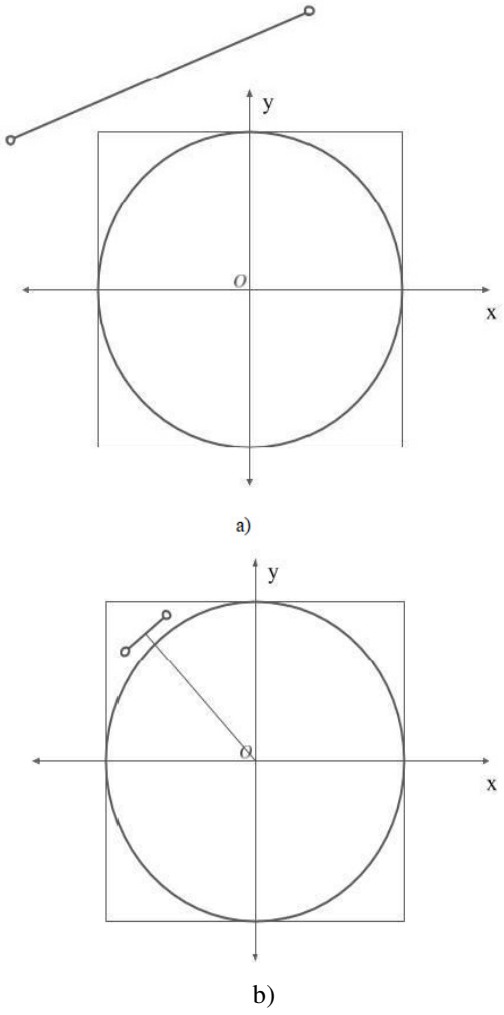


Fig. 6: Showing both cases.

For the first edge case, the first step of our algorithm uses Liang Barsky where the tangential square is the clipping window. Liang-Barsky will return false if the line lies completely outside the tangential square.

For the second edge case, we draw a perpendicular from the center of the circle to the line. If the distance between the point of intersection of line and perpendicular is $\geq r$ that means either the line lies outside the circle or the line is a tangent. In both cases, we exit without clipping. Finally, if the distance is $< r$ the line lies inside the circle and we can begin clipping.

3. CONCLUSION AND DISCUSSION

The algorithm proposed produces the exact same results as with any other line clipping algorithms against circular windows. This paper provides an efficient method to clip line segments against circular windows using modern day parallel processors. The proposed method is extremely fast for the usually occurring case of line segment which lies completely in the circle

having both endpoints outside the circle, taking only constant time for execution. Combined with parallelization, the execution time becomes half of original.

In our literature survey, most proposed algorithms date back to the time when GPUs and parallel processing concepts were starting to develop and focus was on optimizing algorithms to use minimum resources. Now since these tools and concepts have become cost effective, our take is to use them further increase computation speed of our algorithm.

Last, we believe this methodology, albeit a few changes, could be easily applied to clipping of segments like parabola, hyperbola etc.

REFERENCES

[1] Newman WM, Sproull RF. Principles of interactive computer graphics. New York: McGraw-Hill; 1979.

[2] Liang YD, Barsky BA. A new concept and method for line clipping. ACM Transactions on Graphics 1984;3(1):1–22.

[3] Sproull RF, Sutherland IE. A clipping divider. In: Proceedings of Fall Joint Computer Conference. Washington: Thompson Books; 1968. p. 765–75.

[4] Nicholl TM, Lee DT, Nicholl RA. An efficient new algorithm for 2-D line clipping: Its development and analysis. Computer Graphics 1987;21(4):253–92.

[5] Wang J, Liang Y, Peng Q. A 2-D line clipping algorithm with the least arithmetic operations. Chinese Journal of Computers 1991;14(7):495–504 (in Chinese).

[6] Yao H, Song P, Zhang G. Clipping algorithm and practice of circular window. Journal of Computer-Aided Design & Computer Graphics 1992;4(3):14–20 (in Chinese).

[7] Liu Y. Circular and elliptic clipping windows. Computer Engineering and Design 1994;15(4):33–7 (in Chinese).

[8] Cai M, Yuan C, Song J, Cai S. A fast line clipping algorithm against circular windows. Journal of Computer-Aided Design & Computer Graphics 2001;13(12):1063–7 (in Chinese).

[9] Lu G, Xing J, Tan J. New clipping algorithm for line against circular window with multi-encoding approach. Journal of Computer-Aided Design & Computer Graphics 2002;12:1133–7 (in Chinese).