사각형 윈도우에 대한 효율적인 선분 절단 알고리즘

김 응 곤† 허 영 남†† 이 웅 기†††

요 약

사각형 윈도우에 대하여 2차원 선분 절단을 효율적으로 수행하는 알고리즘을 제안한다. 본 알고리즘은 선분 절단에 필요한 산술 및 논리 연산의 수를 줄임으로써 수 많은 선분으로 구성되는 화상을 그리는데 적합하며, 다른 알고리즘과 비교하여 그 효율성을 나타내었다.

An Efficient Line Clipping Algorithm on a Rectangular Window

Eung Kon Kim† Young Nam Heo†† and Woong Ki Lee†††

ABSTRSCT

An efficient algorithm for clipping 2D lines on a rectangular window is proposed.It is suitable for displaying images consisted of many lines for it can reduce the number of arithmetic and logical operations. The algorithm is compared with the Cohen-Sutherland algorithm and it was proved to be efficient.

1. 서 론

사각형 윈도우에 대한 선분 절단은 컴퓨터 그래픽스에서 가장 중요한 알고리즘 중의 하나이다. 선분 절단 알고리즘은 어느 선분이 완전히 윈도우 경계 내부에 있는지 또는 완전히 윈도우 경계 외부에 있는지 또는 부분적으로 절단되는지를 결정한 후 윈도우 내부의 부분만을 그리도록 하는 것이다. 하나의 화상은 수 천 개의 선분을 포함할 수도 있으므로 절단 과정은 가능한 한 효율적으로 이루어져야 한다[1, 2, 3].

가장 잘 알려진 선분 절단 알고리즘은 Cohen과 Sutherland에 의해 개발된 방법[1, 2, 3, 4, 5, 6]으로, 선분을 구성하는 양 끝점마다 그 점의 좌표 영역을 나타내는 4자리의 2진 코드로 된 영역 코드(region code)를 부여하여 그 선분이 윈도우의 내부 또는 외부에 있는지를 검사하게 된다. 윈도우 경계와 선분과의 교차점을 계산하기 전에 먼저 모든 선분들에 대하여 완전히 윈도우 경계 내부에 있는지, 또는 완전히 외부에 있는지를 구분해야 한다. 이러한 선분들을 제외시키고 나서 남은 선분들의 교차점 계산을 수행한다.

이 방법은, 모든 입력 선분마다 코딩을 하여 선분이 완전히 윈도우 내부 또는 외부에 있는지 검사를 거친 후, 연속적으로 윈도우와의 교차점을 계산해야 한다. 교차점 계산으로 선분의 한 끝을 절단하여 절단된 선분에 대하여 다시 앞의 과정을 반복한다[1, 2, 3]. 따라서 그려야 할 선분의 수가 많아지면 연산의 수가 많아지므로 비효율적이다.

본 논문에서는 수 많은 선분으로 구성된 화상을 효율적으로 그리기 위하여 이 알고리즘과는 다른 계산 모델로서 반복문을 이용하지 않고 절단에 필요한 산술 및 논리 연산의 수를 최대로 줄이는 선분절단 알고리즘을 제안한다. 제안한 알고리즘의 소요 연산 수와 수행 시간을 Cohen-Sutherland 알고리즘[1, 2, 3]과 비교함으로써 본 알고리즘의 효율성을 보여준다.

다음 장에서는 본 논문의 선분 절단 알고리즘을 기술하고, 제 3장에서는 다른 알고리즘과의 비교와 실증 결과에 대하여 기술하고, 제 4장에

† 정 회 원 : 순천대학교 전자계산학과 조교수
†† 정 회 원 : 순천대학교 전산계산학과 교수
††† 정 회 원 : 조선대학교 전산통계학과 부교수
논문접수 : 1994년 6월 8일, 심사완료 : 1995년 3월 3일

서는 결론을 맺는다.

2. 선분절단 알고리즘

가장 많이 사용되고 있는 Cohen-Sutherland 알고리즘은 선분의 양끝점을 4자리의 2진 코드로 코딩을 하여 선분이 완전히 윈도우 내부 또는 외부에 있는지 검사를 거친 후, 윈도우 경계와의 교차점을 계산한다. 교차점 계산으로 선분의 한 끝을 절단한 후, 절단된 선분에 대하여 다시 코딩을 하여 경계와의 교차여부를 검사하여 교차점을 계산하는 과정을 절단된 선분이 윈도우 내부 또는 외부에 있을 때까지 반복한다. 따라서 연산의 수가 많아지므로 비효율적이다.

본 논문에서는 선분의 기울기가 양수인 경우와 0이하인 경우로 나누어 선분의 왼쪽 끝점이 윈도우의 왼쪽 경계 밖에 있는지, 오른쪽 끝점이 윈도우의 오른쪽 경계 밖에 있는지, 왼쪽 끝점이 윈도우의 아래쪽 경계 밖에 있는지, 오른쪽 끝점이 윈도우 경계 밖에 있는지를 검사하여 각각의 경우에 반복문을 사용하지 않고 절단에 필요한 연산의 수를 최대로 줄이는 선분절단 알고리즘을 제안한다.

제안 알고리즘은 다음과 같이 선분의 양끝점에 대하여 코딩을 하지 않으며, 반복문을 사용하지 않고 교차점 계산을 최소화하여 처리한다.

선분 양 끝점의 좌표가 (x_1,y_1)와 (x_2,y_2)이고, 윈도우를 구성하는 두 점의 좌표를 (xw_min,yw_min)와 (xw_max,yw_max)라고 하자. 선분 양 끝의 좌표 중 x1의 값이 x2의 값보다 큰 경우에는 두 점의 좌표를 교환한다. 선분의 기울기에 따른 두 가지 경우에 각각 선분을 구성하는 좌표 중 왼쪽 끝점의 x좌표인 x_1의 값이 윈도우의 왼쪽 경계의 x좌표값(xw_min)보다 작은 경우, 선분의 오른쪽 끝점의 x좌표인 x2의 값이 윈도우의 오른쪽 경계의 x좌표값(xw_max)보다 큰 경우, 왼쪽 끝점의 y좌표인 y_1의 값이 윈도우의 아래쪽 경계의 y좌표값(yw_min)보다 작은 경우, 그리고 오른쪽 끝점의 y좌표인 y_2좌표값이 윈도우 경계의 윗쪽 경계의 y좌표값(yw_max)보다 큰 경우로 나누어 각각의 경우에 따른 좌표값을 윈도우 경계의 좌표값과 비교함으로써 윈도우 밖에 있는 선분은 먼저 식별이 되며, 비교 연산에 의해서도 식별이 되지 않는 선분은 계속하여 윈도우 경계와 1회 또는 2회의 절단 연산 후 윈도우 밖에 있음이 식별된다. 또한 적어도 한 끝점이 윈도우 밖에 있는 선분은 계속하여 윈도우 경계와 1회 또는 2회의 절단 연산이 필요하며, 최후로 완전히 윈도우 내부에 있는 선분이나 절단된 선분만이 남게 된다. 프로시져 line_clip은 본 알고리즘을 의사 코드로 표현한 것이다. x_clip은 주어진 점(x_1,y_1)또는(x_2,y_2)를 다음과 같이 [1] 선분과 윈도우의 수직 경계와의 교차점으로 치환한다.

$x_1 \leftarrow xw,$
$y_1 \leftarrow y_1 + m^*(xw-x_1),$
$x_2 \leftarrow xw,$
$y_2 \leftarrow y_2 + m^*(xw-x_2)$

여기서 m은 선분의 기울기이고, xw는 윈도우의 왼쪽 경계인 경우 xw_min이 되고, 윈도우의 오른쪽 경계인 경우 xw_max가 된다. 또한 y_clip은 주어진 점(x_1, y_1)또는(x_2, y_2)를 다음과 같이[1] 선분과 윈도우의 수평 경계와의 교차점으로 치환한다.

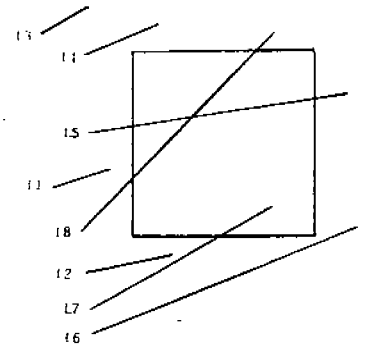
$x_1 \leftarrow x_1+(yw-y_1)/m,$
$y_1 \leftarrow yw,$
$x_2 \leftarrow x_2+(yw-y_2)/m,$
$y_2 \leftarrow yw$

여기서 yw는 윈도우의 아랫쪽 경계인 경우 yw_min이 되고, 윈도우의 윗쪽 경계인 경우 yw_max가 된다.

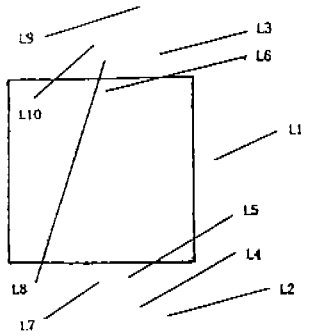
```
procedure line_clip(x1,y1,x2,y2)
//(x1,y1),(x2,y2)는 각각 선분의 왼쪽과 오른쪽
  끝점의 좌표 //
integer xw_min,xw_max,yw_min,yw_max //
  윈도우 경계 좌표 //
real m // 선분의 기울기 //
if y1<y2 then
    if x1 < xw_min then
        if x2<xw_min then return endif
            //(그림 1)의 선분 L1//
```

```
    if y2<yw_min then return endif
        //A(그림 1)의 선분 L2//
    if y1>yw_max then return endif
        // (그림 1)의 선분 L3 //
    m←(y2−y1)/(x2−x1)
    callx_clip(x1,y1,xw_min)
    if  y1>yw_max  then  return
        endif
        // (그림 1)의 선분 L4 //
    if x2>xw_max then
        // (그림 1)의 선분 L5 //
    call x_clip(x2,y2,xw_max)
    endif
    if y2<yw_min then return endif
        // (그림 1)의 선분 L6 //
    if y1<yw_min then
        // (그림 1)의 선분 L7 //
    call y_clip(x1,y1,yw_min)
    endif
    if y2>yw_max then
        // (그림 1)의 선분 L8 //
    call y_clip(x2,y2,yw_max)
    endif
else if x2>xw_max then
    if x1>xw_max then return endif
        // (그림 2)의 선분 L1 //
    if y2<yw_min then return endif
        // (그림 2)의 선분 L2 //
    if y1>yw_max then return endif
        // (그림 2)의 선분 L3 //
    m←(y2−y1)/(x2−x1)
    call x_clip(x2,y2,xw_max)
    if y2<yw_min then return endif
        // (그림 2)의 선분 L4 //
    if y1<yw_min then
        // (그림 2)의 선분 L5 //
    call y_clip(x1,y1,yw_min)
    endif
    if y2>yw_max then
        //(그림 2)의 선분 L6 //
          call y_clip(x2,y2,yw_max)
    endif
    endif
    else if y1<yw_min then
        if y2<yw_min then return endif
            // (그림 2)의 선분 L7 //
        m←(y2−y1)/(x2−x1)
        call y_clip(x1,y1,yw_min)
        if y2 > yw_max then
            // (그림 2)의 선분 L8 //
              call y_clip(x2,y2,yw_max)
        endif
      endif
    else if y2>yw_max then
       if y1>yw_max then return endif
            //(그림 2)의 선분 L9 //
        m←(y2−y1)/(x2−x1)
        call y_clip(x2,y2,yw_max)
            // (그림 2)의 선분 L10 //
        endif
else
        if x1 < xw_min then
        if x2<xw_min then return endif
            // (그림 3)의 선분 L1 //
        if y1<yw_min then return endif
            // (그림 3)의 선분 L2 //
        if  y2>yw_max  then  return
            endif
            // (그림 3)의 선분 L3 //
              call x_clip(x1,y1,xw_min)
        if y1<yw_min then return endif
            // (그림 3)의 선분 L4 //
        if x2 > xw_max then
            // (그림 3)의 선분 L5 //
              call x_clip(x2,y2,xw_max)
        endif
        if  y2>yw_max  then  return
            endif
            // (그림 3)의 선분 L6 //
        if y1>yw_max then
            // (그림 3)의 선분 L7 //
              call y_clip(x1,y1,yw_max)
        endif
        if y2<yw_min then
```

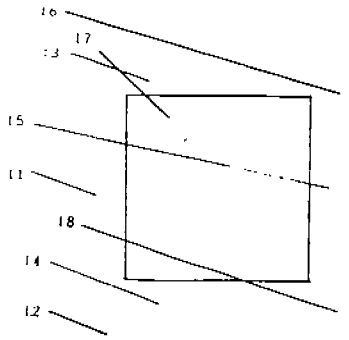
```
            // (그림 3)의 선분 L8 //
          call y_clip(x2,y2,yw_min)
          endif
      else if x2>xw_max then
          if x1>xw_max then return endif
              //(그림 4)의 선분 L1 //
          if y1<yw_min then return endif
              //(그림 4)의 선분 L2 //
          if  y2>yw_max  then  return
              endif
              //(그림 4)의 선분 L3 //
          m←(y2−y1)/(x2−x1)
          call x_clip(x2,y2,xw_max)
          if y2 yw_max then return endif
              //(그림 4)의 선분 L4 //
          if y1>yw_max then
              // (그림 4)의 선분 L5 //
                call y_clip(x1,y1,yw_max)
          endif
          if y2<yw_min then
              //(그림 4)의 선분 L6 //
                call y_clip(x2,y2,yw_m
          endif
      endif
      else if y2<yw_min then
          if y1<yw_min then return endif
              //(그림 4)의 선분 L7 //
          m←(y2−y1)/(x2−x1)
          call y_clip(x2,y2,yw_min)
          if y1 > yw_max then
              //(그림 4)의 선분 L8 //
                call y_clip(x1,y1,yw_max)
          endif
        endif
      else if y1>yw_max then
          if y2>yw_max then return endif
              //(그림 4)의 선분 L9 //
          m←(y2−y1)/(x2−x1)
          call y_clip(x1, y2, yw_max)
              (그림 4)의 선분 L10 //
          endif
endif
line(x1,y1,x2,y2)// draw a line //
end line_clip
```



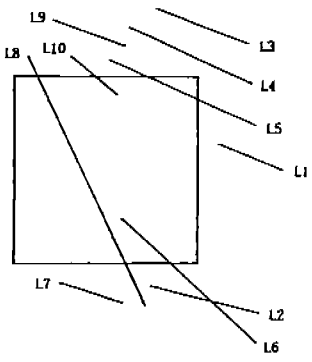
(그림 1) 알고리즘에서 사용하는 선분의 예
(Fig. 1) Lines for the algorithm



(그림 2) 알고리즘에서 사용하는 선분의 예
(Fig. 2) Lines for the algorithm



(그림 3) 알고리즘에서 사용하는 선분의 예
(Fig. 3) Lines for the algorithm



(그림 4) 알고리즘에서 사용하는 선분의 예
(Fig. 4) Lines for the algorithm

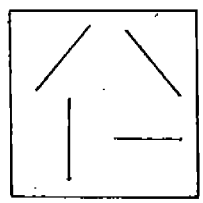
3. 실증 및 분석

기존의 선분절단 알고리즘 중 가장 많이 사용하고 있는 Cohen-Sutherland알고리즘은 코딩 방식에 기초를 둔다. 이 방법은 각 선분의 끝점마다에 그 점의 좌표 영역을 나타내는 4자리의 2진 영역 코드를 부여하여 어느 선분이 완전히 윈도우 내부에 있고, 완전히 윈도우 외부에 있는지를 결정할 수 있다. 이러한 검사로 완전히 내부에 있거나 외부에 있다고 결정할 수 없는 선분들에 대해서는 윈도우 경계와의 교차 여부를 조사한다. 선분의 어느 부분이 버려져야 하는지를 결정하기 위하여 윈도우 경계와 윈도우 외부에 있는 끝점을 비교하여 처리한 다음에 남은 부분들을 다른 경계에 대해 조사한다. 이러한 과정을 그 선분이 완전히 버려지거나 윈도우 내부의 부분을 찾을 때까지 계속해 나간다. 이 방법은 모든 선분마다 영역 코드를 부여하기 위하여 8회의 비교 연산을 수행해야 한다. 선분이 완전히 윈도우 내부에 있는지를 결정하기 위해서 영역 코드를 부여하는 데 8회의 비교 연산, 4회의 비트별 OR 연산과 4회의 비교 연산을 포함하여 총 14회의 비교 연산과 4회의 비트별 OR 연산이 필요하다. 선분이 완전히 윈도우의 밖에 있는지를 결정하기 위해서는 완전히 내부에 있는 경우보다 4회의 비트별 AND 연산과 4회의 비교 연산이 추가로 소요된다. 또한 적어도 선분의 한 끝점이 윈도우 밖에 있고 선분이 윈도우 내부를 지나는 경우에는 한 끝점을 윈도우 경계에 대하여 절단한 후(최대 23회의 비교연산, 8회의 비트별 연산과 최대 4회의 경계선과의 교점을 구하는 연산이 소요), 절단된 새로운 좌표에 대하여 다시 영역 코드를 부여하여야 하므로 비효율적이다.

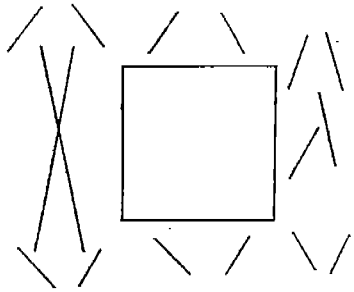
반면 본 알고리즘에서는 선분이 완전히 윈도우 내부에 있는 경우 5회의 비교 연산만이 필요하며, 완전히 윈도우 밖에 있는 경우 최대 6회의 비교 연산이 소요된다. 또한 적어도 선분의 한 끝점이 윈도우 밖에 있고 선분이 윈도우 내부를 지나는 경우, 13회의 비교 연산과 최대 4회의 경계선과의 교점을 구하는 연산만이 소요되므로 매우 효율적이다.

Cohen-Sutherland 알고리즘과 본 알고리즘을 비교하기 위하여 입력 선분을 크게 4가지 유형으로 분류하였다. 첫번째 유형의 입력 선분은(그림 5)와 같이 완전히 윈도우 내부에 있는 선분들로서 윈도우 경계와 절단을 하지 않고 윈도우 내부에 있음을 식별할 수 있는 경우이다. 두번째 유형의 입력 선분은(그림 6)과 같이 완전히 윈도우 외부에 있는 선분들로서 윈도우 경계와 절단을 하지 않고 외부에 있음을 식별할 수 있는 경우이다. 세번째 유형의 입력 선분은(그림 7)과 같이 적어도 선분의 한 끝점이 윈도우 외부에 있고 윈도우 내부를 지나는 선분들로서, 윈도우 경계와 절단 후 윈도우 내부에 있는 부분만을 받아 들이는 경우이다. 또한, 네번째 유형의 입력 선분은(그림 8)과 같이 선분이 완전히 윈도우 외부에 있으나, 이를 식별해 내기 위해서는 윈도우 경계와 절단이 필요한 경우이다.

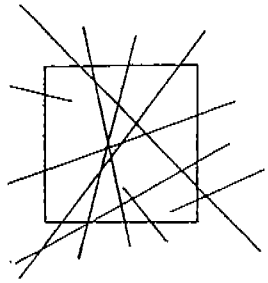
〈표 1〉~〈표 4〉는 4가지 유형의 입력 선분에 대하여 두 알고리즘에서 필요한 연산의 수를 비교하여 나타낸 것이다.



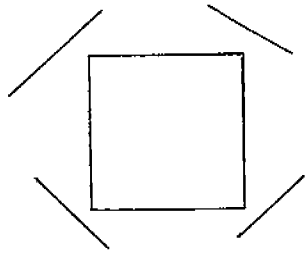
(그림 5) 윈도우 내부의 선분 예
(Fig. 5) Lines inside the window



(그림 6) 윈도우 외부의 선분 예
(Fig. 6) Lines outside the window



(그림 7) 절단 후 윈도우 내부의 선분 예
(Fig. 7) Lines inside the window by clipping



(그림 8) 절단 후 윈도우 외부의 선분예
(Fig. 8) Lines outside the window by clipping

〈표 1〉 윈도우 내부의 선분에 대한 소요 연산 수
〈table 1〉 Comparison of operation count of lines inside the window

연 산	Cohen-Sutherland 알고리즘	본 알고리즘
비 교	14회	5회
비트별 OR	4회	0회

〈표 2〉 윈도우 외부의 선분에 대한 소요 연산 수
〈Table 2〉 Comparison of operation count of lines outside the window

연 산	Cohen-Sutherland 알고리즘	본 알고리즘
비 교	19회	최대 : 6회 최소 : 3회
비트별 OR	4회	0회
비트별 AND	4회	0회

〈표 3〉 절단 후 윈도우 내부의 선분에 대한 소요 연산 수
〈Table 3〉 Comparison of operation count of lines inside the window by clipping

연 산	Cohen-Sutherland 알고리즘	본 알고리즘
비 교	최대 : 46회, 최소 : 37회	10회
비트별 OR	8회	0회
비트별 AND	최대 : 8회, 최소 : 4회	0회
덧 셈	최대 : 4회, 최소 : 1회	최대 : 2회, 최소 : 1회
뺄 셈	최대 : 8회, 최소 : 3회	최대 : 4회, 최소 : 3회
곱 셈	최대 : 2회, 최소 : 0회	최대 : 1회, 최소 : 0회
나 눗 셈	최대 : 4회, 최소 : 1회	최대 : 2회, 최소 : 1회

〈표 4〉 절단 후 윈도우 외부의 선분에 대한 소요 연산 수
〈Table 4〉 Comparison of operation count of lines outside the window by clipping

연 산	Cohen-Sutherland 알고리즘	본 알고리즘
비 교	42회	최대 : 8회, 최소 : 6회
비트별 OR	8회	0회
비트별 AND	8회	0회
덧 셈	최대 : 2회, 최소 : 1회	최대 : 2회, 최소 : 1회
뺄 셈	최대 : 4회, 최소 : 3회	최대 : 4회, 최소 : 3회
곱 셈	최대 : 1회, 최소 : 0회	최대 : 1회, 최소 : 0회
나 눗 셈	최대 : 2회, 최소 : 1회	최대 : 2회, 최소 : 1회

〈표 5〉 두 알고리즘의 수행시간 비교(단위 : 1/100초)
〈Table 5〉 Comparison of execution times

입력선분의 유형	Cohen-Sutherland 알고리즘	본 알고리즘
윈도우 내부의 선분	11	6
윈도우 외부의 선분	13	5
절단 후 윈도우 내부의 선분	63	20
절단 후 윈도우 외부의 선분	38	13

또한 두 알고리즘을 PC-486DX2-50상에서 Turbo C로 프로그래밍하여 수행속도를 비교하였다. [1, 2]. 〈표 5〉는 각 유형마다 입력 선분을 정하여 두 알고리즘을 같은 환경에서 구현하여 수행한 결과의 소요 시간을 비교한 것이다. 입력 선분의 수를 16 개로 정하였고 이 선분들의 절단을 1000회 반복한 것을 100분의 1초 단위로 나타낸 것이다. 이에 대한 C코드의 일부는 다음과 같다.

```
gettime( &start );
for( i=0; i<1000; i++ )
    for( j=0; j<16; j++ )
        clip_line( x1[ j ],y1[ j ],x2[ j ],y2[ j ] )
gettime( &end );
```

4. 결 론

본 논문에서는 사각형 윈도우에 대하여 선분 절단을 효율적으로 수행하는 알고리즘을 제안하고, 이를 PC상에서 구현하여 기존의 Cohen-

Sutherland 알고리즘과 비교하여 그 효율성을 보였다.

선분을 구성하는 양끝점의 좌표 중 어느 한 좌표값이 윈도우 경계를 벗어나는 4가지 경우로 나누어, 각각의 경우에 따른 좌표값이 윈도우 경계와 어떤 관계에 있는가에 따라 완전히 윈도우 외부에 있는 선분, 절단 후 윈도우 외부에 있는 선분, 절단 후 윈도우 내부에 있는 선분, 완전히 윈도우 내부에 있는 선분임을 차례로 식별해 낸다. 반복문을 사용하지 않고 절단에 필요한 산술 및 논리 연산의 수를 최대로 줄였다.

본 알고리즘은 수 많은 선분으로 구성된 화상을 그리는데 매우 효율적이다.

참 고 문 헌

[1] Donald Hearn and M.Pauline Baker, Computer Graphics, Prentice Hall, 1988.

[2] Steven Harrington, Computer Graphics-A programming Approach, 2nd ed., Xerox, 1987.

[3] J.D.Foley,A.vanDam,S.K.Feiner, and J.F. Hughes, Computer Graphics Principles and practice, 2nd ed., Addison-Wesley, 1990.

[4] M.S.Sobkow,P.Pospisil,and Y-H Yang, A fast two-dimensional line clipping algorithm via line encoding,Computers & Graphics 11, pp. 459-467, 1987.

[5] Y-D Liang and B.A.Barsky, A new concept and method for line clipping, ACM Transactions on Graphics 3, pp. 1-22, 1984.

[6] M. Dorr, A new approach to parametric line clipping, Computer Graphics 14, pp. 449-464, 1990.

[7] Nelson Johnson, Advanced in C-Programming and Techniques, MCGraw-Hill, 1987.

김 응 곤

1980년 조선대학교 전자공학과 졸업(학사)
1987년 한양대학교 대학원 전자공학과(공학석사)
1992년 조선대학교 대학원 전기공학과 전산전공(공학박사)
1984년~85년 금성반도체(주) 연구소 연구원
1987년~91년 국방과학연구소 선임연구원
1991년~93년 여수수산대학교 전임강사
1993년~현재 순천대학교 조교수
관심분야 : 컴퓨터그래픽스, CAD

허 영 남

1967년 공주사범대학 졸업(학사)
1982년 조선대학교 대학원 전자공학과(공학석사)
1992년~현재 조선대학교 대학원 전산통계학과 박사과정 수료
1983년~86년 순천대학교 전자계산소장
1983년~현재 순천대학교 전자계산학과 교수
관심분야 : 영상처리, 병렬처리

이 웅 기

1975년 조선대학교 전자공학과 졸업(학사)
1981년 명지대학교 대학원 전자공학과(공학석사)
1980년~88년 조선대학교 컴퓨터공학과 조교수
1985년~86년 미국 Oregon Graduate Center 객원 교수
1988년~현재 조선대학교 전산통계학과 부교수
관심분야 : 영상처리, 계산기구조