2003 정보 및 제어 학술회의 (2003.11.21-22)

접원의 전방향 경로이동에 의한 오프셋 알고리즘

An offset algorithm with forward tracing of tangential circle for open and closed poly-line segment sequence curve

윤 성 용*, 김 일 환**

* 강원대학교 제어계측공학과(전화:(033)250-6347, 팩스:(033)242-2059, E-mail : yong0386@hanmail.net)
** 강원대학교 전기전자정보통신공학부(전화:(033)250-6347, 팩스:(033)242-2059, E-mail : ihkim@kangwon.ac.kr)

Abstract : In this paper we propose a efficient offset curve construction algorithm for C^0-continuous Open and Closed 2D sequence curve with line segment in the plane. One of the most difficult problems of offset construction is the loop problem caused by the interference of offset curve segments. Prior work[1-10] eliminates the formation of local self-intersection loop before constructing a intermediate(or raw) offset curve. ,whereas the global self-intersection loop are detected and removed explicitly(such as a sweep algorithm[13]) after constructing a intermediate offset curve. we propose an algorithm which removes global as well as local intersection loop without making a intermediate offset curve by forward tracing of tangential circle. Offset of both open and closed poly-line segment sequence curve in the plane constructs using the proposed approach.

Keywords : offset algorithm, 2D curve, 2D point-sequence, poly-line offset;

Ⅰ. 서론

오프셋 곡선은 원 곡선(profile curve)으로부터 항상 일정한 거리를 유지하도록 원 곡선의 곡선 성질을 이용해 만든 곡선을 말한다. 오프셋 곡선은 컴퓨터그래픽, NC 머신을 위한 툴패스 생성, VLSI circuit design, robot path planning 등 다양한 적용분야를 가지고 있는 중요한 오퍼레이션이다[1-15].

오프셋 곡선은 NURBS나 Bezier 곡선과 같은 곡선에서 오프셋 곡선을 만드는 방법과 점열곡선으로부터 오프셋 곡선을 만드는 방법으로 크게 나눌 수 있다[1-15].

NURBS, Bezier, B-spline, spline 등의 곡선으로부터 오프셋을 만드는 경우(곡선오프셋)에는 exact, 근사치, Geodisic 오프셋 등의 방법이 있고[1,3,4,6,7,8,9,10,15], 점열곡선(polygonal chain or curve)으로부터 오프셋을 만드는 방법(점열곡선 오프셋)에는 pair-wise 오프셋, Voronoi diagram, 그리고 픽셀을 기반으로 접근한 방법 등이 있다[2,5,8,11,12,13,14].

곡선 오프셋이나 점열곡선 오프셋 모두 자기교차(self-interception)에 의한 국부(local)와 전역(global) 자기교차 루프 문제가 발생하고 이 두 가지 문제를 해결하기위한 방법이 연구되어져 왔다. 자기교차에 의해 발생되는 오프셋곡선과 원곡선과의 오프셋거리를 유지하지 못하면 오프셋 곡선으로는 부적당하기 때문에 오프셋곡선 연구에서는 중요한 부분을 차지한다[1-15]. 이들을 해결하기위한 방법으로 국부자기교차에 대해서는 알고리즘자체에서 없어지는 방법[1,2,5,11,12]과 기본오프셋곡선(raw offset curve)를 만든 후 기하적인 자기교차 루프 테스트를 이용해 삭제하는 방법[3,4,6,8,10]이 있고, 전역 자기교차에 대해서는 알고리즘자체에서 없어지는 방법[11,12]과 기본오프셋곡선를 만든 후 기하적인 자기교차 루프 테스트를 이용해 삭제하는 방법[1,2,3,4,5,6,8,10]이 있다. 기본오프셋곡선을 만든 후 기하적인 특성에 기인하여 자기교차를 삭제하는 경우에는 원곡선과 기본 오프셋곡선의 진행 방향에 따라 자기교차를 판단하게 되는 데 원곡선의 모양이 복잡해질수 록 판단이 용이 하지 않을 뿐만 아니라 잘못된 판단으로 오프셋곡선이 제대로 만들어지지 않는 경우가 발생할 수 있다 또한 점열곡선의 자기교차를 판단하는 sweep 이론[13]은 알고리즘이 복잡하기 때문에 오프셋곡선을 만드는 과정보다 자기교차를 판단하는 과정이 더 복잡해지는 단점을 가지고 있다 이에 대해 제2장 오프셋문제에서 논한다.

본 논문은 평면상에서 LS(Line Segment)가 순차적으로 연결된 곡선으로 열린(open) 또는 닫힌(closed) PLS 곡선(Poly-Line segment Sequence curve)을 그 대상으로 한다. LS로 연결된 곡선은 점열곡선에 속하지만 점열곡선의 오프셋 생성 알고리즘은 닫힌곡선이 되어 있어야 하는 전제조건을 바탕으로 적용되는 알고리즘이므로 열린곡선에는 적용하지 못하는 제한이 따른다[2,5,11,12].

따라서, 본 논문에서는 열린곡선과 닫힌곡선에 모두 적용될 수 있고, 자기교차문제를 쉽게 해결하는 방법으로 접원의 전방향 이동방법에 의한 오프셋(foward tracing of tangential circle) 알고리즘을 제안한다.

본 논문의 나머지 구성은 다음과 같다. 먼저 2장에서 오프셋곡선 생성에서 발생하는 문제에 대해 논하고, 논문에서 사용하는 정의를 3장에서 설명하고, 4장

에서 본 알고리즘의 대략적인 개요, 5장에서 본 알고리즘에 대한 설명, 6장에서 실험결과 및 토론, 마지막으로 7장에서 결론을 낸다.

II. 오프셋 문제

1. 자기교차(self-intersection)

평면상의 곡선의 오프셋은 두 개의 파라미터 s, t가 $s \neq t$일 때

$$r(s)+dn(s) = r(t)+dn(t)$$

인 곳에서 발생한다[15]. 여기서 $0 \leq s \leq 1, 0 \leq t \leq 1$이다. 이런 자기교차는 그림 1(a) 국부 자기교차와 그림1(b) 전역 자기교차가 있다.

자기교차는 오프셋곡선에선 발생할 수밖에 없는 문제이기 때문에 자기교차 문제를 해결하기위한 연구가 행해져왔다[1-15]. 이들 연구 중에 여기서 논하고자하는 부분은 원곡선에서 기본 오프셋 곡선을 만든 후 기하학적 특성을 이용해 오프셋 곡선의 자기교차를 검출하고 삭제한 후 삭제하고 남은 부분으로 오프셋 곡선을 만드는 경우에 생기는 문제점이다.

오프셋곡선의 자기교차는 원곡선과 같은 방향으로 만들어지지 않은 부분이면 삭제하게 된다[1,2,3,4,5,6,8,10,13]. 예로 들어, 그림 1은 Lee[8]의 논문의 fig.6.Decision of the eliminated self-intersection loop에서 발췌한 그림인 데 그림 1에서 원곡선이 CCW 방향으로 진행되는 곡선일 때 오프셋 곡선에서 국부와 전역은 CW방향으로 만들어지게 되고 오프셋곡선의 CW인 부분은 삭제되고 최종적으로 CCW인 부분이 원곡선에 대한 오프셋곡선으로 만들어 진다.

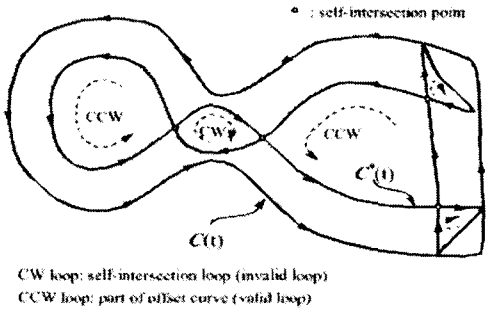


그림 1 삭제할 자기교차루프의 판단

어느 정도 타당하지만 아래와 같이 원곡선에 대해 오프셋곡선이 만들어질 때 문제가 발생한다. 그림 2를 보면 곡선에 대한 오프셋 곡선에 국부와 전역 자기교차가 있다 원곡선이 CCW일 때 오프셋곡선에서 국부 자기교차는 해결되나 전역 자기교차는 삭제되지 않는 문제가 발생한다. 방향으로 자기교차를 판단하는 데 문제가 있다는 것을 알 수 있다. 이에 대한 해결방안으로 본 논문에서는 기본 오프셋곡선을 만들고 나서 오프셋곡선을 만드는 것이 아니라 접원의 전방향 경로이동으로 오프셋 곡선을 만드는 것에 의해 국부 및 전역 자기교차 문제가 없어지게 됨을 보이겠다.

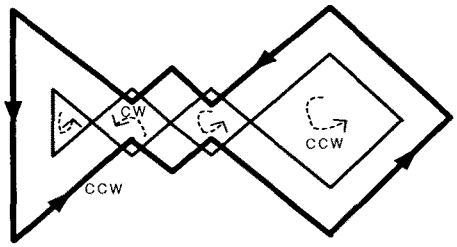


그림 2 전역 자기교차에서 방향 판단 오류

2. 열린곡선의 오프셋 생성에서의 문제점

그림 3(a)에 굵은 실선으로 된 부분은 원곡선이 주어져 있는 데 기본 오프셋곡선을 먼저 만든 후 기하적인 특성에 따라 본 오프셋곡선을 만드는 경우에 그림 3(b)처럼 부적당한 오프셋곡선이 만들어 질 수 있다

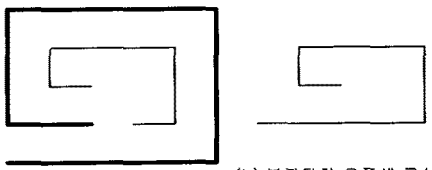


그림 3 열린곡선의 오프셋 생성에서의 문제점

또한, pair-wise[2] 방법에 의해 그림 3(a)의 오프셋곡선을 만드는 경우에 씨앗점(seed point)를 어디서부터 시작하느냐에 따라 그 결과가 달라진다. 그림 4(a)는 적당한 오프셋이 생성되지만 그림 4(b)는 부적당한 오프셋이 생성된다. 점선부분은 설정된 씨앗점에 따라 PWID 테스트를 해서 삭제된 부분이다. 일단 PWID 테스트를 해서 무효한 부분으로 판단되면 오프셋 생성과 관계없이 삭제되기 때문에 닫힌 PLS에서는 문제가 없지만 열린 PLS에서는 그림 4(b)와 같은 문제가 발생하게 된다.

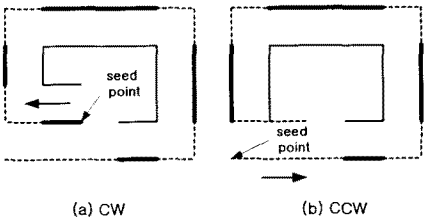


그림 4 열린곡선의 오프셋 생성에서의 문제점(계속)

반면, 본 논문에서 제안하는 FTTC에서는 시점에서 종점까지 경로이동을 하면 간섭이 발생하는 부분이 실제로 오프셋 생성과 관련 있는 부분을 모두 판단하여

오프셋을 만들므로 그림 5(a)처럼 CW 방향으로 오프셋을 만들던 그림 5(b)처럼 CCW 방향으로 만들던 항상 같은 오프셋이 생성된다. 다음 장에서 접원의 전방향 경로이동방법에 의한 오프셋 생성 알고리즘에서 사용하는 정의를 설명한다.

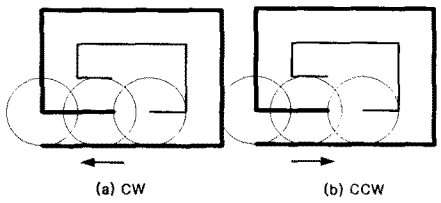


그림 5 열린곡선의 오프셋 생성에서의 문제점(계속)

III. 정의

1. Poly-Line segment Sequence curve(PLS 곡선)

PLS 곡선은 평면상에서 LS(Line Segment)와 LS가 순차적으로 연결되어 있는 곡선으로 시작되는 시점과 끝난는 종점이 주어져야 한다. 즉, PLS 곡선에 속하는 n개의 연속적인 LS는 연속적인 모든 LS_i와 LS_{i+1}에서 LS_i의 종점과 LS_{i+1}의 시점은 일치한다. 단 열린곡선인 경우에는 LS_0의 시점과 LS_{n-1}의 종점이 다르다.

그림6은 열린(open) 또는 닫힌(closed) PLS 곡선를 보인다. 그림6.(a) 열린 PLS 곡선은 5개의 점과 4개의 LS로 되어있고, 그림6.(b) 닫힌 PLS 곡선은 8개의 점과 8개의 LS로 되어있다.

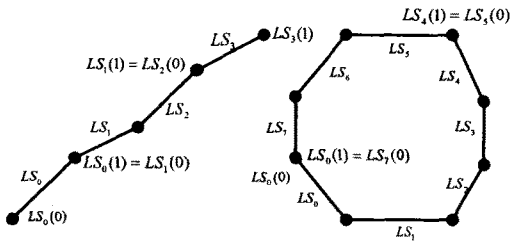


그림 6.PLS곡선 (a)열린 PLS 곡선 (b)닫힌 PLS 곡선

2. 오프셋 방향

닫힌 PLS 곡선의 오프셋은 원래의 PLS 곡선에 대해 안쪽과 바깥쪽 오프셋으로 나누어지지만 열린 PL 곡선에 대해서는 원곡선에 대해 만들어지는 방향에 따라 오른쪽(안쪽) 오프셋, 왼쪽(바깥쪽) 오프셋으로 나눌 수 있다. PLS 곡선에서는 오프셋 방향에 따라 왼쪽 또는 오른쪽 오프셋으로 사용한다.

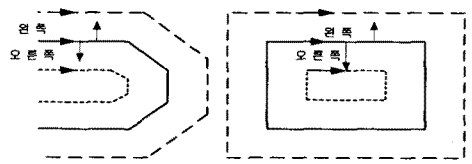


그림 7. 오프셋 방향

3. 연결점의 종류(Vertex)

선분과 선분이 연결되는 각도에 따라 두선분이 연결되는 교점이 이루는 각(internal angle)이 π인 경우인 직선점(tangent vertex), 왼쪽 방향에 대해 π보다 작은 경우인 오목점(concave vertex), π보다 큰 경우인 볼록점(convex vertex)이다.

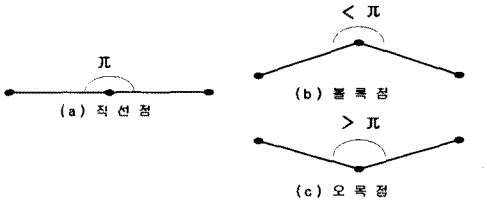


그림 8 연결점 (a) 직선점 (b)블록점 (c)오목점

4 접원(Tangential circle)

곡선에 대한 오프셋 곡선은 다음과 같이 정의된다.

$$Co(t) = C(t) \pm wN(t)$$

여기서 w는 오프셋 폭 또는 거리이고, +w는 왼쪽오프셋,-w는 오른쪽오프셋이 된다.

N(t)는 원곡선 C(t)의 unit normal로 아래와 같다.

$$N(t) = \frac{(y'(t), -x'(t))}{\sqrt{x'(t)^2 + y'(t)^2}}$$

반지름 w인 원이 LS 위에 있는 한점에서 접하는 원을 접원이라고 하고 하나 이상의 LS와 접하는 원을 공통접원이라고 한다. LS에 접하는 접원의 중심은 오프셋 곡선식과 선분방정식에서 구할 수 있다. 평면상의 두 점 A(x1,y1), B(x2,y2)를 지나는 선분을 LS(t)=(x(t),y(t))라 할 때 매개변수 t를 $0 \le t \le 1$이라고 하면,

$$x(t) = (1-t)x1 + tx2$$
$$y(t) = (1-t)y1 + ty2$$
$$x'(t) = (x1+(x2-x1)t)' = x2 - x1$$
$$y'(t) = (y1+(y2-y1)t)' = y2 - y1$$
$$N(t) = \frac{(y2-y1, x1-x2)}{\sqrt{(x2-x1)^2 + (y2-y1)^2}}$$
$$C(t) = LS(t) \pm d \cdot N(t)$$

로 나타낼 수 있다.

IV. 알고리즘의 개요

오프셋 곡선의 정의에 따라 오프셋곡선은 모든 PLS에 대해 OPLS(Offset PLS)에서 오프셋거리에 존재해야 한다는 이유에서 접원을 이용해서 OPLS 생성하면 된다는 것이 본 알고리즘의 기본 생각이다. 즉, 주어진 PLS에서 접원을 이동해서 얻은 경로가 OPLS가 되는

것이다. LS는 양끝점으로 구성되므로 OPLS는 PLS와 마찬가지로 LS의 양끝점으로 구성된다.

n개의 PLS 곡선이 주어졌을 때 $LS_i(t)$ (i=0~n-1)에서부터 접원을 이동해서 m개의 $OPLS_j$ (j=0~m-1) 만들게 된다. 제일 먼저 시작점 n=0에서부터 접원을 경로 이동하게 되는 데 그림 9(a)는 열린 PLS에서 시점 $LS_0(0)$에 접하는 원의 중심을 식()로 구하고 그 중심을 OPLS로 저장하게 된다. 종점 또한 마찬가지의 방법으로 $LS_{n-1}(1)$에서 접원의 중점을 구해 OPLS로 만들면 된다.

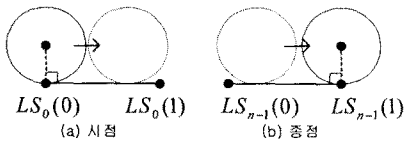


그림 9 시점과 종점

인접한 PLS간의 오프셋 방향에 대한 사이각이 π인 직선점에서는 그림 10처럼 양끝점만 알면 된다는 직선의 성질에 따라 OPLS를 하나만 만든다.

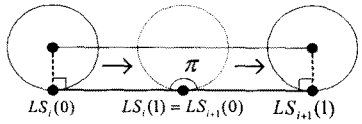


그림 10 직선점의 오프셋

불록점에서는 두개의 인접한 PLS에 대해 OPLS를 만든 후 두개의 OPLS가 만나는 점을 새로운 OPLS의 한쪽 점으로 설정한다. NC 머신의 pocketing과 다른 점인데 이런 교점에서 NC pocketing에서는 호로 설정하지만 여기서는 LS로 설정한다[2,5].

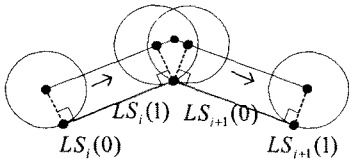


그림 11 불록점의 오프셋

오목점에서는 LS(1)과 LS(0)의 TC의 중점을 구해 OPLS의 한쪽 끝점으로 설정한다.

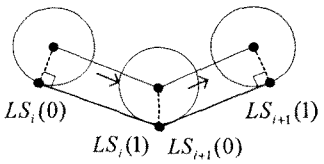


그림 12 오목점의 오프셋

가지점(branch offset point)은 그림 13에 있는 것처럼 두 가지 있는 데 그 첫 번째로 그림13 (a)는 두 개의 LS사이에 존재하고 두 번째는 한 LS와 다른 LS의 불록점이 만날 때 발생한다. 이런 가지점에서는 가지점을 이루는 두 LS 사이에 존재하는 LS를 검사하여 오프셋점이 존재하는 지를 판단 후 오프셋점이 존재하면 가지스택에 그 가지점을 저장하고, LS의 큰 쪽을 먼저 경로이동하여 OPLS를 만든 후 더 이상 OPLS를 만들 수 없을 때 이를 테면, 종점을 만나거나 done 플래그가 TRUE일 때 가지스택에서 저장된 가지점을 가져와 다시 경로이동으로 OPLS를 만들게 된다. 그리고, 오프셋점이 존재하지 않는 경우는 그 부분의 done 플래그를 TRUE로 설정해 다시 경로이동을 하지 않는다. 이런 영역은 국부자기교차가 일어나는 부분으로 여기서 국부자기교차가 자동으로 삭제되게 된다. 자세한 내용은 5장 알고리즘의 설명에서 논한다.

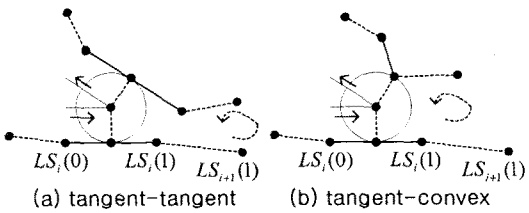


그림 13 가지점의 오프셋

위에서 기술한 방법으로 아래 열린 PLS와 닫힌 PLS에 대해 OPLS를 만든 예가 그림 14와 그림 15에 있다.

V. 접원의 전방향 경로이동에 의한 오프셋

PLS 곡선은 시점에서부터 오프셋 경로이동을 시작해서 모든 LS에 대해 오프셋 경로이동을 하면 끝나게 되고, OPLS(offset PLS)가 만들어 진다. PLS는 i=0에서부터 n-1까지 주어지고 각 LS는 순차적으로 주어진다고 보고, 접원의 경로이동을 하면서 OPLS을 만든다. 접원의 경로이동 중에 가지 구간을 만나면 n이 현재값보다 큰 쪽으로 이동하고, 두 개의 갈래가 나오는 구간에서는 두 갈래 중 n이 큰 쪽으로 이동 한다 이렇게 큰 쪽으로 이동하면서 전진 경로 이동을 하게 되면 앞서 말한 전역 자기교차가 발생하지 않고 OPLS가 만

들어지게 된다.

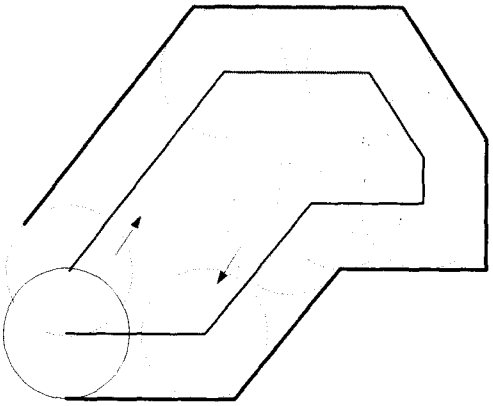


그림 14 열린 PLS 곡선의 오프셋

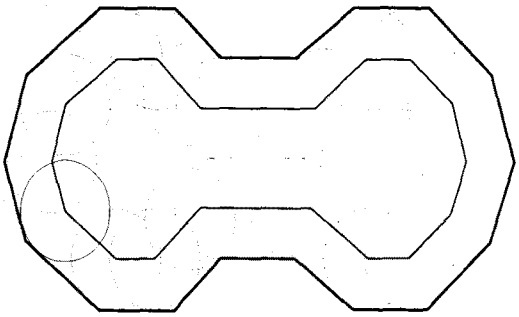


그림 15 닫힌 PLS 곡선의 오프셋

앞서 기술된 오프셋 만드는 알고리즘에서와는 달리 본 알고리즘은 오프셋 경로이동시에 국부자기교차와 전역자기교차들이 동시에 제거되어 기본 오프셋곡선을 만들어 국부와 들을 검사하는 루틴들이 필요 없어진다. 아래는 접원의 전방향 경로이동에 의한 오프셋 알고리즘(Forward Tracing with Tangential Circle Algorithm)을 설명한다. 모두 6단계를 거치게 되는 데 먼저 알고리즘에서 사용하는 변수들을 선언하고, 각 단계별로 설명을 한다.

PLS : PLS는 배열 형식으로 n 개의 순차적으로 연결된 LS가 저장되고 아래와 같은 변수들이 사용된다.
done : done 플래그는 오프셋 생성과 관련된 경로 이동을 이미 한 경우에는 TRUE, 아닌 경우에는 FALSE 가 된다.
t는 경로 이동한 t값으로 $0.0 \le t \le 1.0$
dir : 오프셋 방향 + or -
w : 오프셋 폭 또는 오프셋 거리
τ : 톨런런스로 경로이동 단위이다.
OPLS : 오프셋 PLS이다.
CurPLS : 현재 경로이동 중인 PLS이다.
NextPLS : 다음 이동할 PLS
BranchPLS : 가지구간에서 만나는 PLS
BranchStack: 가지구간에서 가지 오프셋점을 만드는 PLS를 저장하는 스택이다.

```
Procedure Forward tracing with Tangential Circle
input: PLS, τ
output: OPLS
begin
   step 1 :  Initilalize_PLS     //OPLS 및 PLS 초기화
   step 2 :  Make_InitialOPLS
   step 3 :  if Check_BranchPLS == TRUE
               then
                  if Scan_LoopPLS == TRUE
                    then Push_BranchStack
                    goto step 4
                  else
                    Mark_DoneFlag
                  endif
               endif
   step 4 :   Make_NextOPLS
              if terninalPLS == TRUE
                 or NextPLS->done == TRUE
                 then
                 goto step 5
              endif
              go to step 3
   step 5 :   if BranchStackCnt == 0
                 then goto step 6
              else
                 Pop_BranchStack
                 goto step 4
               endif
   step 6 : link OPLS
end
```

각 단계별 설명은 아래와 같다

step 1: FTTC의 초기화 작업을 수행한다.

FTTC의 step 1은 각 변수들의 초기값 설정이다 각 리스트 PLS와 오프셋 리스트 OPLS 초기 PLS의 done 플래그는 모두 FALSE 값을 할당한다.

```
Procedure Initialize_PLS
input: PLS, PLS의 개수 n, OPLS
output: PLS 초기값 설정
begin
  CurPLS  = 0
  NextPLS = 0
  BranchPLS = none
  BranchStackCnt = 0
  i = 0
  while i < n do
       PLS[i].done = FALSE
       PLS[i].t = 0.0
       i++;
  end
end
```

step 2: PLS의 시점에 대한 오프셋점을 구한다.

CurPLS에 대해 t=0.0에서 t=1.0까지 톨러런스 τ만큼 증가 시키면서 3장에서 언급한 오프셋점이 생성될 때까지 증가 시키고 오프셋점을 생성할 수 없을 때는 CurPLS에 done 플래그를 TRUE로 설정한다. 이렇게 done 플래그가 TRUE가 되면 다시 오프셋점을 체크하지 않게 된다. CurPLS를 NextPLS값을 설정한 후 위 과정을 다시 해서 오프셋점이 설정될 때 까지 반복한다.

```
procedure Make_InitialOPLS
input: PLS, τ
output: OPLS[0]
```

```
begin
   ls1 = PLS[0]
   ls2 = PLS[n-1]
   t1 = 0.0
   t2 = 1.0
   FindOffsetPoint
end

procedure Find_OffsetPoint
input: LS1, t1, LS2, t2
output: offset point
begin
  t = t1
  ls = LS1
  while ls≤LS2 do
     while t≤1.0 do
       if ls > LS2 and t>t2
            then return FALSE
       if tangent-vertex == TRUE
            then return TRUE
       else if concave-vertex == TRUE
            then return TRUE
       else if convex-vertex == TRUE
            then return TRUE
       else if tangent-tangent == TRUE
            BranchPLS = ls
            then return TRUE
       else if tangent-convex == TRUE
            BranchPLS = ls
            then return TRUE
       endif
       t = t+τ
     end
     ls = ls+1
     t = 0.0
  end
  return FALSE
end
```

그림 14(a)를 보면 CurPLS=0에서 열린 PLS의 시점으로 PLS[0]와 PLS[1]에 대한 오프셋점을 만들어 진다.

step 3: BranchPLS 검사

step 2에서 오프셋점을 구하면 그 구해진 오프셋점이 가지 구간인지 검사하게 되는 데 가지 구간이면 NextPLS를 BranchStack에 저장한 후 BranchPLS를 CurPLS에 설정한 다음 계속 오프셋점을 찾게 된다.

그림 14(a)를 보면 CurPLS=1지점에서 가지 구간인데 NextPLS=2이고 BranchPLS=12이므로 큰 쪽인 12로 이동한다. 즉, NextPLS는 BranchStack에 저장하고 BranchPLS의 값 12를 CurPLS에 설정한다. Branch Stack에 저장된 PLS는 나중에 접원의 경로이동이 중에 종점까지 경로 이동을 했거나 done 플래그 값이 TRUE, 즉 이미 그 PLS와 관련된 오프셋점이 만들어진 경우에는 더 이상 경로이동을 할 수 없으므로 BranchStack에 저장된 PLS를 읽어서 다시 경로 이동을 시작하게 된다. Check_BranchPLS는 3장에서 언급한 가지가 발생할 수 있는 관계에 있는 오프셋점이 있는지 검사하게 된다. 이런 가지구간이 생길 수 있는 관계는 tangent-convex와 tangent-tangent 오프셋점인 경우이다 그림 14(a)는 tangent-convex 오프셋점으로 가지구간임을 알 수 있다.

```
Procedure Check_BranchPLS
input: PLS, τ
output: OPLS
begin
    if tangent-convex == TRUE
        then return TRUE
    else if tangent-tangent == TRUE
      then return TRUE
    endif
    return FALSE
end
```

가지구간에서는 루프 테스트를 하게 되는 데 가지구간 내에서 오프셋점가 생성될 수 있는 지를 판단한다. 오프셋점이 생성될 수 없는 구간은 done 플래그를 TRUE로 설정하고, 생성될 수 있는 구간은 가지 오프셋점을 BranchStack에 저장한 후 나중에 경로이동을 하게 된다 그림 14 (a)는 tangent-convex 오프셋점을 저장한다.

```
Procedure Scan_LoopPLS
input:  LS1, LS2, t1,t2,τ
output: TRUE or FALSE
begin
   if FindOffsetPoint
     then return TRUE
   endif
   return FALSE
end
```

아래의 프로시저는 가지스택에 가지구간의 PLS를 저장한다.

```
Procedure Push_BranchStack
input: NextPLS
begin
    BranchStack[++BranchStackCnt] = NextPLS
end
```

step 4 : 다음 PLS에 대한 오프셋점을 구한다.

step 4에서는 대한 NextPLS에 설정된 PLS를 경로이동해서 오프셋점를 찾는 작업을 계속하게 된다 .

```
Procedure Make_NextOPLS
input: NextPLS, τ
output: OPLS
begin
    FindOffsetPoint
end
```

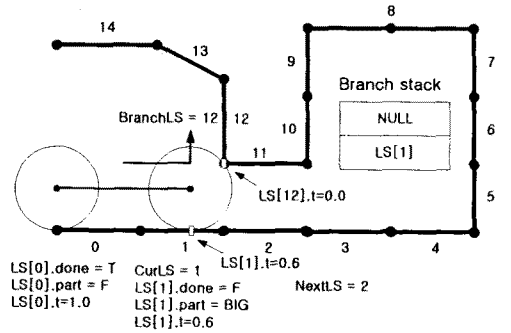
step 5 : 남은 가지구간이 있는 지 검사하고 가지구간이 있으면 BranchStack에서 가지점을 읽어온 다음 계속 오프셋점을 구한다.

남은 가지점이 있는 지의 여부는 BranchStackCnt가 0인지 아닌지를 검사하게 되는 데 0이면 BranchStack에 저장되어 있는 가지점이 없으므로 더 이상 검색할 PLS가 없으므로 step 6으로 넘어간다. 그림 14 (c)처럼 마지막 PLS인 11 경로이동이 끝난 후 종료하고 step 6로 넘어가게 된다.

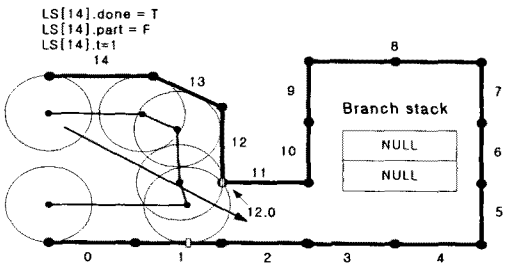
```
if BranchStackCnt == 0
   then goto step 6
else
   Pop_BranchStack
   goto step 4
endif

Procedure Pop_BranchStack
begin
   NextPLS = BranchStack[BranchStackCnt--]
end
```

step 6 : link OPLS

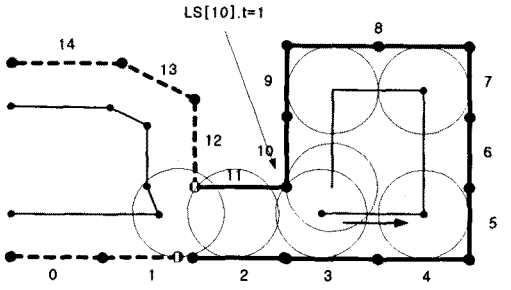
본 단계는 마지막 단계로 만들어진 OPLS들을 연결하여 하나의 OPLS를 만들고 완료하게 된다.



(a) PLS curve and Offset PLS



(b) Offset PLS 1



(c) Offset PLS 2

그림 14 FTTC의 OPLS

Ⅵ. 실험 결과

본 논문에서 제안된 접원의 전방향 경로이동에 의한 오프셋 알고리즘(FTTC)의 실험은 여러 가지 열린 또는 닫힌 PLS 곡선에 대해 수행되었다. 그림 15는 닫힌 PLS 곡선에 대해 CCW 방향에 대해 왼쪽과 오른쪽 방향으로 FTTC를 수행해서 만든 오프셋 곡선이다. 주어진 원 PLS 곡선은 시점과 종점이 연결되어 있고 시점과 종점은 가장 왼쪽 모서리로 CCW 방향으로 연결되어있고, n = 105개의 LS와 104개 연결점으로 구성되어 있고, 이중 오목점 12개, 볼록점 12개 직선점 80개이다.

PLS 곡선의 왼쪽방향에 대해 dir=+1 , w = 5.0mm, τ = 0.001mm로 주어진 파라미터로 생성한 OPLS는 그림 15에서 가장 처음을 넓어 보이는 OPLS로 m=24로 직선점 0개 오목점 12개 볼록점 12 개의 OPLS 1개가 생성된다 네 번째 OPLS는 4개의 tangent-covex 관계인 가지점을 가지고 있어서 4개의 섬(island)으로 만들어 진다 4개의 FTTC의 마지막 단계인 step 6에서 연결된 OPLS 리스트를 구성하는 데 FTTC의 전방향 경로이동 특성에 따라 왼쪽, 아래쪽, 오른쪽, 위쪽 순으로 구성된다. PLS 곡선의 오른쪽방향에 대해 dir=-1 , w = 3.0mm, τ = 0.001mm로 주어진 파라미터로 생성한 OPLS는 그림 15에서 가장 처음을 좁아 보이는 OPLS로 m=24로 직선점 0개 오목점 12개 볼록점 12 개의 OPLS 1개가 생성된다.

그림 16는 열린 PLS 곡선에 대해 CW 방향에 대해 왼쪽과 오른쪽 방향으로 FTTC를 수행해서 만든 오프셋 곡선이다. 주어진 원 PLS 곡선은 시점과 종점이 연결되어 있지 않는 열린 PLS 곡선으로 시점(하단부의 왼쪽 모서리)에서부터 CW 방향으로 연결되어 종점(하단부의 오른쪽 모서리)까지 연결된다 n = 215개의 LS와 216개 연결점으로 구성되어 있고, 이중 오목점 99개, 볼록점 85개 직선점 32개이다.

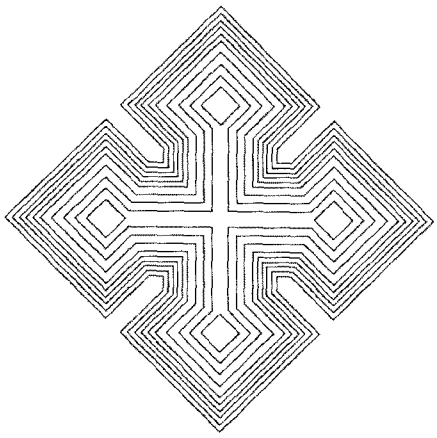


그림 15 닫힌 PLS와 OPLS

PLS 곡선의 오른쪽방향에 대해 dir=-1 , w = 5.0mm, τ = 0.001mm로 주어진 파라미터로 생성한 OPLS는 그림 16에서 가장 처음을 넓어 보이는 OPLS로 m=183로 직선점 0개 오목점 99개 볼록점 85 개의 OPLS 1개가 생성된다. 네 번째 OPLS는 4개의 tangent-covex 관계인 가지점을 가지고 있어서 4개의 섬(island)으로 만들어 진다. 4개의 FTTC의 마지막 단계인 step 6에서 연결된 OPLS 리스트를 구성하는 데 FTTC의 전방향 경로이동 특성에 따라 가운데, 왼쪽, 위쪽, 오른쪽 순으로 구성된다.

열린 PLS 곡선의 왼쪽방향에 대해 dir=+1 , w = 3.0mm, τ = 0.001mm로 주어진 파라미터로 생성한 OPLS는 그림 16에서 가장 처음을 좁아 보이는 OPLS로 m=202로 직선점 0개 오목점 122개 볼록점 81 개의 OPLS 1개가 생성된다.

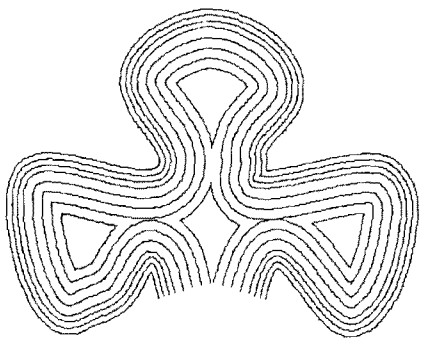


그림 16 열린 PLS와 OPLS

Ⅶ. 결론

본 논문에서 제안된 접원의 전방향 이동방법에 의한 오프셋(foward tracing of tangential circle) 알고리즘은 닫힌곡선이 되어 있어야 하는 전제조건을 바탕으로 적용되는 기존의 점열곡선 오프셋 생성 알고리즘과는 달리, 열린곡선과 닫힌곡선에 모두 적용될 수 있는 알고리즘이다[2,5,11,12].

또한, 2장 자기교차문제에서 기본오프셋곡선을 만든 후 기하적인 특성에 기인하여 자기교차를 삭제하는 경우에는 원곡선과 기본 오프셋곡선의 진행 방향에 따라 자기교차를 판단하게 되는 데 원곡선의 모양이 복잡해질 수 록 판단이 용이 하지 않을 뿐만 아니라 잘못된 판단으로 오프셋곡선이 제대로 만들어지지 않는 경우가 발생할 수 있다 또한 점열곡선의 자기교차를 판단하는 sweep 이론[13]은 알고리즘이 복잡하기 때문에 오프셋곡선을 만드는 과정보다 자기교차를 판단하는 과정이 더 복잡해지는 단점을 가지고 있다

이러한 오프셋 생성에서 문제가 되는 자기교차문제를 알고리즘 내에서 쉽게 해결할 수 있음을 실험을 통해 보여 주었다. Choi[2]는 국부 자기교차는 접원을 이용해 오프셋을 만드는 경우에 간섭 판단과 삭제를 통해 쉽게 삭제할 수 있음을 보여주고 있으나 전역자기교차에 대해서는 sweep 이론[13]에 의해 삭제해야 하는 문제가 있으나 본 논문에서 제안한 FTTC는 알고리즘 자체에서 국부자기교차뿐만 아니라 전역 자기교차도 자동으로 삭제되는 알고리즘이다

FTTC는 주어진 PLS 곡선이 LS가 겹치지 않는 단순곡선(*simple curve*)이고, 어떠한 교점도 반복되지 않는 기본곡선(*elementary curve*)에 제한을 두고 있으므로 이런 곡선에 대한 처리도 수행할 수 있는 방법을 찾는 것이 앞으로의 과제이다

참고문헌

[1] V.D.Holla, K.G.Shastry, B.G.Prakash, "Offset of curves on tessellated surfaces", Computer-Aided Design, 35-12, pp.1099-1108, 10. 2003.

[2] B.K. Choi and S.C. Park, "B.K. Choi and S.C. Park, A pair-wise offset algorithm for 2D point-sequence curve", Computer-Aided Design, 31-12, pp.735-745, 10, 1999.

[3] I.K.Lee, M.S.Kim and G.Elber, "Planar curve offset based on circle approximations", Computer-Aided Design, 28-8, pp.617-630, 10, 1996.

[4] L.A. Piegl, W.Tiller, "Computing offsets of NURBS curves and surfaces.", Computer-Aided Design 31-2, pp. 147 - 156, 2,1999

[5] Hansen.A, Arbab F, "An algorithm for generating NC tool paths for arbitrarily shaped pockets with islands", ACM Transactions on Graphics 11-2, pp. 152 - 182, 8,1992

[6] Shi-Nine Yang,Ming-Liang Huang, "A new offsetting algorithm based on tracing technique", ACM Symposium on Solid Modeling and Applications, pp.201 - 210,1993

[7] G.Elber, I.K.Lee, M.S.Kim, "comparing offset curve approximation methods", IEEE Computer

Graphics and Applications 17-3, ,pp.62-71, 5/6,1997
[8] Eungki Lee, "Contour offset approach to spiral toolpath generation with constant scallop height". Computer-Aided Design 35-6, pp. 511 - 518, 5,2003
[9] G.V.V. Ravi Kumar, K.G. Shastry, B.G. Prakash, "Computing constant offsets of a NURBS B-Rep". Computer-Aided Design 35-10, pp. 935 - 944, 9,2003
[10] G.V.V. Ravi Kumar, K.G. Shastry, B.G. Prakash, "Computing non-self-intersecting offsets of NURBS surfaces". Computer-Aided Design 34-3, pp. 209 - 228, 3,2002
[11] D.S. Kim, B.G. Prakash, "Polygon offsetting using a Voronoi diagram and two stacks". Computer-Aided Design 30-14, pp. 1069 - 1076, 12,1998
[12] M. Held, "Voronoi diagram and offset curves of curvilinear polygons". Computer-Aided Design 30-4, pp. 287 - 300, 3,1998
[13] S.Fortune,"A sweepline algorithm for Voronoi diagrams",Annual Symposium on Computational Geometry,pp.313-322,1986
[14] Park SC, Shin H, Choi BK, "A sweep line algorithm for polygonal chain intersection and its applications", Proceedings of IFIP WG5.2 GEO-6 Conference in Tokyo University, p.187-195 ,12,1998
[15] T.Maekawa, "an overview of offset curves and surfaces.", Computer-Aided Design 31-3, pp.163 - 232, 3,1999