구 볼록 다각형들의 분리 및 교차를 위한 간선 기반 알고리즘의 구현

(An Implementation of an Edge-based Algorithm for Separating and Intersecting Spherical Polygons)

하 종 성 † 천 은 홍 †
(Jong-Sung Ha) (Eun Hong Cheon)

요 약 본 논문에서는 구상에서 주어진 볼록 다각형의 집합 $\Gamma=\{P_1,...,P_n\}$의 최대 또는 최소 교차를 결정하기 위하여 다각형의 간선으로 구를 면으로 분할하는 문제를 고려한다. 이 문제는 Γ의 최대 부분집합을 포함하는 반구를, Γ를 분리하는 대원을, Γ를 이분하는 대원을, Γ를 최소 또는 최대 부분집합을 교차하는 대원을 각각 찾는 다섯 가지 기하적 문제와 공통적으로 관련이 있다.

구다각형의 최대 및 최소 교차를 효율적으로 구하기 위하여 우리는 **간선 기반 분할**의 방식을 취하는데 이 방식에서는 구가 각 다각형에 의해 증분적으로 분할되면서 면이 아닌 면을 구성하는 간선의 소유권이 처리된다. 마지막에는 최대수의 소유권을 가지는 분할된 비정렬 간선들을 모아 해가 되는 면들의 경계를 구성하지 않고 그들의 중심을 근사적으로 얻는다.

최대 교차를 찾는 우리의 알고리즘은 효율적인 시간복잡도 $O(nv)$를 가지는 것으로 분석된다. 여기서 n과 v은 각각 다각형과 모든 정점의 개수들이다. 더구나 견고하게 수치를 계산하고 모든 degeneracy 경우를 다루기 때문에 구현의 관점에서도 실제적이다. 유사한 방식을 사용하여 일반적인 교차의 모든 경계는 $O(nv+L\log L)$ 시간에 구성할 수 있다. 여기서 L은 해로 출력되는 간선의 개수이다.

Abstract In this paper, we consider the method of partitioning a sphere into faces with a set of spherical convex polygons $\Gamma=\{P_1,...,P_n\}$ for determining the maximum or minimum intersection. This problem is commonly related with five geometric problems that find the densest hemisphere containing the maximum subset of Γ, a great circle separating Γ, a great circle bisecting Γ, and a great circle intersecting the minimum or maximum subset of Γ.

In order to efficiently compute the minimum or maximum intersection of spherical polygons, we take the approach of **edge-based partition**, in which the ownerships of edges rather than faces are manipulated as the sphere is incrementally partitioned by each of the polygons. Finally, by gathering the unordered split edges with the maximum number of ownerships, we approximately obtain the centroids of the solution faces without constructing their boundaries.

Our algorithm for finding the maximum intersection is analyzed to have an efficient time complexity $O(nv)$, where n and v, respectively, are the numbers of polygons and all vertices. Furthermore, it is practical from the view of implementation, since it computes numerical values robustly and deals with all the degenerate cases. Using the similar approach, the boundary of a general intersection can be constructed in $O(nv+L\log L)$ time, where L is the output-senstive number of solution edges.

1. 서 론

다축 수치제어(NC) 절삭가공에서 최적의 workpiece setup에 관한 연구에서 단위구 S^2상에서 구볼록다각형(spherical convex polygon) 형태인 가시맵(visibility map) 집합의 교차를 조사하는 기하적 문제[1]를 찾아볼 수 있다. 이 교차 문제를 해결하기 위한 연구를 크게 **조각화 기법**(tessellation approach)[2,3]과 **분할 기법**

* 이 논문은 우석대학교 교내학술연구비지원에 의하여 연구됨.
† 종신회원 : 우석대학교 정보통신컴퓨터공학부 교수
jsha@core.woosuk.ac.kr
ehcheon@corc.woosuk.ac.kr
논문접수 : 2000년 11월 27일
심사완료 : 2001년 5월 8일

(partition approach)[1,4,5]의 두 종류로 나누어 볼 수 있다. 조각화 기법은 S^2를 작은 삼각형과 같은 조각의 집합으로 나누어 다각형에 의해 소유되는(교차되는) 영역을 각 조각 단위로 쉽게 계산하는 디지털-맵-지향 방법이다. 분할 기법은 다각형의 경계로 S^2를 면(face)의 집합으로 분할하고 각 면의 소유권을 계산하는 다각형-객체-지향 방법이다.

조각화 기법은 해를 견고하게 계산할 수 있으므로 실제적으로 사용되어오던 방법이다. 그러나 조각화의 정도에 따라 해의 정확성과 계산 시간의 효율성간에는 교환관계가 있다. 한편 분할 기법은 효율적인 시간복잡도안에 모든 해의 경계를 구할 수 있는 것으로 분석되고 있지만 Suh와 Kang[2]이 지적한데로 구현시 수치적 복잡성에 빠질 수 있다.

본 논문에서는 분할 기법중 가장 일반적인 Chen등[1]의 알고리즘을 견고하게 구현할 수 있도록 개선하고 구현한다. 원래의 알고리즘에서는 경계 표현[6,7,8]의 자료구조를 사용하여 분할된 면의 소유권(ownership)을 유지시킨다. 이 알고리즘은 이론적인 시간복잡도 분석에서 매우 우수하지만 구현의 관점에서 이러한 경계 표현은 근접하는 교차점간에 수치적 계산의 한계로 발생하는 위상 위배를 해결해야만 하는 심각한 문제를 안고 있다. 이와 같이 기하적 자료구조를 무효화시키는 위상 위배를 해결하기 위한 몇 가지 기법들[9,10,11,12]이 제시되고 있기는 하지만 구현의 오버헤드가 크거나 완전하지 못하다.

위와 같은 여러 가지 이유로 분할 기법은 아직까지 구체적인 구현 방법이 알려져 있지 않다. 우리의 개선된 알고리즘은 위상 위배에 대한 문제없이 단방향순환리스트[13]의 자료구조를 사용하여 분할된 간선의 소유권을 유지한다. 원래의 알고리즘이 면 기반 분할이라면 우리의 알고리즘은 간선 기반 분할이라 할 수 있다. 이 자료구조는 직교부품을 포함하는 모형에서 자주 나타나는 하나의 정점 또는 간선으로 구성된 가시맵과 같은 degeneracy 경우에도 자연스럽게 처리할 수 있다는 다른 장점이 있다.

이론적인 분석에서 보면 모든 해의 경계를 구하는데 면 기반 분할은 $O(nv\log n)$의 시간복잡도를 가지는 반면 간선 기반 분할은 $O(nv+L\log L)$의 시간복잡도를 가진다. 여기에서 n과 v는 각각 다각형의 개수와 모든 정점의 개수이고 L은 해로 출력되는 간선의 개수이다.

대부분의 최적화 문제에서 사용되는 가시맵의 최대 교차의 경우에는 해의 경계를 구성하지 않고 해 영역의 기하적 특성을 사용하여 각 해 영역의 중심을 근사적으로 구한다. 실제적으로는 해의 경계 자체보다는 경계의 중심이 보통 사용되므로 이 기술은 매우 유용하며 또한 시간복잡도도 $O(nv)$로 축소시킨다.

본 논문에서는 가시맵의 최대 교차의 중심 해를 구하는 부분에 집중하여 알고리즘을 C++ 프로그래밍언어의 클래스로 구현한 과정을 구체적으로 설명하여 관련 소프트웨어를 개발하는데 도움이 되도록 노력하였다. 2장에서 문제 및 알고리즘의 윤곽을 설명하고 3장에서는 클래스의 주요 구현 코드 및 사용법을 설명한다. 4장에서는 이론적인 분석과 실험 결과를 비교하고 5장에서 결론을 내린다.

2. 구상의 기하적 문제의 정의

두 점간에 아무 충돌없이 그들을 잇는 선분이 존재하면 두 점은 서로 **가시적**(visible)이라고 한다. 어느 표면의 **가시맵**은 무한한 곳으로부터 그 표면의 모든 점이 가시적인 모든 방향을 나타내는 구상의 점의 집합이다. 그림 1과 같이 가시맵은 표면의 수직벡터의 집합인 **가우시안맵**과 밀접한 관련이 있다. 어느 표면 S의 가우시안맵 $P=\{p1,...,pm\}$이 주어지면 S의 (양)가시맵은 $\bigcap_{i=1}^{m} HS(p_i)$로 얻어지며 음가시맵은 $\bigcap_{i=1}^{m} HS(-p_i)$로 얻어진다. 여기에서 $HS(pi)$는 반구 $\{x \mid x \cdot pi \geq 0\}$이며 pi는 그 반구의 극(pole)이다.

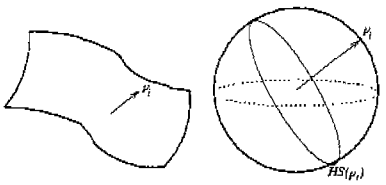

그림 1 표면의 가우시안맵과 가시맵

다축 수치제어 절삭가공에서 최적의 workpiece 방위를 구하는 문제는 가우시안맵 또는 가시맵의 볼록헐을 나타내는 구다각형의 집합 $\Gamma=\{P_1,...,P_n\}$를 사용하여 다음 다섯 가지 구상의 기하적 문제로 형식화되었다[1].

Γ의 최대 부분집합을 포함하는 반구(the densest hemisphere), Γ를 분리하는 대원(separator), Γ를 이분하는 대원(bisector), Γ의 최소 또는 최대 집합을 교차시키는 대원을 구하라.

3. 알고리즘의 개요

3.1 면 기반 분할

두 구다각형 P_1과 P_2에 대하여 서로 **이원**(dual) 관계가 있다고 하는 것은 P_1의 정점 v_i가 $e_i \in \{x \mid x \cdot v_i = 0\}$인 P_2의 간선 e_i와 대응 관계가 있고 즉 e_i가 v_i를 극으로 하는

대원상에 있고 P_1의 정점들은 P_2의 대응되는 간선들과 같은 순서임을 의미한다. 한 표면의 가우시안맵의 볼록헐과 가시맵의 볼록헐과는 서로 이러한 이원성이 있다[14].

주어진 구다각형의 집합 $\Gamma=\{P_1,...,P_n\}$의 이원다각형의 집합 즉 (양)가시맵을 $\Lambda=\{V_1,...,V_n\}$라 하고 대응되는 음가시맵을 $\Lambda^-=\{V_1^-,...,V_n^-\}$라 하자. 그러면 수치제어 절삭가공과 관련된 5 가지 기하적 문제는 Λ와 Λ^-의 교차를 조사하는 통일된 방법으로 해결할 수 있다. 면 기반 분할 방식에서는 Λ와 Λ^-로 S^2를 면의 집합으로 분할하여 각 면의 소유권을 완전히 계산한다. 즉 어느 가시맵이 각 면을 소유하는지 그 면에 **소유권 벡터**를 할당하는 것이다. 한 면의 소유권 벡터의 i 번째 비트는 그 면이 V_i에 포함되면 +1로, V_i^-에 포함되면 −1로, 그렇지 않으면 0으로 각각 세팅된다.

면의 소유권 벡터에 세팅된 +1과 −1의 비트 개수를 세어서 다섯 가지 기하적 문제를 해결할 수 있다. 분할된 면 F_k의 소유권 벡터에서 +1과 −1인 비트의 개수를 각각 n_k^+와 n_k^-라 하자. 그러면 densest hemisphere의 극의 궤적은 n_k^+가 최대인 면이다. separator는 $n_k^+ + n_k^- = n$인 면을 찾는 것이다. 여기서 n은 다각형의 개수이다. bisector는 $|n_k^+ - n_k^-| \le 1$인 separator를 찾는 것이다. Γ를 최소로 또는 최대로 교차시키는 대원은 $n_k^+ + n_k^-$이 각각 최대 또는 최소인 면을 결정하는 것이다. 보다 자세한 내용은 문헌[1]을 참고하면 된다.

3.2 간선 기반 분할

간선 기반 알고리즘은 분할된 간선에 소유권의 정보를 표현하는 것이다. 그림 2에서 보는 바와 같이 $2n$개의 다각형 $\{V_1,...,V_n,V_1^-,...,V_n^-\}$에 의하여 분할되는 각 간선에 그 간선을 소유하는 다각형의 개수가 유지된다고 가정하자. 지금부터 k 개의 다각형에 의하여 소유되는 간선 또는 면을 k-**교차**된다고 말한다. 소유권의 최소값과 최대값을 각각 min과 max라 하면 k-교차되는 면을 찾는 것은 다음과 같은 간선을 찾는 것이다.

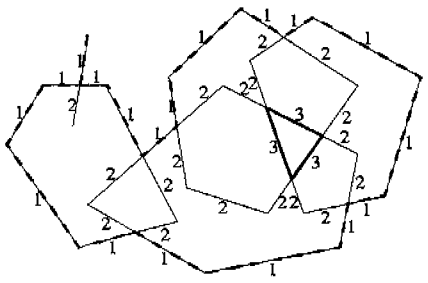

그림 2 분할된 간선의 소유권

- $k=min$인 면: k-교차되는 간선의 외부
- $min<k<max$인 면: k-교차되는 간선의 내부 또는 $(k+1)$-교차되는 간선의 외부
- $k=max$인 면: k-교차되는 간선의 내부

최적화 문제는 k의 값을 극대화시키는 것이다. 즉 min-교차되거나 max-교차되는 면을 찾는 것이다. 즉 k-교차되는 간선만을 찾는 것이다. 수치제어 절삭가공과 관련되는 다섯 가지 구상의 기하적 최적화 문제중에 네 가지는 max-교차되는 면을 찾는 것이고 나머지 한 가지는 min-교차되는 면을 찾는 것이다. 이러한 관찰로부터 우리의 간선 기반 알고리즘은 소유권의 극값을 결정하고 그 값을 가지는 간선을 수집하여 처리한다.

일반적으로 해가 되는 면의 경계를 구하기 위해서는 수집된 간선을 정렬하여야 한다. 그러나 필요한 해는 보통 해의 중심에 있는 하나의 점이면 족하다. 그런데 가장 많이 찾게 되는 max-교차되는 면의 경우에는 항상 볼록하고 그 면안에 홀(hole)이 없다. 더구나 그 해의 경계는 같은 소유권 벡터를 가지는 간선으로만 구성되었다. 이러한 좋은 특징을 이용하여 해의 경계를 구성하지 않고 우리는 경계를 내접하는 원을 찾음으로써 해의 근사적인 중심을 효율적으로 결정한다.

4. 구 현

4.1 클래스의 구조

먼저 주어진 가우시안맵과 이것과 이원성이 있는 가시맵을 나타내기 위한 두 개의 주요 자료구조를 정의한다.

```
typedef  double      tPoint[3];

typedef  struct      tGaussianMap {
         int         nopoint;
         tPoint      *v, center;
} tGaussianMap;

typedef enum {
         NULL, VERTEX, EDGE, POLYGON
} tVisibilityMapType;

typedef  struct      tNode {
         tPoint                  v;
         unsigned long           plus, minus;
         BOOL                    inserted, tag;
         tNode                   *next;
} tNode;

typedef  struct      tVisibilityMap {
         tVisibilityMapType      vmtype;
         int         nonode;
         tNode       *head;
} tVisibilityMap;
```

사용자 자료형인 **tGaussianMap**은 하나의 가우시안 맵을 나타내는데 **nopoint**는 점의 개수를, **v**는 입력된 점들의 좌표값을, **center**는 모든 점을 포함하는 가장 작은 원의 중심을 각각 나타낸다.

사용자 자료형인 **tVisibilityMap**는 하나의 가시맵을 나타내는데 **vmtype**은 가시맵의 형태를, **nonode**는 극점의 개수를, **head**는 반시계 방향으로 극점을 표현한 순환 리스트의 헤드를 각각 나타낸다. 리스트 노드의 자료형인 **tNode**에는 하나의 간선의 시작 끝점을 나타내는 **v**, 그 간선의 양/음 소유권 벡터인 **plus/minus**, 그리고 두 개의 부울변수인 **inserted/tag**, 그리고 노드의 링크인 **next**가 있다. **inserted** 변수는 간선을 분할하여 새로운 간선이 생기고 이 간선을 나타내기 위하여 그 노드가 삽입될 때 참이며 default는 거짓이다. **tag** 변수는 순환 리스트를 순회할 때 사용된다.

클래스의 주요 구조는 다음과 같다.

```
class     S2 {
private:
   int     NoGM;
   tGaussianMap                *GM[];
   tVisibilityMap              *VM[];

   int     GrahamScan(tGaussianMap *gm);
   void    ConstructPolygonVM(int i, int j);
   void    IntersectTwoVM(int i, int j);
   tNode   *AdvanceEdge(tNode **xl, tNode *x, int *xn);
   void    InsertBetween(tNode *xl, tNode *x, tPoint v);
   void    UpdateOwnership(int i, int j);
   void    SetOwnership(tVisibilityMap *vm, int owner);
   ....
public:
   void    GetGaussianMap(void);
   void    ConstructVisibilityMap(void);
   void    SplitEdges(void);
   int     GetCentroids(tGreatCircleType  gctype,  tPoint
*v);
};
```

클래스 **S2**는 구상에서 5가지 기하적 문제를 해결하기 위하여 간선 기반 분할 알고리즘을 구현한 것이다. **NoGM**은 가우시암맵의 개수를, 포인터 배열 **GM[]**과 **VM[]**은 각각 가우시안맵과 대응되는 양음가시맵을 표현한다. 전용(private) 부분은 주요 함수만 표현하였다. 구체적인 구현의 설명에 들어가기 앞서 클래스 **S2**를 사용하는 예를 먼저 살펴 전체적인 구조를 개략적으로 설명한다.

```
void main(int argc, char *argv[])
{
     S2          S;

     S.GetGaussianMap();
     S.ConstructVisibilityMap();
     S.SplitEdges();

     tPoint      *centroid;
     New(centroid, tPoint, MAX_NO_CENTROID);
     int nocentroid = S.GetCentroids
                  (MIN_INTERSECTION, centroid);
     for (int i = 0; i < nocentroid; i++) {
                  cout << "Solution " << i << ": ("
                  << centroid[i][0] << centroid[i][1]
                  << centroid[i][2] << ")\n";
     }
}
```

프로시져는 사용보다는 설명의 편의성을 위하여 처리 단계에 따라 4단계로 나누었다. 공용 함수 **GetGaussianMap**는 가우시안맵의 값을 포인터 배열 **GM[]**에 읽어들이는 부분이다. 실험에서는 불규칙적으로 값을 발생시켰지만 응용에 따라 표면의 수직벡터를 근사시켜 얻기도 하여야 한다. **ConstructVisibilityMap**는 입력된 가우시안맵으로부터 대응되는 양음가시맵을 구성하고 이를 순환 리스트로 표현하고 초기화하는 부분이다. **SplitEdges**는 모든 쌍의 가시맵을 교차시켜 분할되는 간선과 변경되는 소유권를 위하여 순환 리스트를 갱신하는 부분이다. **GetCentroids**는 해가 되는 간선을 수집하여 처리함으로써 모든 해 영역의 중심을 계산하여 그 개수를 반환하는 부분이다.

4.2 가시맵의 구성

어느 표면의 가우시안맵이 입력된 후에 대응되는 가시맵을 구하기 위해서는 가우시안맵의 볼록헐을 구성해야만 한다. 하나의 가우시안맵 $P=\{p1,...,pm\}$에 대하여 그 볼록헐을 $CH(P)$로 나타내고 이에 대응되는 양가시맵과 음가시맵을 각각 $VM(P)$과 VM^{-}로 나타낸다.

P를 포함하는 반구가 존재하지 않으면 $CH(P)$가 정의되지 않으므로, 즉 $VM(P)$가 존재하지 않으므로 $CH(P)$를 구성하기 전에 먼저 P의 반구성을 점검하여야 한다. 일반적인 경우라면 $CH(P)$가 구성된 후에 $VM(P)$는 간단하게 계산되지만 이 계산전에도 점과 간선만으로 degeneracy 경우를 점검하여 해결하여야 한다.

P를 포함하는 반구가 존재하면 $\bigcap_{i=1}^{m} HS(p_i) \neq \emptyset$이고 역도 성립한다[15]. 이러한 반구의 교차는 모든 $pi=(xi, yi, zi)$에 대하여 $x_i x+y_i y+z_i z \geq 0$의 부등식의 동질(homogeneous) 시스템으로 표현된다. 이 시스템에 선

형프로그래밍(LP)를 적용한다면 LP 알고리즘들[16,17,18, 19]이 항상 가능한 해 영역의 극점을 찾으므로 trivial 해인 (0,0,0)이 찾아진다.

P의 반구성을 검사하기 위하여 P의 모든 점을 $z=1$인 평면에 중심투상시켜 $z>0$인 점 집합과 $z<0$인 점 집합의 선형분리성 문제로 변형하기도 한다[15]. 그러나 이 방법은 기하적인 측면에서는 이론적으로 우아하지만 중심투상을 사용하므로 $z=0$에 가까운 점에 대한 수치적 처리 때문에 구현할 때 문제가 발생한다.

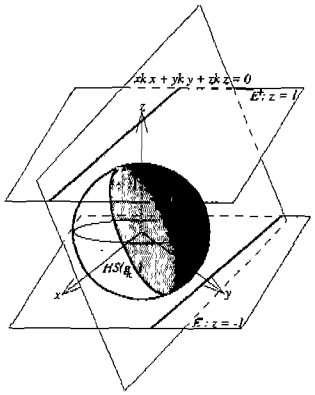

그림 3 두 평면에 대한 대원의 중심투상

우리는 중심투상의 개념을 이용하면서도 완전한 해를 견고한 계산으로 구할 수 있는 두 개의 LP 시스템을 만든다. 그림 3에서 보는 바와 같이 $HS(p_i)$의 경계인 대원을 E^+:$z=1$과 E^-:$z=-1$인 두 평면에 중심투상시키면 $z>0$인 대원의 부분은 E^+의 직선에 대응하고 다른 부분은 E 의 직선에 대응한다. 그리고 부등식 $x_ix+y_iy+z_iz\geq 0$은 두 평면에서 각각 $x_ix+y_iy+z_i\geq 0$와 $x_ix+y_iy-z_iz\geq 0$로 표현된다. 그러므로 안전한 수치 계산으로 두 개의 이질(heterogeneous) LP 시스템을 풀어 해를 결정할 수 있다.

반구성 검사가 끝난 후 P가 반구안에 포함되면서 볼록헐이 정의되지 않는 degeneracy는 경우는 두 가지밖에 없다. 하나는 $pi=-pi$와 같이 서로 부호만 다른 두 점이 있는 경우이고 다른 하나는 세 점이 하나의 대원 g상에 있으며 g의 어느 반원에도 포함되지 않을 때이다. 전자의 경우에는 가시맵이 하나의 간선이며 후자의 경우는 가시맵이 하나의 점뿐이다. 따라서 이러한 경우에는 P의 볼록헐을 구성하지 않고 직접 가시맵을 구성한다.

이러한 degeneracy를 찾기 위하여 P를 포함하는 최소구를 이용한다. (x^c, y^c, z^c)를 찾아진 최소구의 중심이라고 하자. $(x^c, y^c, z^c)\neq(0, 0, 0)$이면 최소포함구 s^c는 S^2보다 작으므로 P의 볼록헐을 구성할 수 있다. 그러나 $(x^c, y^c, z^c)\neq(0, 0, 0)$이면 s^c는 S^2와 일치하고 P의 볼록헐은 정의되지 않는다. P를 경계짓는 대원을 g^c라 하자. 그러면 degeneracy중 두 가지는 g^c상에 있는 P의 점들이 어느 반원에 포함되는지 아닌지를 검사하면 된다. 이 문제는 반구성 문제의 2차원 형태이다.

지금까지 설명한 가시맵 $VM(P)$를 구성하는 함수 **ConstructVisibilityMap**는 다음과 같다.

```
void    S2::ConstructVisibilityMap(void)
{
    for (int i = 0; i < NoGM; i++) {
        if (!IsHemispherical(GM[i])) {
            VM[i]->vmtype = NULL;
            continue;
        }
        SmallestSphere(GM[i]);
        if (InnerProduct(GM[i]->center, GM[i]->center)
< EPSILON) {
            // treat degeneracy
        } else {
            int noextreme = GrahamScan(GM[i]);
            if (noextreme < 3) {
                // treat degeneracy
            } else {
                ConstructPolygonVM(i, noextreme);
            }
        }
    }
}
```

위에서 호출되는 두 함수 **IsHemispherical**과 **SmallestSphere**는 각각 두 LP 시스템을 풀어 반구성을 결정하고 최소포함구를 결정하는 함수이다. LP에 대한 연구는 많은 결과가 보고되어 있으며 차원이 고정되었을 때 LP를 $O(n)$ 시간에 해결하는 알고리즘은 Megiddo[18]와 Dyer[16]에 의하여 처음 제시되었다. 그 후 Seidel[19]은 구현하기 쉬운 간단한 불규칙 알고리즘을 제안하였고 Hohmeyer[20]에 의하여 구현되었다. 최소 포함구를 구하는 $O(n)$ 시간의 알고리즘은 Megiddo[17]에 의하여 처음 제안되었다. 그 후 Welzl[21]은 구현하기 간편하고 실제적으로 매우 빠른 간단한 불규칙 알고리즘을 제안하였고 Erickson&Honda[22]에 의하여 구현되었다. 본 논문에서도 구현된 두 불규칙 알고리즘을 이용하였다.

함수 **GrahamScan**은 $CH(P)$를 구성하는 것으로 중심투상에서 발생하는 수치적 복잡성을 피하기 위하여

Graham[23]의 평면 알고리즘을 직접 구상에 적용시켜 구현한 것이다. 이 알고리즘은 극점을 먼저 찾고 나머지 점들을 그 극점을 기준으로 각도정렬하여 차례로 스캔하며 볼록헐을 구성하는 3 단계로 구성된다. 그 중에 전체적인 시간복잡도를 지배하며 주의해야할 부분은 각도정렬이다.

여기에서는 P를 포함하는 최소포함구의 중심 (x^c, y^c, z^c)를 중심으로 각도정렬하고 이 각도정렬을 견고하고 효율적으로 구현하기 위하여 정(orthogonal)투상을 이용한다. $PL(x^c, y^c, z^c)$을 점 (x^c, y^c, z^c)에서 S^2를 접하는 평면이라 하고 P의 모든 점을 이 평면에 정투상시켜 보자. 그러면 (x^c, y^c, z^c)를 중심으로 한 P의 순서가 평면 $PL(x^c, y^c, z^c)$에서도 변하지 않는다. 그런데 이러한 정투상은 중심투상과는 달리 수치값이 한정되므로 계산의 어려움이 없다.

P가 $PL(x^c, y^c, z^c)$에 투상된 결과의 좌표값을 구하기 위해서는 점 (x^c, y^c, z^c)를 z축의 점 (0, 0, 1)에 일치하도록 모두 회전시켜 x와 y 값을 그대로 취하면 된다. 따라서 평면상에 정투상된 점을 얻기 위하여 곱해지는 조합변환행렬은 다음과 같다.

$$R_x \cdot R_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{b}{d} & 0 \\ 0 & -\frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} d & 0 & a & 0 \\ -\frac{ab}{d} & \frac{c}{d} & b & 0 \\ -\frac{ac}{d} & -\frac{b}{d} & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $(a, b, c) = (x^c, y^c, z^c)$ and $d = \sqrt{b^2 + c^2}$.

평면에서 두 직선사이의 각을 직접 정렬하기 위해서는 많은 계산이 필요하다. 따라서 견고하고 효율적인 방법으로 두 점 (x_1, y_1), (x_2, y_2)를 지나는 직선의 기울기를 대신하여 수치 $\frac{dy}{|dx|+|dy|}$를 사용하는 방법이 많이 사용되고 있다. 여기에서 $dx = x_2 - x_1$이고 $dy = y_2 - y_1$이다. 그러므로 중심점 (a, b, c)에 대하여 점 $(x_i, y_i, z_i) \in P$의 각도 정렬에 이용되는 최종적인 값은 다음과 같이 계산된다.

$$\theta(x_i, y_i, z_i) = \frac{cy_i - bz_i}{|(b^2 + c^2)x_i - aby_i - acz_i| + |cy_i - bz_i|}$$

가우시안맵의 볼록헐을 구하기 위하여 구상에 적용된 **GrahamScan** 함수의 코드는 최소포함구의 중심으로부터 가장 먼점 하나를 구하여 극점 하나를 정하여 그 극점을 시발점으로 모든 점을 $\theta(x_i, y_i, z_i)$의 값을 이용하여 정렬한 후 극점 순서의 마지막 인덱스를 반환한다.

일반적인 $CH(P)$가 구성된 후 양음가시맵을 계산하여 $VM(P)/VM^-(P)$를 순환리스트인 포인터 배열 **VM[]**에 나타내는 것이다. $VM(P)$와 $VM^-(P)$는 이 포인터 배열에서 항상 **NoGM**의 인덱스 거리만큼 떨어져 있다. $VM(P)$의 각 간선의 시작 끝점을 리스트의 노드로 삽입하며 이 끝점은 대응되는 $CH(P)$의 간선의 양 끝점을 외적하여 얻어진다. 또한 각 노드안의 부울 변수들과 소유권 벡터들이 모두 초기화된다. 극점 **noextreme**개를 가지는 가우시안맵 **GM[i]**의 양가시맵 **VM[i]**와 음가시맵 **VM[NoGM+i]**를 계산하는 함수 **Construct PolygonVM**은 다음과 같다.

```
void    S2::ConstructPolygonVM(int i, int noextreme)
{
   int  di = i + NoGM;
   tNode                              *tnode[2];

   New(VM[i], tVisibilityMap, 1);
   New(VM[di], tVisibilityMap, 1);
   VM[i]->head = VM[di]->head = NULL;
   VM[i]->vmtype = VM[di]->vmtype = POLYGON;
   VM[i]->nonode = VM[di]->nonode = noextreme;

   for (int k = 0; k < noextreme; k++) {
      New(tnode[0], tNode, 1);
      New(tnode[1], tNode, 1);
      tnode[0]->tag = tnode[1]->tag = FALSE;
      tnode[0]->inserted = tnode[1]->inserted = FALSE;
      tnode[0]->plus = tnode[1]->minus = 1 << i;
      tnode[0]->minus = tnode[1]->plus = 0;
      CrossProduct(GM[i]->v[k], GM[i]->v[(k+1)%noex
      treme], tnode[0]->v);
      AddNode(VM[i]->head, tnode[0]);
      CrossProduct(GM[i]->v[(k+1)%noextreme],  GM[i]
      ->v[k], tnode[1]->v);
      AddNode(VM[di]->head, tnode[1]);
   }
}
```

4.3 구의 분할

간선 기반 분할의 가장 중요한 부분은 간선을 분할하여 그 소유권을 계산하고 자료구조를 갱신하는 단계이다. 가시맵을 나타내기 위하여 구성된 순환리스트는 처음에는 완전한 간선만을 포함하고 있고 모든 변수가 초기화되어 있다. 모든 쌍의 가시맵을 교차시킬 때마다 분할된 간선을 나타내기 위하여 새로운 노드가 삽입되고 간선의 소유권도 갱신된다.

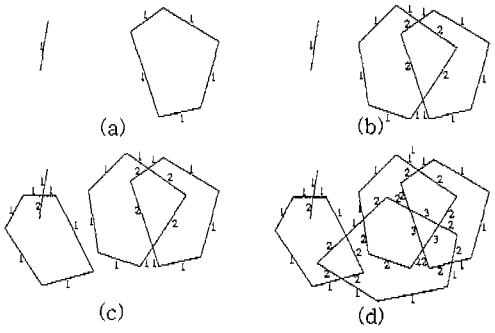

그림 4 증분적 교차에 의한 간선의 소유권 갱신

그림 4에 간선 기반 분할의 예를 도시하였다. 여기에서 증분적 교차의 순서는 결과와 무관하다. 다음에 표현된 함수 **SplitEdges**는 가시맵을 차례대로 교차시켜 간선을 분할하고 소유권을 갱신한다.

```
void S2::SplitEdges(void)
{
    for (int i = 0; i < NoGM*2 - 1; i++) {
        for (int j = i+1; j < NoGM*2; j++) {
            if ((i%NoGM) != (j%NoGM)) {
                IntersectTwoVM(i, j);
                UpdateOwnership(i, j);
            }
        }
    }
}
```

한 쌍의 볼록다각형 **VM[i]**와 **VM[j]**를 교차시키는 함수 **IntersectTwoVM**은 중요한 기초 연산이다. 평면에서는 주어진 두 다각형의 교차에 대해서는 많은 연구가 이루어져 두 볼록 다각형의 교차는 선형 시간에 계산할 수 있다[24,25,26]. 이 중에서 가장 우아하고 구현하기 간단한 것은 간선을 전진시켜나가면서 교차를 계산하는 O'Rourke 등의 알고리즘[24]이다. 수치적으로 안전한 계산을 위하여는 구상의 간선을 평면에 중심투상시키지 않고 구상에서 직접 간선을 전진시켜야 한다 그러나 평면의 알고리즘은 간선의 추적(chasing) 관계와 상대적 위치를 기초로 하고 있으며 구상에서는 이러한 관계와 위치가 모호하여 정의되지 않는다.

여기에서는 구상의 간선의 전진을 위하여 제시된 새로운 알고리즘[27]을 사용하여 구현한다. 이 알고리즘은 두 간선간의 직면(facing) 관계와 상대적 위치를 기초로 하고 있다. 다각형 $P=\{p_1,...,p_m\}$와 $Q=\{q_1,...,p_n\}$의 정점은 반시계 방향으로 되어있다고 가정하고 간선 $\overline{p_{i-1}p_i}$는 간단히 $\overline{p_i}$로 나타낸다. 또한 간선 $\overline{p_i}$를 지나는 대원으로 나뉘어지는 왼쪽 및 오른쪽 반구를 각각 $LHS(\overline{p_i})$와 $RHS(\overline{p_i})$로 나타낸다. 이러한 표현을 사용하여 다음과 같이 두 간선간의 직면 관계와 두 다각형간의 맞물림 상태를 먼저 정의한다.

정의 1.

$Face(\overline{q_j}, \overline{p_i})$:
$p_{i-1} \in LHS(\overline{q_j}) \wedge p_i \in RHS(\overline{q_j}) \wedge q_j \in RHS(\overline{p_i})$
$\vee p_{i-1} \in RHS(\overline{q_j}) \wedge p_i \in LHS(\overline{q_j})) \wedge q_j \in LHS(\overline{p_i})$

정의 2. 교차하는 두 다각형 P와 Q상에서 진행중인 각각의 간선 $\overline{p}$와 $\overline{q}$가 있을 때 $\overline{q} \in RHS(\overline{p}) \vee Face(\overline{p}, \overline{q}) \vee \overline{p} \in RHS(\overline{q}) \vee Face(\overline{q}, \overline{p})$라면 **맞물림 상태**(**geared state**)에 있다고 한다.

위 두 정의를 사용하여 구상에서 간선의 전진을 위한 직면 규칙을 다음과 같이 표현할 수 있다. 자세한 내용은 문헌 [27]을 참고하면 된다.

표 1 S^2상에서 직면 관계에 의한 간선의 전진 규칙

	$Face(\overline{p}, \overline{q})$	$\neg Face(\overline{p}, \overline{q})$
$Face(\overline{q}, \overline{p})$	(규칙 0) advance the one that has not recently advanced, if geared advance $\overline{p}$, if non-geared	(규칙 1) advance $\overline{q}$
$\neg Face(\overline{q}, \overline{p})$	(규칙 2) advance $\overline{p}$	(규칙 3) (1) advance $\overline{p}$, if $p \in RHS(\overline{q})$, (2) advance $\overline{q}$, else if $q \in RHS(\overline{p})$ (3) advance $\overline{p}$, otherwise

구상에서 직면 관계에 의한 간선의 전진 규칙을 적용하여 한 쌍의 가시맵을 교차시킴으로써 순환리스트를 갱신하는 **IntersectTwoVM**는 다음과 같다.

```
void    S2::IntersectTwoVM(int i, int j)
{
   tNode *a1, *b1, *a, *b;
   int    a1HB, aHB, b1HA, bHA;
   int    an = 0, bn = 0;
   BOOL firstpoint = TRUE,
        aturn = FALSE;

   if ((VM[i]->vmtype == VERTEX || VM[i]->vmtype == EDGE
        || VM[j]->vmtype == VERTEX || VM[j]->vmtype == EDGE) {
        // treat degeneracy
   }
   a1 = VM[i]->head; a = a1->next; while (a->inserted) a = a->next;
   b1 = VM[j]->head; b = b1->next; while (b->inserted) b = b->next;
   while ((aHB = CCW(b1->v, b->v, a->v)) < 0
        && (bHA = CCW(a1->v, a->v, b->v)) < 0) {
        a = AdvanceEdge(&a1, a, &an);
   }
   do {
        a1HB = CCW(b1->v, b->v, a1->v);
        b1HA = CCW(a1->v, a->v, b1->v);
        aHB = CCW(b1->v, b->v, a->v);
        bHA = CCW(a1->v, a->v, b->v),

        tPoint    apole;
        CrossProduct(a1->v, a->v, apole);
        BOOL parallel = (InnerProduct(apole, b1->v)
                  == InnerProduct(apole, b->v));
        if ((aHB * a1HB * bHA * b1HA == 0) || parallel) {
             // treat degeneracy
        }
        if ((aHB * a1HB < 0) && (bHA * b1HA < 0) && (aHB != bHA)) {
             tPoint    p;
             IntersectTwoEdge(a1, a, b1, b, p);
             InsertBetween(a1, a, p);
             InsertBetween(b1, b, p);
             if (firstpoint ) {
                  an = bn = 0;
                  firstpoint = FALSE;
             }
        }
        if ((aHB * a1HB < 0) && (bHA * b1HA < 0) && (aHB == bHA))
{
             if (aturn) {
                  a = AdvanceEdge(&a1, a, &an); aturn = FALSE;
             } else {
                  b = AdvanceEdge(&b1, b, &bn); aturn = TRUE;
             }
        } else if (aHB*a1HB < 0 && aHB == bHA) {
             b = AdvanceEdge(&b1, b, &bn); aturn = TRUE;
        } else if (bHA*b1HA < 0 && aHB == bHA) {
             a = AdvanceEdge(&a1, a, &an); aturn = FALSE;
        } else if (aHB < 0) {
             a = AdvanceEdge(&a1, a, &an); aturn = FALSE;
        } else {
             b = AdvanceEdge(&b1, b, &bn); aturn = TRUE;
        }
   } while (((an < VM[i]->nonode) || (bn < VM[j]->nonode))
        && (an < 2*VM[i]->nonode) && (bn < 2*VM[j]->nonode));
}
```

순환리스트는 가시맵의 극점뿐만이 아니라 교차점을 노드로 포함하고 있다. 따라서 다음에 표현된 전진 함수 **AdvanceEdge**는 부울 변수 **inserted**를 검사하여 간선의 전진시 이 교차점들을 건너뛴다. 또한 호출될 때마다 전진 계수를 증가시킨다.

```
tNode    *S2::AdvanceEdge(tNode **x1, tNode *x, int *xn)
{
   *x1 = x; x = x->next; while (x->inserted) x = x->next;
   (*xn)++;
   return(x);
}
```

다음 함수 **InsertBetween**는 생성된 교차점을 위한 새로운 노드를 삽입한다. 이 연산은 간선이 분할됨을 의미한다. 따라서 삽입되는 새로운 노드는 분할되기 전의 간선의 소유권과 링크를 계승받는다.

```
void S2::InsertBetween(tNode *x1, tNode *x, tPoint v)
{
     tNode     *tnode;

     while (!IsBetween(x1->v, x1->next->v, v) && x1->next != x) {
          x1 = x1->next;
     }
     New(tnode, tNode, 1);
     *tnode = *x1;
     tnode->v = v;
     tnode->tag = tnode->inserted = TRUE;
     x1->next = tnode;
}
```

두 가시맵 **VM[i]**와 **VM[j]**이 서로 교차되는 부분에 속하는 간선들의 노드를 갱신하는 함수 **UpdateOwnership**에서는 먼저 **VM[i]**의 해당 간선들의 소유권 벡터 **j** 번째를 세트한 후 대칭적으로 **VM[j]**의 해당 간선들의 소유권 벡터 **i** 번째를 세트한다.

```
void S2::UpdateOwnership(int i, int j)
{
     SetOwnership(VM[i], j);
     SetOwnership(VM[j], i);
}
```

두 다각형간의 교차점은 항상 짝수 개이므로 순환리스트를 순회하면서 교차점이 찾아질 때마다 교대로 소유권벡터를 세트시키거나 하지 않는다. 새로 삽입된 교

차점은 부울 변수 **tag**가 세트되어 있으므로 이를 체크함으로써 찾아내고 다시 리세트시킨다. 구현된 함수 **SetOwnership**은 다음과 같다.

```
void S2::SetOwnership(tVisibilityMap *vm, int owner)
{
    BOOL    invm;
    tNode   *tnode;

    // investigate the starting point
    while (vm->head->tag) {
        vm->head = vm->head->next;
    }
    invm = IsInsideVM(vm->head->v, VM[owner]);
    // traverse vm
    tnode = vm->head;
    do {
        if (tnode->tag) {
            tnode->tag = FALSE;
            invm = !invm;
        }
        if (invm) {
            if (owner < NoGM) {
                tnode->plus |= 1 << owner;
            } else {
                tnode->minus |= 1 << (owner - NoGM);
            }
        }
        tnode = tnode->next;
    } while (tnode != vm->head);
}
```

4.4 최소 및 최대 교차의 계산

소유권의 최소 및 최대 교차수는 최종적으로 얻어진 순환리스트를 순회하여 소유권 벡터를 보고 결정하고 재순회하여 그 값을 가지는 간선들을 비정렬된 순서로 수집한다. 이 수집된 간선들을 정렬하면 모든 해의 경계를 구성할 수 있으나 최대 교차의 경우에는 해의 영역이 항상 볼록하고 홀이 없다는 특성이 있다. 또한 하나의 경계에 있는 간선은 같은 소유권을 가지므로 더 효율적인 방법을 생각할 수 있다. 앞에서 언급했듯이 실제적으로는 최대 교차 영역의 중심 해가 대부분 사용된다.

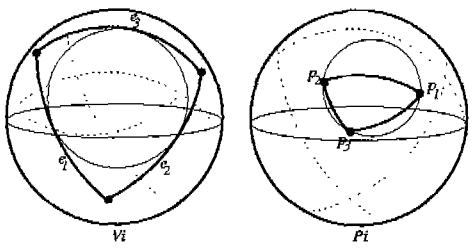

그림 5 이원 다각형의 내접원과 외접원

따라서 최대 교차 영역의 기하적 특성을 이용하여 경계를 구성하지 않고 근사적인 중심을 구한다.

볼록 다각형을 내접하는 원의 중점은 그 다각형의 중심에 좋은 근사치가 된다. 그러나 이 원이 접하는 간선이 될 수 있는 후보는 조합적(combinatorial)이기 때문에 그 원의 중심을 정하기에 복잡하다. 그런데 그림 5에서 보는 바와 같이 그 다각형의 내접원의 중심은 이원 다각형의 외접원의 중심과 같다는 성질[14]을 이용하면 효율적으로 계산할 수 있다. 이 외접원은 최소포함구를 구는 것과 동일하다.

이 방법은 수치적 한계로 발생하는 교차점들의 위상 위배에도 이 방법이 견고하다. 왜냐하면 위상 위배가 생긴 경우에 최대 소유권을 가지는 간선을 모으면 하나의 면이 구성되지는 않으나 위의 내접원을 구하는 방식의 계산은 이산적인 간선을 처리하므로 같은 방법으로 계산하면 된다.

아래의 함수 **GetCentroids**는 최대 교차 영역들의 중심 해를 계산하여 그 개수를 반환한다. 여기에서 최소 교차를 찾는 부분은 구현하지 않았다. 매개변수 **gctype**은 구상의 5가지 기하적 문제의 형태를 나타내고 **v**는 해로 나온 점의 좌표를 나타낼 곳이다. 이 프로시져에서는 우선 최대 교차수를 결정하고 그 교차수를 가지는 간선들을 저장소에 보관한다. 그런 후에 그 간선들을 같은 소유권을 가지는 그룹으로 나누어 그들의 중심을 구한다.

```
int   S2::GetCentroids(tGreatCircleType gctype, tPoint *v)
{
    int   max, tmp, nocentroid = 0;
    tNode     *tnode;

    if (gctype == MAX_INTERSECTION) {
        // find the minimum intersection of visibility maps
        return(nocentroid);
    }
    // determine the maximum number
    // by traversing VM[i]->head for all i=1...(NoGM*2-1)
    max = 0;
    for (int i = 0; i < NoGM*2; i++) {
        tnode = VM[i]->head;
        do {
            tmp = NoOfBits(tnode->plus);
            if (gctype != DENSEST_COVER) {
                tmp += NoOfBits(tnode->minus);
            }
            max = (tmp > max) ? tmp : max;
            tnode = tnode->next;
        } while (tnode != VM[i]->head);
    }
    if ((gctype == SEPARATOR || gctype == BISECTOR) && (max
!= NoGM)) {
```

```
        return(nocentroid);
    }
    // gather edges with the maximum number into a list pool
    // by traversing VM[i]->head for all i=1...(NoGM*2-1)
    ...omitted...
    // get the centroids of solutions by grouping the edges into a
       bucket
    tGaussianMap   *bucket;
    New(bucket, tGaussianMap, 1);
    New(bucket->v, tPoint, pool->nonode);
    while (pool->head) {
        unsigned long   tp = 0, tm = 0;
        bucket->nopoint = 0;
        tnode = pool->head;
        do {
            tNode*snode = tnode->next;
            if ((!tp  && !tm )
                || (tp == tnode->plus) && (tm == tnode->minus))
{
                tp = tnode->plus; tm = tnode->minus;
                bucket->v[bucket->nopoint--] = tnode->v;
                // delete the processed node
                if (snode == snode->next) snode = NULL;
                if (tnode == pool->head) pool->head = snode;
                Delete(tnode);
            }
            tnode = snode;
        } while (tnode != pool->head);

        if ((gctype == BISECTOR) &&
          (abs(NoOfBits(tp)-NoOfBits(tm)) > 1)) {
            continue;
        }
        SmallestSphere(bucket);
        v[nocentroid++] = bucket->center;
    }
    return(nocentroid);
}
```

5. 시간복잡도 분석과 실험

5.1 시간복잡도

$VM(P_i)$의 정점의 개수를 v_i라 하고 $v=\Sigma_{i=1}^{n} v_i$라 하자. 그러면 위 알고리즘에서 한 쌍의 구 볼록 다각형을 교차시키는 선형시간 알고리즘[26]을 이용하므로 모든 교차점을 찾아내기 위하여 $\Sigma_{i=1}^{n-1}\Sigma_{j=i+1}^{n} O(v_i+v_j)=O(nv)$ 시간이 소요된다. 최대 교차수를 결정하고 그 수의 소유권을 가지는 간선을 수집하기 위하여 각각 순환 리스트를 한번씩 순회하며 이것은 모든 점의 개수에 선형적이다. 수집된 간선에 이웃인 점을 포함하는 최소포함구도 역시 간선의 개수에 선형적으로 수행되므로 다음과 같은 결과를 얻을 수 있다.

정리 1. 구 볼록 다각형의 각 최대 교차 영역의 근사적인 중심 해는 $O(nv)$ 시간에 찾아진다. 여기서 n과 v는 각각 다각형과 모든 정점의 개수이다.

해가 되는 모든 면의 경계를 구성하기 위해서는 수집된 간선을 정렬하여야 한다. 따라서 최대 교차수뿐만이 아니라 일반적인 교차수를 가지는 모든 영역을 구성하기 위한 결과는 다음과 같다.

정리 2. 구 볼록 다각형의 모든 k-교차 영역의 경계는 $O(nv+L\log L)$ 시간에 구성할 수 있다. 여기서 L은 해로 나오는 간선의 개수이다.

5.2 적용된 예

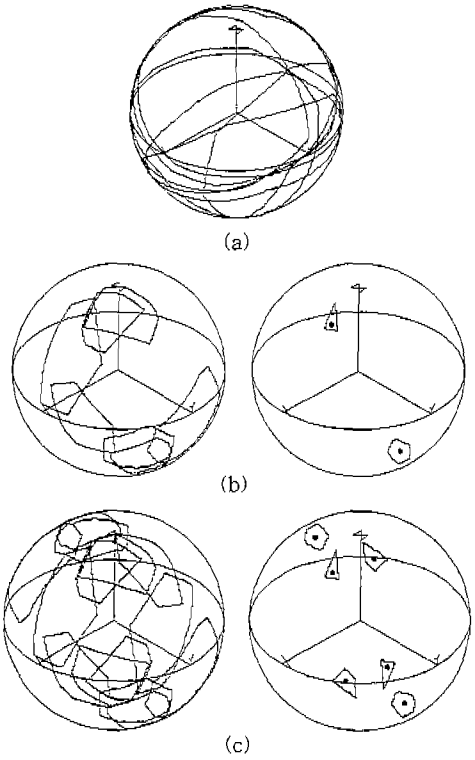

그림 6 최대 교차 영역 해의 중심

그림 6는 구현된 코드에 의하여 여러 가시맵의 최대 교차 영역을 구한 결과를 출력한 예이다. (a)는 주어진 가우시안맵의 볼록헐들을 나타내고 (b)는 (a)의 양가시맵에서의 최대 교차면과 그 중심해를 (c)는 양가시맵 및 음가시맵에서의 최대 교차면과 중심 해를 각각 구한 결과이다.

표 2 볼록다각형의 개수와 모든 정점의 개수에 따른 수행 시간의 변화

항목	1	2	3	4	5	6	7	8	9	10	11	12	13
볼록다각형개수(n)	5	8	10	15	15	20	20	23	25	25	27	29	29
모든정점의개수(v)	45	71	90	103	149	163	187	179	203	233	228	234	272
n*v	225	568	900	1545	2235	3260	3740	4117	5075	5825	6156	6786	7888
수행시간(1/1000초)	24	40	68	119	156	255	262	314	388	410	465	520	589
수행시간*100/(n*v)	10.6	7.04	7.58	7.7	7	7.82	7.01	7.64	7.66	7.04	7.55	7.67	7.46

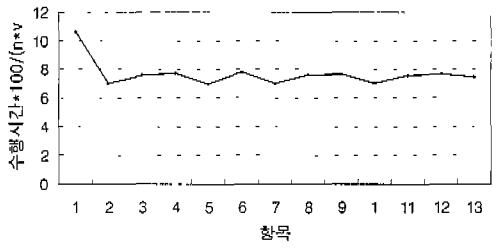

표 2는 불규칙적으로 생성된 자료로부터 사전에 구성된 가시맵 볼록다각형의 개수 n과 모든 정점의 개수 v가 변함에 따라 양가시맵의 최대 교차를 계산하는 시간이 변하는 것을 여러번 측정한 결과이다. 수행시간을 nv로 나눈 값이 거의 일정하므로 실제 구현된 코드에서도 수행 시간이 이론적으로 분석된 시간복잡도 $O(nv)$에 근사함을 확인시켜주고 있다. 프로그램이 개발된 환경은 64 메가 메모리의 펜티움 컴퓨터이다.

6. 결 론

본 논문에서는 구 볼록 다각형의 교차를 구하기 위하여 구를 간선에 기반하여 분할하는 견고한 알고리즘을 제시하였다. 모든 해의 경계는 $O(vn+L\log L)$ 시간에 구성할 수 있다. 여기서 n과 v는 각각 다각형과 모든 정점의 개수이고 L은 해로 나오는 간선의 개수이다.

$L\cong n$이라면 $O(vn\log n)$ 시간이 소요되는 면 기반 알고리즘에 비하여 우수하다. 또한 $L\gg n$이더라도 실제 구현시에는 이산적인 간선을 병합하는 시간 $O(L\log L)$은 경계 표현과 복잡한 자료구조를 사용하는 추가 비용보다 미미할 것으로 여겨지며 특히 우리의 알고리즘은 교차점의 위상 위배를 극복할 수 있고 점 또는 선분 가시맵과 같은 degeneracy를 처리할 수 있으며 수치적 계산이 견고하다.

실제 CAD/CAM 응용에서 선호되는 최대 교차 영역의 중심 해는 기하적 특성을 이용하여 효율적인 시간 $O(nv)$에 구할 수 있었다. 또한 C++ 언어의 클래스를 사용하여 구체적으로 구현하는 방법을 설명하였고 실험함으로써 이론적인 시간복잡도가 실제에도 적용됨을 증명하였으므로 가시맵과 관련된 응용 소프트웨어 개발시 기초 코드로 유용하게 사용될 수 있으리라 기대한다.

참 고 문 헌

[1] L. L. Chen, S. Y. Chou, and T. C. Woo, "Separating and intersecting spherical polygons: computing machinability on three- four- and five-axis numerically controlled machines," *ACM Tr. on Graphics*, Vol. 12, No. 14, pp.305-326, 1993.

[2] S. H. Suh and J. K. Kang, "Process planning for multi-axis NC machining of free surfaces," *Int. J. Prod. Res.* Vol. 33, No. 10, pp.2723-2738, 1995.

[3] J. W. H. Tangelder, "Automated fabrication of shape models of free-form objects with a sculpturing robot," Ph. D. Thesis, Industrial Design Engineering of Delft Univ. of Technology, 1998.

[4] P. Gupta *et al.*, "Efficient geometric algorithms for workpiece orientation in 4- and 5-axis NC machining," *Computer-Aided Design*, Vol. 28, No. 8, pp.577-587, 1996.

[5] K. Tang, T. Woo, and J. Gan, "Maximum intersection of spherical polygons and workpiece orientation for 4- and 5-axis machining," *ASME J. Mechanical Design*, Vol. 114, pp. 477-485, 1992.

[6] B. G. Baumgart, "A polyhedron representation for computer vision," In National Computer Conference, Anaheim, CA, *IFIPS*, pp.589-596, 1975.

[7] K. Weiler, "Edge-based data structures for solid modeling in curved-surface environments," *IEEE Computer Graphics and Application*, Vol. 5, No. 1, pp.21-40, 1985.

[8] T. C. Woo, "A combinatorial analysis of boundary data structure schema," *IEEE Comput. Graph. Appl.*, Vol. 5, No. 3, pp.19-27, 1985.

[9] S. M. Barker, "Towards a topology for computational geometry," *Computer-Aided Design*, Vol.

27, No. 4, pp.311-318, 1995.
[10] S. Fortune, "Efficient exact arithmetic for computational geometry," *Proc. 9th ACM Symp. Comp. Geom.*, pp.163-172, 1993.
[11] C. M. Hoffmann, "The problems of accuracy and robustness in geometric computation," *Computer*, Vol. 22, No. 3, pp.31-41, 1989.
[12] V. Milenkovic, "Verifiable implementations of geometric algorithms using finite precision arithmetic," *Artificial Intelligence*, Vol. 37, pp.377-401, 1988.
[13] E. Horowitz and S. Sahni, Fundamentals of Data Structure, Computer Science Press, 1973.
[14] J. G. Gan, T. C. Woo and K. Tang, "Spherical maps: their construction, properties, and approximation," *ASME J. Mechanical Design*, Vol. 116, pp.357-363, 1994.
[15] L. L. Chen and T. C. Woo, "Computational geometry on the sphere with application to automated machining," *Tr. ASME*, Vol. 114, pp. 288-295, 1992.
[16] M. E. Dyer, "Linear time algorithms for two- and three-variable linear programs," SIAM *J. Computing*, Vol. 13, No. 1, pp.31-45, 1984.
[17] N. Megiddo, "Linear-time algorithms for linear programming in and related problems," In *Proc. 23rd Annual IEEE Sym. Found. Comput. Sci.*, pp 329-338, 1982.
[18] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *J. ACM*, Vol. 31, No. 1, pp.114-127, 1984.
[19] R. Seidel, "Small-dimensional linear programming and convex hulls made easy," *Discrete and Computational Geometry*, Vol. 6, pp.423-434, 1991.
[20] M. E. Hohmeyer, *Implementation code of linear programming.*
[21] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," New Results and New Trends in Computer Science, Springer Lecture Notes in Computer Science 555, pp.359-370, 1991.
[22] J. Erickson and H. Honda, *Implementation code of the smallest enclosing sphere.*
[23] R. L. Graham, "An efficient algorithm for determining the convex hull of finite planar set," *Information Processing Letters*, Vol. 1, pp.132-133, 1972.
[24] J. O'Rourke, C. B. Chien, T. Olson, and D. Naddor, "A new linear algorithm for intersecting convex polygons," *Computer Graphics and Image Processing*, Vol. 19, pp.384-391, 1982.
[25] M. I. Shamos and D. Hoey, "Geometric intersection problems," *Seventeenth Annual IEEE Symposium on Foundations of Computer Science*, pp.208-215, 1976.
[26] G. T. Toussaint, "A simple linear algorithm for intersecting convex polygons," *The Visual Computer*, Vol. 1, pp.118-123, 1985.
[27] 하종성, "구상의 볼록 다각형의 교차 계산을 위한 선형 시간 알고리즘", *한국정보과학회지논문지: 시스템및 이론*, Vol. 28, No. 2, pp.58-63, 2001.

하 종 성
정보과학회논문지 : 시스템 및 이론
제 28 권 제 2 호 참조

천 은 홍
1981년 광운대학교 응용전자공학과 졸업(공학사). 1985년 아주대학교 대학원 전자공학과 졸업(공학석사). 1998년 아주대학교 대학원 컴퓨터공학과 졸업(공학박사). 1985년 ~ 1988년 삼성종합기술원 정보시스템연구소 주임연구원. 1998년 ~ 2000년 우석대학교 전자계산소 소장. 1988년 ~ 현재 우석대학교 정보통신컴퓨터공학부 부교수. 관심분야는 정보보안, 컴퓨터네트워크 등