잔재영역 최대 활용 시스템

김승환 · 곽승신 · 김규란 · 권순각
동의대학교 컴퓨터소프트웨어공학과

Maximum Utilization System of Vessel Materials

Seunghwan Kim · Seungsin Kwak · Gyuran Kim · Soonkak Kwon
Department of Computer Software Eng., Dongeui University

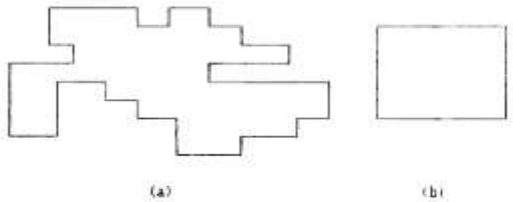
요 약

본 논문은 2D도면을 제작하는 모듈과 기본커널 기능을 이용한 2D판재 네스팅 기능을 구현한 시스템을 활용하여 2D도면으로 만들어진 부품의 형상을 구매된 원자재에 자동/수동으로 배치하면서 자재의 손실을 최소화 할 수 있으며, 절단 경로, DNC, 각종 자재 및 생산정보를 산출할 수 있는 모듈을 개발하여 구현한다. 이 시스템의 기대효과로는 좀 더 편하고 좀 더 정확하게 다양한 원자재를 생산할 수 있도록 하는 것과 오차가 크게 발생했을 시 재작업을 하는 경우를 방지하고, 보다 효율 높은 알고리즘을 응용 개발함으로써 또 다른 분야로 활용성을 높이기 위한 것이다.

1. 서론

목적이 어떠한 기업이라고 해도 기업의 사업목표는 결론적으로 이윤추구이다. 이윤을 끌어올려서 매출증대를 꾀하는 방법은 여러 가지가 있겠지만 그 중 하나는 비용절감부분이다. 비용절감을 위해서는 원, 부자재의 원가를 줄이는 방법이 있으며 관리비차원에서의 절감 그리고 원, 부자재의 필요 없는 부분 감소 등이 있다. 대부분의 기업들은 이러한 사실을 알고 있고 활용하고 있지만 원, 부자재의 필요 없는 부분의 감소에 대해서는 효율적으로 행할 방법이 없다. 그리하여 자재의 남은 부분을 최대한으로 활용하여 소모되는 경제적 비용을 절감하여 결과적으로 기업의 매출증대를 꾀하는 알고리즘이 필요하다. 이 논문에서는 잔재영역 인식과 잔재영역을 최대한 활용하는 것에서 비롯된 오차를 최대한 줄여서 허용오차 범위까지 안전하게 도달하여 좀 더 효율적인 자재비의 절감을 꾀하는 알고리즘에 대해 생각해 본다.

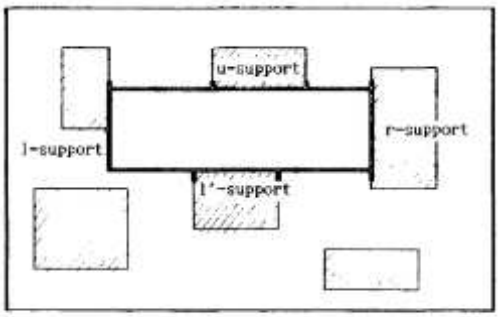
직교다각형 : 기하학적인 문제의 대상으로써 좌표축에 평행한 변으로만 구성되며, 각 변들은 변의 끝점에서만 교차하고 그 외에는 교차하지 않는 다각형



(a) 직교 다각형과 (b) 직사각형

오목 절점 : 직교 다각형에서 내부각이 180도 이상인 절점, 직교 다각형에서의 오목 절

점은 내부각이 항상 270도이다.
볼록 절점 : 직교 다각형에서 내부각이 180도 이하인 절점
최대 직사각형 : 직교다각형이 X축이나 T축에 대해 평행하고 내부에 완전히 포함되면서 네 변의 좌표값에 평행한 직사각형 중에서 가장 넓이가 큰 것



최대 직사각형의 support

2. 알고리즘

본 장에서는 문제 해결을 위한 구체적인 알고리즘 예시를 보여준다. 크게 최대 볼록 다각형 문제와 최대 직사각형 문제로 나뉘어 볼 수 있다.

2-1 단조 직교 다각형

```
Algorithm MRP_TO_MSRPs
    input  : 단조 직교 다각형(monotone rectilinear polygon)인 R
    output : 분할가능 단조 직교 다각형(monotone separable rectilinear
             polygon)으로 나누어진 R

    step 1 :
        e_l ← x좌표값이 가장 작은 W-edge
        e_r ← x좌표값이 가장 큰 E-edge
        C_U ← e_l과 e_r을 연결하는 R의 chain 중에서 N-edge를 포함하는 것
        C_L ← e_l과 e_r을 연결하는 R의 chain 중에서 S-edge를 포함하는 것
        V ← e_l을 제외한 모든 수직의 변을 x좌표값이 증가하는
            순서대로 배열한 것
    step 2 :
        L ← e_l
        e=(u,v) ← V의 첫번째 변    /* y(u) > y(v) */
        reset USTACK and LSTACK
        while x(e) < x(e_r) do
            if empty(LSTACK)
                then y(f) ← -∞
                else f ← bottom(LSTACK)이 지시하고 있는 S-edge
            if empty(USTACK)
                then y(g) ← +∞
                else g ← bottom(USTACK)이 지시하고 있는 N-edge
            case
                e ∈ C_U and y(v) ≤ y(f) : do    /* case 1 */
                    f에서 시작하고 e와 직교하는 선분 m을 이용하여 R을
                    분할한다
                    reset USTACK
                    push(USTACK,m)
                    bpop(LSTACK)
                    L ← f에 연결되어 있는 W-edge
                end
                e ∈ C_L and y(u) ≥ y(g) : do    /* case 2 */
                    g에서 시작하고 e와 직교하는 선분 m을 이용하여 R을
                    분할한다
                    reset LSTACK
                    push(LSTACK,m)
                    bpop(USTACK)
                    L ← g에 연결되어 있는 W-edge
                    end
                otherwise do    /* case 3 */
                    현재 상태의 SL에 대해 USTACK과 LSTACK에 올바른
                    정보가 유지될 수 있도록 USTACK과 LSTACK의 내용을
                    수정한다. 이 때 사용하는 명령은 push와 tpop이다.
                    e=(u,v) ← V에서 다음 변
                    end
            end    /* case */
        end    /* while */
end MRP_TO_MSRPs
```

2-2 분할가능 단조 직교 다각형

```
Algorithm MAX_RECTANGLE1
    input  : 분할가능 단조 직교 다각형(monotone separable rectilinear polygon)
    output : 3 개의 support가 H_L에 존재할 경우의 최대 직사각형

    step 1 :
        임의의 점 p의 right(p)와 left(p)를 구하기 위한 전처리 작업을 수행한다
        query 구간 [x_1,x_2]의 Em(x_1,x_2)를 구하기 위한 전처리 작업을 수행한다
    step 2 :
        for S-edge e_i=(u,v) in H_L do
            w ← left(u)
            z ← right(v)
            f ← Em(x(w),x(z))
            다음과 같은 조건을 만족하는 직사각형 R_i를 찾는다
                (1) ll-corner ← w
                (2) ur-corner ← (x(z),y(f))
        end
    step 3 :
        최대 직사각형 ← R_i (1≤i≤n, n은 S-edge의 갯수) 중에서
                        가장 면적이 큰 것
end MAX_RECTANGLE1
```

2-3 직교 다각형 알고리즘

```
Algorithm MAX_RECTANGLE(R)
    input  : 직교 다각형(rectilinear polygon)인 R
    output : R에 포함되는 최대 직사각형

    step 1 :
        if ( R이 직사각형 ) then do
            최대 직사각형 ← R
            return
        end
```

step 2 :
R을 두개의 작고 나각형 R_1과 R_2로 분할한다.
M_1 ← MAX_RECTANGLE(R_1)
M_2 ← MAX_RECTANGLE(R_2)
H를 끌어낸다
M_3 ← MAX_RECTANGLE1(H)
M_4 ← MAX_RECTANGLE2(H)
최대 직사각형 ← M_1,M_2,M_3,M_4 중에서 가장 면적이 큰 것

end MAX_RECTANGLE

이 알고리즘의 시간 복잡도를 T(N)이라 하면 다음과 같은 식이 성립한다.

(1) N ≤ 4 일 경우
T(N) = O(1)

(2) N > 4 일 경우
$T(N) = T(N_1) + T(N_2) + O(N\log^2 N)$
$(N_1+N_2=N+4,\ N_1 \le N_2 \le 2N/3)$

2-4 최대 직사각형 알고리즘

알고리즘 1 최대 직사각형

입력 : 내부 Hole을 가진 직사각형
출력 : 최대 직사각형

단계 1 : 수평 분할
각 Slab 면적 계산
단계 2 : Slab 확장과 면적 계산
단계 3 : 수직 분할
각 Slab 면적 계산
단계 4 : Slab 확장과 면적 계산
단계 5 : 결과

2-5 수평 분할과 수직 분할

알고리즘 2 수평 분할

입력 : Hole의 Y좌표의 선분
출력 : 수평 분할선

```
단계 1 : Y 좌표에 대해 2N개의 끝점을 정렬(Sorting)하고
        POINT[1:2N]의 배열에 저장
단계 2 : for i:=1 to 2n do
        begin
            p:=POINT[i];          s:=p가 끝점인 선분
            if (p가 아래쪽 끝점) then
            begin
                ABOVE(s)
                BELOW(s)
                INSERT(s)
            end
            else  (* P가 윗쪽 끝점 *)
            begin
                DELETE(s)
                ABOVE(s)
                BELOW(s)
            end
        end
```

알고리즘 3 수직 분할

입력 : Hole의 X좌표의 선분
출력 : 수직 분할선

```
단계 1 : X 좌표에 대해 2N개의 끝점을 정렬(Sorting)하고
        POINT[1:2N]의 배열에 저장
단계 2 : for i:=1 to 2n do
        begin
            p:=POINT[i];          s:=p가 끝점인 선분
            if (p가 왼쪽 끝점) then
            begin
                ABOVE(s);     BELOW(s);     INSERT(s);
            end
            else  (* P가 오른쪽 끝점 *)
            begin
                DELETE(s);    ABOVE(s);     BELOW(s);
            end
        end
```

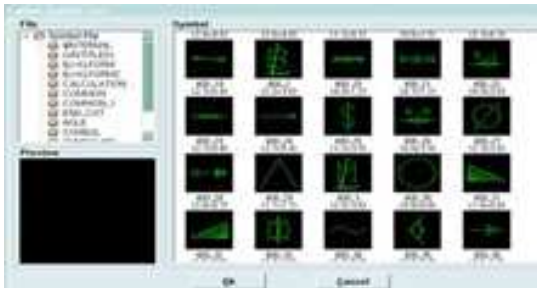
2-6 Slab 확장 알고리즘

알고리즘 4 Slab 확장

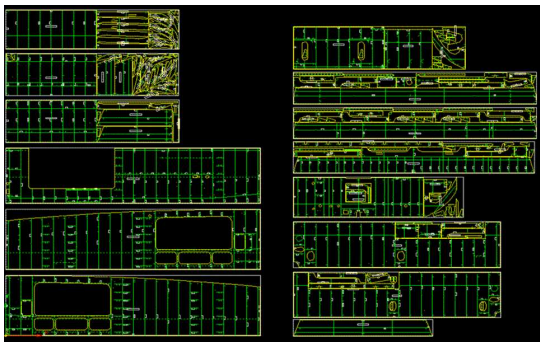
입력 : Slab
출력 : 확장 Slab

```
단계 1. Segment tree 형성
단계 2. While (Slab의 갯수)
        begin
          p:=Slab의 구간
          Case 1 (* 유형 a인 경우 *)
                Segment tree에서 윗쪽에서 제일 가까운
                Hole을 찾는다.
          Case 2 (* 유형 b인 경우 *)
                Segment tree에서 아래쪽에서 제일 가까운
                Hole을 찾는다.
          Case 3 (* 유형 d인 경우 *)
                Segment tree에서 윗쪽에서 제일 가까운
                Hole을 찾는다.
                Segment tree에서 아래쪽에서 제일 가까운
                Hole을 찾는다.
        end
```

<Symbol 형상>



<형상 생성>



3. 결론

본 논문에서는 직교 다각형과 최대 직사각형 문제에 대한 알고리즘을 제시하여 강판 내부의 다각형을 찾는 문제에 대한 해결책을 제시하였다. 이 알고리즘을 이용하여 선분을 픽셀단위로 나누어 선분의 끝 지점을 인식하여 진행되는 방향을 탐색하고 그것을 바탕으로 최대 크기를 갖는 형상을 인식하는 방법으로 잔재영역을 인식하여 활용할 수 있다.

참고문헌

[1] KarenDaniels, Victor Milenkovic, DanRoth, Finding the Maximum area Axis-Parallel Rectangle in a Ploygon, Harvard University Division of Applied Sciences, October 1993.

[2] Jun Cheol Lee, Algorithm for Finding Maximum Rectangle of Rectangle With Holes, Department of Computer Engineering in Kyungpook National University, December 1990.

[3] Choi Seung-Hak, Algorithms for Finding the Largest Convex Subpolygon of a Rectilinear Polygon, Department of Computer Science in Korea Advanced Institute of Science and Technology, December 1986.