International Journal of CAD/CAM Vol. 2, No. 1, pp. 23~28 (2002)

INTERNATIONAL JOURNAL OF CAD/CAM
www.ijcc.org

A Sweep-Line Algorithm and Its Application to Spiral Pocketing

Tawfik T. EL-Midany[1], Ahmed Elkeran[1] and Hamdy Tawfik[2]*
[1]*Prod. Eng. & Mechanical Design Dept. Mansoura University, Mansoura, Egypt*
[2]*Technology Development Dept. Workers' University, Mansoura, Egypt*

Abstract – This paper presents an efficient line-offset algorithm for general polygonal shapes with islands. A developed sweep-line algorithm (SL) is introduced to find all self-intersection points accurately and quickly. The previous work is limited to handle polygons that having no line-segments in parallel to sweep-line directions. The proposed algorithm has been implemented in Visual C++ and applied to offset point sequence curves, which contain several islands.

Keywords*: *Monotone chain, sweep-line, self-intersection, spiral pocketing, line-offset

1. Introduction

In order to machine complex pockets on milling machines, it is necessary to fill 2D areas with a back and forth sweeping motions of the cutting tool. There are two sweeping motions, spiral offset and zigzagging paths. The spiral offset is defined as a locus of the points, which are at constant distance *d* along the normal from the generator curve. Spiral offsets are widely used in various applications, such as tool path generation for 2.5-D pocket machining [3, 9, 14, 15, 20], 3D NC machining, and access space representations in robotics. Spiral milling is an important operation in CAD/CAM, and the problem has been widely studied, mostly, as a pocket-machining problem through three approaches. Line-offset (pair-wise) [8, 13, 17], Voronoi diagram [10], and pixel-based approach [4]. Voronoi diagram needs a very careful implementation to avoid numerical computational error [10]. Pixel-based approach would require a large amount of memory and an excessive computation time to achieve an adequate level of precision [5]. Line-offset approach is more stable, not prone to computational errors, and would not require a large amount of memory [5]. Self-intersection is one of the main problems in line-offset so, it is an essential task for practical applications to detect all polygons of the self-intersection points correctly and generate valid polygons. The literature survey on offset curve and self-intersection polygons prior to 1992 was conducted by Pham [18] and after 1992 by Takashi [22]. The self-intersection polygons can be handled through two approaches, line-segments intersections [7, 18] and sweep-line [1, 11, 12]. Sweep-line is more efficient than line-segments intersections [21]. Bentley and Ottmann 1979 [1] introduced a sweep-line algorithm to find all k intersections among n line-segments with an $O((n+k).\log n)$ time complexity. Chazelle *et al.* [2] and Mehlhorn *et al.* [16] developed Bentley *et al.* algorithm [1], but their algorithm is more complicated to implement [17]. Park *et al.* 98, developed a sweep-line algorithm to find all intersections k among polygonal chain which has m monotone and n line-segments with an $O((n+k).\log m)$ time complexity, but it is only restricted for polygons which contain line segments nonparallel to sweep-line direction.

In this paper, a sweep-line algorithm, for general polygonal shapes with islands, is developed. The developed algorithm can be applied to find self-intersection points, even if the sweep-line was parallel to one or more line-segment in the polygon. Also, invalid-loops detection and removing algorithm are proposed. The proposed algorithm has been implemented in Visual C++, and extensively tested for several polygonal shapes. The results show robustness, and quickness of the developed algorithm for offsetting general polygonal shapes with islands.

2. Definitions and Terminology

This section contains some preliminary definitions and terms that are used throughout this paper. The following definition of a monotone chain is based on those of Preparata *et al.* [19] and Park *et al.* [17]. To handle the chains with line-segments parallel to sweep-line, parallel monotone is suggested in this paper.

2.1. Definition of Chain

Chain is a connected sequence of line segments, and a polygon is a chain that is closed and non self-intersecting [17]. It is assumed that a consecutive collinear sequence of line-segments is merged together into a single line segment.

*Corresponding author:
E-mail : tawfikhtm@yahoo.com

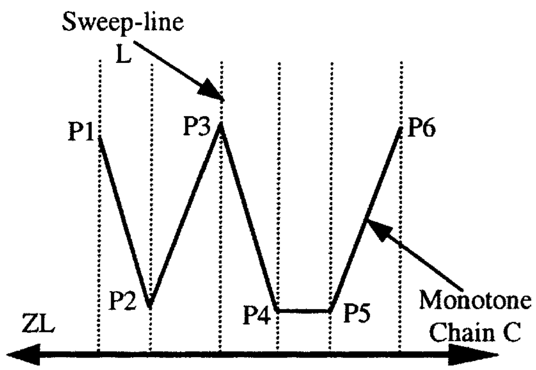


Fig. 1. Monotone chain w.r.t line L.

2.2. Definition of Monotone Chain

A chain C in Fig. 1 is a monotone with respect to a line ZL, if C has at most one intersection point with a line L perpendicular to ZL [17]. The line ZL is called the monotone direction, and the line L becomes a sweep line. It is assumed that ZL-line has an x-axis direction. There are two types of monotone: non-parallel monotone (which contains no parallel line-segment to sweep-line direction), and parallel monotone (which is only one line segment parallel to sweep-line direction).

2.3. Definition of Parallel Monotone Chain (PMC)

The Monotone is parallel, if it has at most one line-segment whose direction is parallel to sweep-line Fig. 2. It has also two vertices (P1, P2) i.e. two sweep-lines L_1 and L_2 at P1 and P2 respectively. The two sweep-lines are overlapped. It is assumed that the sweep-line L_1 intersects the PMC at point P1 and sweep-line L_2 intersects the PMC at point P2. While traversing a chain, each of the locals "extreme" points (with respect to their x-values) are marked either as a left or right-extreme point and up or down-extreme point as follows:

2.4. Definition of Extreme Point

A point in a chain is called a left-extreme and/or right-extreme point, if its x-value is locally minimum or maximum. The monotone contains right & left extreme point [17]. PMC contains two points: the first point (P1 Fig. 2) is an up-extreme point and the last point (P2 Fig. 2) is a down-extreme point, like right and left extreme points in general chain (non parallel chain).

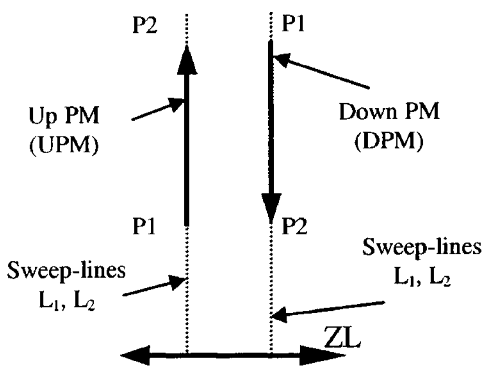


Fig. 2. Parallel Monotone Chain (PMC).

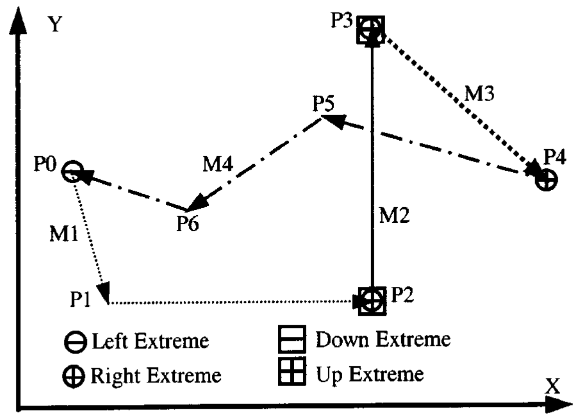


Fig. 3. Monotone & extreme points.

2.5. Definition of Sweep Step (SS) & Monotone Sweep Value (MSV)

Sweep-step (SS) is the x-coordinate of SL, and intersection of SL with certain monotone is called monotone sweep value (MSV)

2.6. Monotone Chains & Extreme Points

Shown in Fig. 3 are local extreme points of a closed polygonal chain consisting of 7 points (or 7 line-segments): There are two left-extreme points, P0 and P3, two right-extreme points, P2 and P4, one up-extreme point, P3, and one down-extreme point, P2. The chain can easily be divided into monotone chains. Since left & right-extreme points, up & down-extreme point alternate, each sequence of the line-segments starting from left to right-extreme point or from up-to down-extreme point (or vise versa) is identified as a monotone chain. The sweep-line steps are defined at vertices of polygonal chain and sorted by a quick-sort algorithm. It is assumed that a vertical sweep-line is used in the developed algorithm. There is no problem, if the chains contain line-segments in parallel with the sweep-line through using of PMC, i.e. the fundamental limitation of the Park *et al.* [17] sweep-line method is removed.

3. Sweep-line Algorithm (SL)

The proposed polygonal-chain intersection algorithm mainly works on a set of monotone chains. The properties of a monotone chain are: (1) it has no self-intersections among its line segments, and (2) its points are in a sequence order of x-values allowing an efficient use of the sweep-line method. The following is the explanation of sweep-line algorithm.

//Sweep Line Algorithm
SweepLine (Array of Points [n])

```
{
    Polygon←Convert points data to lines /* n-lines */ and store them in a polygon;
    Polygon←Filter;
    /* Remove collinear and close this polygon if not closed */
    Calculate extreme point (Polygon);
    /* Left or right and up or down for parallel monotone */
    PolygonToMonotones←Convert polygon data to m Monotones;
    SweepLineArray←Find sweep-lines array;
    for i←0 until n do
        for j←0 until  m do
            if monotone=PMC and sweep-line is 1st sweep then
                take 1st of PMC as intersection points;
            else if monotone=PMC and sweep-line is 2nd sweep then
                take 2nd of PMC as intersection points;
            else if monotone j intersects sweep-line i then
                Find y-intersection between sweep-line i and Monotone j and store them in
SLV[i][j];
    for i←0 until m-1 do
      for j←i+1 until j<i do
        for k←0 until n do
          for kk←k+1 until kk<k do
          {
            if (sweep-line k intersects monotone i, j and sweep-line kk intersects monotone j, i respectively) then
                Find intersection point;
                /* Call intersection function */
            else
                continue; /* There is no intersection found */
          }
}
PolygonToMonotones (Array of Points [n])
{
    Make min. left extreme is the first point of polygon;
    for i←0 until n do
    {
        if (Line[i].DX>0) then
            Define increasing monotone;
        else if (Line[i].DX<0) then
            Define decreasing monotone;
        else
            Define PMC;
    }
}
SweepLineArray(Array of Points [n])
{
    for i←0 until n do
    Define sweep-line as a line through this point with length=SWEEP_LENGTH;
```

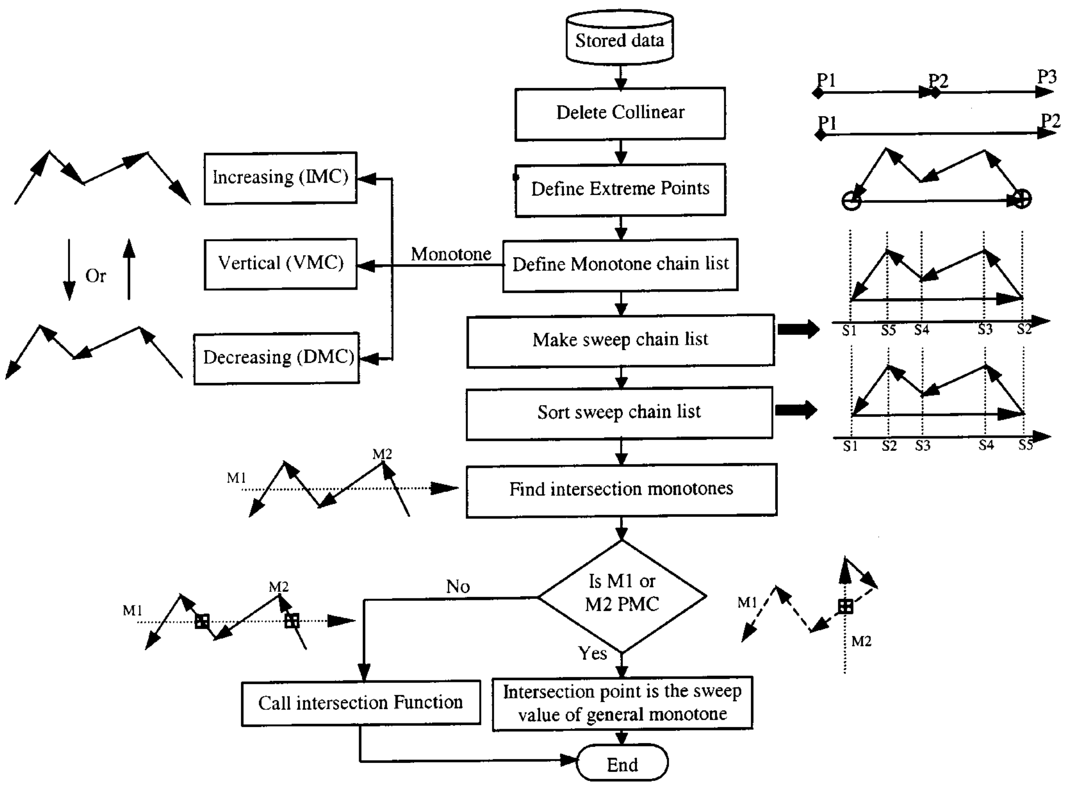


Fig. 4. Data flow through sweep-line algorithm.

/ where SWEEP_LENGTH is the length of sweep-line */*
Use quick-sort algorithm to sort sweep-lines based on x-coordinates;
}

Where: n is the No. of points or line, m: is the No. of monotones; Line.DX=X2-X1 where X2, X1 is the x-coordinates of line end points.

4. Information Flow through the Sweep-line algorithm

The point data are exported from CAD system in DXF format and imported to data filter, Fig. 4. In this step collinear points are removed, and stored in monotones. These monotones are stored in monotone chain. The sweep-line values are sorted and stored in sweep chain. And then, monotone-intersection module will find self-intersection points.

5. Island Making Algorithm (IM)

The IM algorithm is proposed to handle the island during pocketing, Fig. 5. The algorithm contains the following steps:

Step 1: Detect boundary and island polygons, make boundary CCW, and island CW direction.

Step 2: Make one outward offset for island and inward offset until it meets for boundary up to meet island offset. Store the generated offset of boundary and island in temporary polygons.

Step 3: Use SL to find self-intersection points for the temporary polygon, and call detection valid polygons [23] (DVP) to find all valid polygons.

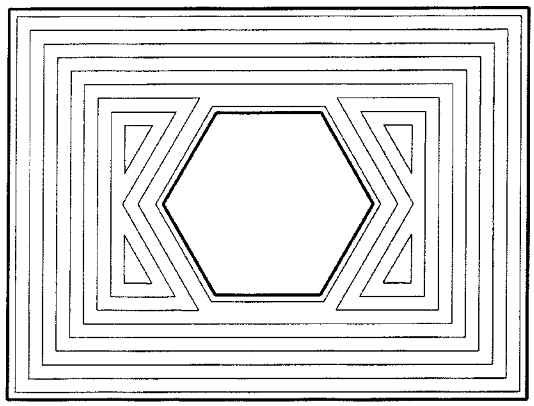


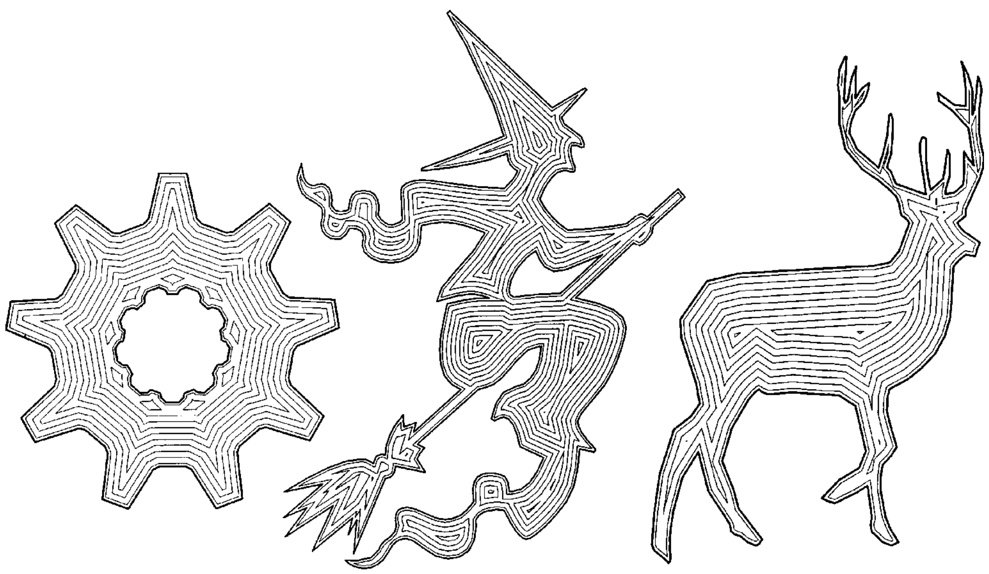
Fig. 5. Island making.

6. Applications

This section contains two parts: Part I, shows the execution time for three-sample examples vs. offset distance for full offset. These sample examples are performed on PIII-800 MHz PC, the execution time calculated through a built-in Visual C++ function (refer to Fig. 6). It is assumed that the offset in the first generated polygon=50% from a cutting tool diameter and 90% for the remaining generated offset. Part II, shows the relationship between the number of offset and the execution time.

6.1. Part (I) Execution Time for Full Offset

As shown in Table 1, the execution time is decreasing while the offset distance is increasing. This variation is more significant for small offset distance, and almost



(a) Bearing Holder (b) Magician (c) Deer

Fig. 6. Proposed system sample example.

Table 1. Sample example execution time vs. offset distance

Offset (mm)	Bearing Holder ms	Deer ms	Magician ms
1	1128	522	2685
7	202	127	555
10	132	127	438
15	106	93	473
20	90	85	305
25	75	82	285

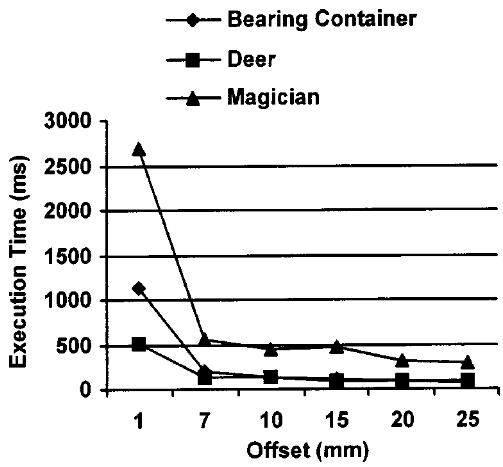


Fig. 7. Execution Time vs. offset (for Full offset).

linearly with large values of offset distance. These results are plotted on a line-chart shown in Fig. 7 for three-sample examples (bearing holder, deer, and magician).

6.2. Part (II) Effect of Repeated Offset

The relationship between the number of offset and the execution time for the three-sample example is given in Fig. 8.

This figure shows that the execution time is increasing linearly for small values of offset and becomes almost

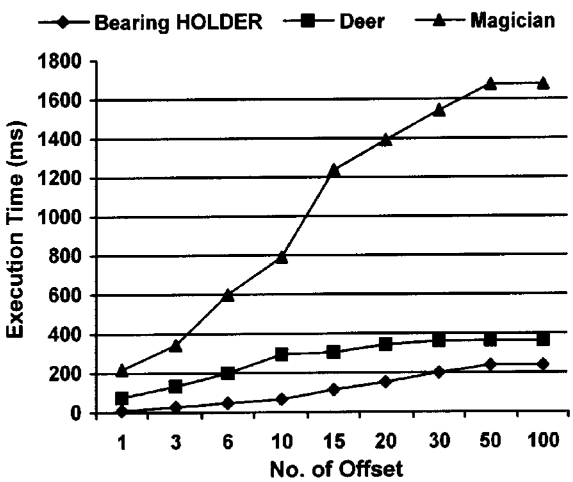


Fig. 8. Execution time vs. No. of offset.

Table 2. Execution time vs. No of offset

No. of Offset	Bearing Holder	Deer	Magician
1	11	75	215
3	24	138	345
6	51	201	600
10	70	294	787
15	114	303	1235
20	152	344	1390
30	198	359	1541
50	241	359	1674
100	241	359	1674

constant for large values of repeating offset. This constant variation occurs when the repeating offset value becomes closer to the full offset values of the application. Also, the figure shows that Magician starts with increasing rather sharply than Bearing Holder and Deer. This sharp variation depends on the complexity of the shape and the number of islands.

7. Conclusion

Presented in this paper an efficient line-offset algorithm for general polygonal shapes with islands. A developed sweep-line algorithm (SL) is introduced to find all self-intersection points accurately and quickly. The developed algorithm is a considerable improvement over previous work algorithms which were limited to handle polygons that having no line-segments in parallel to sweep-line directions. The proposed algorithms are tested through several application examples.

References

[1] Bentley, J. L. and Ottmann, T. A. (1979), "Algorithms for reporting and counting geometric intersections", *IEEE Transactions on Computers*, 643-647.

[2] Chazelle, B. and Edelsbrunner, H. (1992), "An Optimal algorithm for intersecting line segments in the plane", *Journal of the Association for computing machinery*, **39**(1), 1-54.

[3] Chen, Y.J. and Ravani B. (1987), "Offset surface generation and contouring in computer-aided design", *Journal of Mechanisms, Transmissions and Automation in Design: ASME Transactions*, **109**(3), 133-142.

[4] Choi, B.K. and Kim, B.H. (1997), "Die-cavity pocketing via cutting simulation", *Computer Aided Design*, **29**(12), 837-846.

[5] Choi, B.K. and Park, S.C. (1999), "A pair-wise offset algorithm for 2D point-sequence curve", *Computer-Aided Design-31*, 735-745.

[6] Dobkin, D., Guibas, L., Hershberger, J. and Snoeyink, J. (1988), "An Efficient algorithm for finding the CSG representation of a simple polygon", *Computer Graphics*, 31-40.

[7] Gavrilova, M. and Rokne J.G. (2000), "Reliable line segment intersection testing", *Computer Aided Design*, 737-745.

[8] Hansen, A. and Arbab, F. (1992), "An algorithm for generating NC tool paths for arbitrarily shaped pockets with islands", *ACM Transactions on Graphics*, **11**(2), 152-182.
[9] Held, M. (1991), On the computational geometry of pocket machining, *Berlin, Germany: Springer-Verlag.*
[10] Held, M. (2001), "VRONI: An Engineering Approach to the Reliable and Efficient Computation of Voronoi Diagrams of Points and Line Segments", *CGTA.*
[11] Internet web-site (**IWS**-1), "Algorithms for Intersecting Segments" <http://www.cs.jhu.edu/~goodrich/teach/geom/> and <http://www.cs.jhu.edu/~goodrich/teach/geom/notes/intersection.ps>
[12] Internet web-site (**IWS**-2), "An efficient algorithm for calculating red and blue line segment intersections" -site. <http://www.cs.unc.edu/~mantler/258/prop_slides.html>.
[13] Kalmanovichf, G. and Nisnevich, G. (1998), "Swift and stable polygon growth and broken line offset", *Computer-Aided Design*, **30**(11), 847-852.
[14] Kuragano, T., Sasaki, N., and Kikuchi, A. (1988), "The FRESDAM system for designing and manufacturing freeform objects", *In: Martin R, editor. USA-Japan Cross Bridge. Flexible Automation*, 93-100.
[15] Li, H., Dong Z., and Vickers, G. W. (1998), "Optimal Tool Path Generation for 2½-D Milling of Dies and Molds", *SSM'98 Sculptured Surface Machining Conference.*
[16] Mehlhorn, K. and Naher, S. (1994), "An Implementation of a sweep line algorithm for the straight line segment intersection problem", *Technical Report No. MPI-I-94-160. Max-Planck-Institut fur Informatik*.
[17] Park, S.C., Shin H., and Choi B.K. (1998), "A sweep line algorithm for polygonal chain intersection and its applications", *Proceedings of IFIP WG5.2 GEO-6 Conference in Tokyo University*, 187-195.
[18] Pham, B. (1992), "Offset curves and surfaces: a brief survey", *Computer Aided Design*, **24**(4), 223-231.
[19] Preparata, F.P. and Shamos, M.I. (1985), "Computational geometry-An introduction", *Springer Verlag, New York*.
[20] Rohmfeld, R.F. (1998), "IGB-offset curves-loop removal by scanning of interval sequences", *Computer Aided Geometric Design*, **15**(3), 339-375.
[21] Schrder, P. (CS138A 1999), "Line Segment Inter-section".
[22] Takashi, M. (1999), "An overview of offset curves and surfaces", *Computer Aided Design*, 165-173.
[23] Yang, S.N. and Huang, M.L. (1993), "A new offset algorithm based on tracing technique",' *Second ACM/IEEE Symposium on Solid Modeling and Applications, Montreal, Canada*, 201-210.