직선 선분의 대칭성을 이용한 수정 브레제남 직선 그리기 알고리즘

이 상 락† · 홍 윤 식†

요 약

직선 선분은 선분의 중점에 대하여 대칭 성질을 갖고 있다. 우리는 이 대칭성에 착안하여 직선 생성을 위한 픽셀 선택 시 한 번에 두 개의 픽셀을 동시에 선택할 수 있는 수정 브레제남 알고리즘을 제안하였다. 본 알고리즘은 한번에 두 개의 픽셀 위치를 결정하며, 픽셀 별로 각각의 결정 파라미터(decision parameter)를 사용하는 것이 아니라 오직 한 개의 결정 파라미터만을 사용한다. 이러한 선택 방법을 뒷받침하기 위한 이론적 증명을 아울러 제시하였다. 본 논문에서 제안한 알고리즘은 픽셀 선택을 위한 루프 수행 회수가 줄어들기 때문에 직선 생성에 걸리는 시간을 약 5% 내외로 단축할 수 있음을 실험을 통하여 알 수 있었다. 또한, 생성된 직선의 모양은 기존 브레제남 알고리즘의 그것과 일치함을 확인할 수 있었다.

A Modified Bresenham's Line Drawing Algorithm Using Symmetrical Property of Line Segment

Sang-Rak Lee† · Youn-Sik Hong†

ABSTRACT

A line segment has a *symmetrical* property about its midpoint. With this symmetrical property for a line segment we have proposed a variant of *Bresenham's* line drawing algorithm that selects two pixels at the same time. It implies that *bi-directional* line drawing toward the midpoint of line segment from each-end be possible. Besides, it can select two pixels using only one decision parameter, instead of two different parameters for each pixel. We also present a theoretical proof for the correctness of such a selection. Thus, we can reduce the time of generating line segment by approximately less than 5% compared to the *Bresenham's* method. Also, our experimental results show that the shape of line segment generated by the proposed approach is the exactly same as that of the *Bresenham's* method.

1. 서 론

직선은 컴퓨터 그래픽에서 중요한 출력 기본 요소 중 하나로서 원하는 직선을 얼마나 신속하게 그릴 수 있는가 하는 것이 화면에서 그림을 얼마나 빨리 그려낼 수 있는가 하는 문제의 관건이 된다. 특히 물체를 wire frame 모델로 많이 나타내는 CAD와 같은 분야에서는 직선의 신속한 생성이 매우 긴요한 일이다.

그래픽에서 직선을 그리기 위한 방법은 여러 가지가 있어 이에 관한 상당수의 연구 결과가 발표되어 있다 [1]-[12]. 초기에는 디지털 플로터(digital plotter) 에서의 직선을 그리기 위한 방법인 DDA(digital differential analyzer) 가 주로 사용되었다. BRM(binary rate mult-

※ 본 연구는 한국 과학 재단 지역 협력 연구 센터(RRC) 사업의 연구비 지원에 의한 결과임.
† 정 회 원 : 인천대학교 전자계산학과 교수
논문접수 : 1999년 3월 10일, 심사완료 : 1999년 7월 20일

iplier)은 그 방법이 단순하여 직선을 빠르게 그릴 수는 있었으나 정확성은 매우 불량하였다[9]. Suenaga등[11]은 직선을 정확하고 빠르게 구할 수 있는 방법인 MDCM (modified displacement comparision method)을 제안하고 그 이전의 연구인 Danielsson[5]의 방법, Jordan et al.[6]의 방법 및 Kamae[7]의 방법과의 차이점을 설명하고 있다. Sproll[9] 은 알고리즘 변형 기법을 이용한 여러 가지 Bresenham의 변형을 소개하고 특히 한번에 여러 개의 픽셀을 계산하는 병렬처리를 제안하고 있다. Baker[12]에서는 멀티 프로세서를 이용한 병렬 처리 방법으로 직선을 프로세서의 수만큼의 구간으로 나누어 각 프로세서가 하나의 구간을 맡아 직선을 그리는 방법을 소개하고 있다. 또 프로세서의 수가 굉장히 많을 때에는 각 프로세서가 일정한 범위 내에 있는 픽셀들을 하나씩 맡아서 각 픽셀이 직선으로부터 떨어진 거리를 계산하여 직선을 그리는데 필요한 픽셀을 선택하는 방법도 있음을 보이고 있다. Pokorney[2]는 직선 그리기를 구조적 방법(structural method)과 조건적 방법(conditional method)으로 구분하여 구조적 방법인 브론의 방법[3]과 Casthe과 Prittway[4]가 제안한 best-fit방법을 소개하고 있다. 그러나 이들의 조건적 방법은 정확한 직선을 그리기가 어렵다고 보여진다.

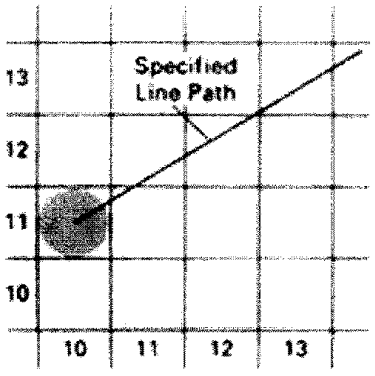
직선 그리기에는 위와 같은 많은 방법이 있으나 이들 중에서 Bresenham의 알고리즘이 일반적으로 많이 사용되고 있다. 그러나 우리는 이것을 변형시켜 시간 복잡도(time complexity)측면에서 최고 차수 항의 상수 값을 줄일 수 있는 알고리즘을 발견하였다. Bresenham은 결정 파라메터(decision parameter)의 값에 따라 필요한 픽셀을 선택하면서 직선 그리기를 직선의 시작점에서부터 올라가면서 그리기 시작한다. 그러나 한 직선 선분은 선분의 중간 점에 대하여 대칭의 성질을 가지고 있다. 이 대칭성을 이용하여 직선의 한쪽 끝에서 상향된 위치의 픽셀을 선택하였다면 직선의 다른 끝 쪽에서는 하향된 위치의 픽셀을 동시에 선택하여도 논리적으로 잘못된 것이 없다고 생각하였다. 그리고 이러한 생각을 뒷받침하기 위해 직선을 내려오면서 생성하는데 필요한 픽셀을 선택하기 위한 결정 파라메터의 식을 유도해 내었다. 그 결과 공교롭게도 직선을 올려 그릴 때의 파라메터의 식과 내려 그릴 때의 파라메터의 식이 같음을 발견하였다. 이것은 한번 파라메터의 값을 조사하여 두 개의 픽셀을 선택할 수 있음을 의미한다. 결국 이것은 알고리즘의 수행에 필요한 루프의 실행 횟수를 반으로 줄일 수 있게 함으로서 직선을 그리기 위한 소요 시간에서 상당한 이득을 볼 수 있게 한다.

본 논문은 다음과 같이 구성되었다. 2장에서는 본 논문에 대한 이해를 돕기 위해 이미 알려진 Bresenham의 직선 그리기 알고리즘을 살펴본다. 그리고 3장에서는 본 논문에서 제시하고자 하는 수정된 Bresenham의 직선 그리기 알고리즘(Modified BLGA)을 제시한다. 4. 5장은 알고리즘의 정당성과 관련되는 정리를 포함하여 알고리즘의 분석과 실행 결과를 보인다. 마지막으로 6장은 결론이다.

2. 브레제남 직선 그리기 알고리즘

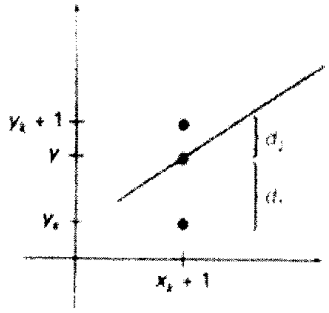
브레제남 직선 그리기 알고리즘은 오직 정수 연산만을 이용하여 직선을 그림으로서 정확하고 효율적이다. 이 알고리즘은 원이나 다른 곡선을 그리는데도 적용될 수 있다[10].

(그림 1)은 직선 선분이 그려질 디스플레이 스크린이다. 수직 축은 스캔 라인의 위치이고 수평 축은 픽셀 칼럼이다. 직선을 그리기 위해서는 각 픽셀 칼럼에서 두 개의 가능한 픽셀 중에서 어느 것이 실제의 직선 경로에 더 가까운지를 결정해야 한다. 그림1에서 왼쪽의 시작점에서 시작하여 다음의 선택 위치에서는 (11,11)의 픽셀이나 (11,12)의 픽셀 중에서 하나를 선택하여야 한다. 브레제남의 알고리즘은 실제의 직선으로부터 두 픽셀 위치까지의 거리 차이에 비례하는 파라메터를 도입하고 이 파라메터의 부호를 검사하여 두 픽셀 중의 하나를 결정하는 방법이다.



(그림 1) 직선이 그려지는 디스플레이 스크린

브레제남의 접근 방법을 좀더 구체적으로 설명하기 위해 기울기가 1보다 작고 양수인 직선에 대한 스캔라인 컨버젼 과정을 살펴본다. 기울기가 1보다 클 경우에는 x 와 y의 역할을 바꾸어 생각하면 된다.



(그림 2) 픽셀 위치와 직선의 y 좌표 사이의 거리(x 좌표의 증가)

직선 상에 있는 픽셀의 위치는 x 좌표에 대하여 y 좌표를 선택함으로서 결정된다. 주어진 직선의 왼쪽 끝점 (x_0, y_0) 에서 시작하여 다음 칼럼(x 좌표)에서 스캔 라인 y 값이 직선 경로에 가장 가까운 픽셀을 그려나간다. 그림2 는 이 과정에서 k번째 단계를 나타내고 있다. 픽셀 (x_k, y_k)를 디스플레이하기로 결정했다면 다음은 x_{k+1}번째 칼럼에 있는 픽셀을 결정해야 한다. 이것은 (x_{k+1}, y_k)과 (x_{k+1}, y_k+1)의 두 픽셀 중에서 하나를 선택하는 것이다. 여기서 $x_{k+1}=x_k+1$이다. x_{k+1}에서 각 픽셀이 직선과 떨어진 거리를 d_1, d_2 로 표시한다. 칼럼 x_{k+1}에서 직선의 y값은 다음과 같다.

$$y=m(x_k+1)+b$$

그러면 $d_1=y-y_k$

$$=m(x_k+1)+b-y_k \text{ 이고}$$

$$d_2=(y_k+1)-y$$

$$=y_k+1-m(x_k+1)-b \text{ 이다.}$$

d_1과 d_2의 차는

$$d_1-d_2=2m(x_k+1)-2y_k+2b-1 \text{ 이다}$$

k번째 스텝에서 결정 파라메터는 정리하여 다음과 같이 표시된다.

$$p_k=\Delta x(d_1-d_2)$$

$$=2\Delta y\cdot x_k-2\Delta x\cdot y_k+c$$

여기서 c는 상수항으로 $2\Delta y+\Delta x(2b-1)$의 값을 가진다. y_k의 픽셀이 y_{k+1}의 것보다 직선에 더 가까우면 (즉 $d_1<d_2$) p_k는 음수이다. 이 경우 아래쪽 픽셀을 그대로 선택하고 그렇지 않은 경우에는 위의 픽셀을 선택한다. 즉 상향된 위치의 픽셀을 선택하는 것이다. 나아가서 스텝 k+1에서는 p_k를 이용하여 p_{k+1}을 좀더 편리하게 증분의 방법으로 구할 수 있도록 파라메터를 순환식으로 표현하면 결정 파라메터는 다음과 같이 표현할 수 있다.

$$p_{k+1}=p_k+2\Delta y-2\Delta x(y_{k+1}-y_k)$$

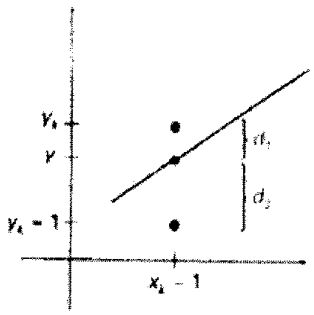
여기서 $y_{k+1}-y_k$는 p_k의 부호에 따라 0 또는 1의 값을 가지기 때문에 p_{k+1}의 값은 쉽게 구해질 수 있다. p_k의 초기값 p_0는 직선의 출발점 좌표 (x_0, y_0)를 이용하여 다음과 같이 구해진다.

$$p_0=2\Delta y-\Delta x$$

위의 식들에서도 알 수 있지만 브레제남의 알고리즘은 오직 정수 연산만을 사용하기 때문에 매우 효율적이다.

3. 수정 브레제남 직선 생성 알고리즘

본 논문의 기본 착상을 논리적으로 설명하기 위해 먼저 본래의 알고리즘과는 반대로 직선의 오른 쪽 끝점에서 시작하여 왼쪽으로 내려오면서 직선을 생성하는 방법을 생각해보기로 한다.



(그림 3) 픽셀 위치와 직선의 y 좌표 사이의 거리(x좌표의 감소)

(그림 3)에서 현재 픽셀 (x_k, y_k)를 선택하였다면 다음 스텝에서는 (x_{k+1}, y_{k+1})을 선택하여야 하는데 (x_{k+1}, y_{k+1})은 (x_{k+1}, y_k) 와 (x_{k+1}, y_k-1) 의 둘중의 하나이다. 여기서 $x_{k+1}=x_k-1$이고 y_{k+1}은 y_k이거나 y_k-1이다.

(그림 3)에서 실제 직선의 y 값이 스캔 라인 y_k와 y_k-1 로부터 떨어진 거리를 각각 d_1와 d_2라 하면 이들은 다음과 같다.

$$\begin{aligned} d_1 &= y_k - y \\ &= y_k - mx_{k+1} - b \\ d_2 &= y - y_{k+1} \\ &= mx_{k+1} + b - y_{k+1} \\ &= mx_{k+1} + b - (y_k - 1) \end{aligned}$$

그러므로

$$\begin{aligned} d_1 - d_2 &= y_k - mx_{k+1} - b \\ &\quad - mx_{k+1} - b + (y_k - 1) \\ &= -2mx_{k+1} + 2y_k - 2b - 1 \end{aligned}$$

여기서 새로운 결정 파라메터 q_k를 도입한다.(그러나 실제 알고리즘에서는 이 파라메터가 필요하지 않다)

$$\begin{aligned} q_k &= \triangle x(d_1 - d_2) \\ &= \triangle x(-2mx_{k+1} + 2y_k - 2b - 1) \end{aligned}$$

m = △y/△x 이고 $x_{k+1}=x_k-1$ 임을 이용하여 이를 정리하면 다음과 같다.

$$\begin{aligned} q_k &= -2\triangle y\, x_k + 2\triangle x\, y_k - c \\ &= -(2\triangle y\, x_k - 2\triangle x\, y_k + c) \end{aligned}$$

단 $c = \triangle x(2b+1) - 2\triangle y$

$\triangle x$가 양수이므로 $q_k<0$ (즉 $d_1 < d_2$) 이면 다음 스텝에서는 픽셀 (x_{k+1}, y_k)을 그대로 선택하고 $q_k>0$ (즉 $d_1 > d_2$) 이면 픽셀 (x_{k+1}, y_k-1) 를 선택한다. 즉 하향된 위치의 픽셀을 선택하는 것이다.

q_k의 부호에 따른 픽셀의 선택 방법을 잘 살펴 보면 p_k의 부호에 따른 픽셀의 선택 방법과 같다. 즉 $p_k<0$ 이어서 y의 변동이 없이 그대로 같은 높이의 픽셀을 선택하였다면 $q_k<0$일 때에도 같은 방법으로 픽셀을 선택하게 되고, $p_k \geq 0$ 이어서 상향된 y 값의 픽셀을 선택하였다면 $q_k \geq 0$ 일때에는 역시 같은 룰내를 내어 하향된 y 값의 픽셀을 선택한다는 것을 알 수 있다.

상수항의 영향을 배제하고 q_k의 값을 좀더 효과적으로 구하기 위해 q_k를 순환식으로 표현하면 다음과 같다.

$$q_{k+1} = -(2\triangle y\, x_{k+1} - 2\triangle x\, y_{k+1} + c)$$

$$q_{k+1} - q_k = -(2\triangle y\,(x_{k+1} - x_k) - 2\triangle x\,(y_{k+1} - y_k))$$

$x_{k+1} = x_k - 1$이므로

$$\therefore q_{k+1} = q_k + (2\triangle y - 2\triangle x\,(y_k - y_{k+1}))$$이다.

여기서 $y_k - y_{k+1}$은 0 또는 1이다.

결과적으로 q_{k+1}는 p_{k+1}과 같은 값을 가지는 것을 알 수 있다.

q_k의 초기값은 오른 쪽 끝점 좌표 (x_e, y_e)를 사용하여 구하면 다음과 같다.

$$q_0 = -2\triangle y + \triangle x$$

지금까지 유도된 식들은 앞의 2절에서 유도된 식들과 비교해보면 식의 형식은 같고 다만 부호만 다르다. 이것은 (그림 2)와 (그림 3)에서 보듯이 d_1과 d_2를 서로 바꾸어 나타내었기 때문이다.

지금까지의 논의를 바탕으로 하여 본 논문에서 제시하고자 하는 수정된 브레제남 직선 생성 알고리즘을 제시하면 다음과 같다.

수정 브레제남 직선 생성 알고리즘($|m| \leq 1$ 인 경우)

1. 직선의 두 끝점 좌표 (x_0, y_0), (x_e, y_e)를 입력
2. 상수 $\triangle x$, $\triangle y$, $2\triangle y$ 및 $2\triangle y - 2\triangle x$를 계산
3. 파라메터의 초기값 p_0를 구한다.

$$p_0 = 2\triangle y - \triangle x$$

4. 변수 초기화

$$x_1 = x_0, \quad y_1 = y_0, \quad x_2 = x_e, \quad y_2 = y_e$$

5. $k=0$에서 시작하여 각 k에서 다음의 블록을 수행한다.

plot (x_1, y_1) , plot (x_2, y_2)

$$x_1 = x_1 + 1, \quad x_2 = x_2 - 1$$

if $p_k<0$

$p_{k+1}=p_k+2\triangle y$

if $p_k>=0$

$y_1=y_1+1,\quad y_2=y_2-1$

$p_{k+1}=p_k+2\triangle y-2\triangle x$

6. 스텝 5를 $x_2<x_1$일 때까지 반복.

7. plot (x_1, y_1)

8. 끝

4. 증명 분석 및 실행

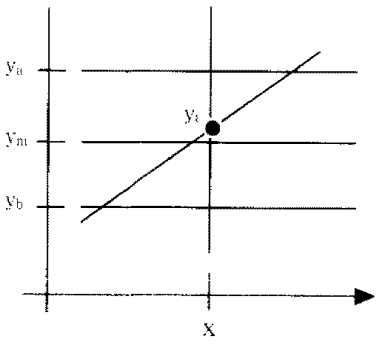
4.1 증 명

수정 알고리즘에서는 직선의 두 끝에서 서로 마주보면서 생성해 갈 때 중간의 만나는 점에서 과연 정확하게 일치하는지, 일치하지 않으면 몇 픽셀 정도로 어긋나는 지를 밝힐 필요가 있다. 이를 위하여 다음의 정리를 제시하고 이를 증명한다.

<보조정리>

임의의 x 칼럼에서 선택된 직선의 y 좌표가 y_a이고 직선의 실제 y 값이 y_t 라면 $|y_a-y_t|\le\frac{1}{2}$ 이다.

<증명>



(그림 4) 스캔라인과 실제 직선 사이의 거리

(그림 4)에서 두 스캔 라인 y_a와 y_b의 중간점을 y_m라 하자. 그러면 $|y_a-y_m|\le\frac{1}{2}$ 이다. y_t는 y_a와 y_m의 사이에 있으므로 따라서 $|y_a-y_t|\le\frac{1}{2}$ 이다. y_t가 y_m와 y_b의 사이에 있어도 결과는 동일하다.

정리

x를 올라 가면서 생성되는 직선과 내려가면서 생성되는 직선이 만나는 곳의 x좌표라 하자. 그리고 x에서 올라가는 것의 선택한 y좌표를 y_1, 내려가는 것의 y좌표를 y_2라 하자. 그러면 $|y_1-y_2|\le1$이다.

<증명>

x 좌표에서 실제 직선의 y 값을 y_t 라면 보조 정리로부터 $|y_1-y_t|\le\frac{1}{2}$ 이고 $|y_2-y_t|\le\frac{1}{2}$ 이다. 고로

$|y_1-y_t-y_2+y_t|\le1$이고

따라서 $|y_1-y_2|\le1$이다.

4.2 분 석

원래의 브레제남 알고리즘의 수행 회수는 $\triangle x$에 비례한다. 즉 $\triangle x=n$ 이라면 이 알고리즘의 시간 복잡도는 $\Theta(n)$이다.

본 논문에서 제시하는 수정알고리즘은 한번에 두 개의 픽셀을 선택하므로 루프의 수행회수가 원래의 알고리즘에 비하여 절반 밖에 되지 않는다. 즉 이 알고리즘의 시간 복잡도는 $\Theta(\frac{1}{2}n)$이다. 수정 알고리즘은 루프의 수행 회수가 반으로 줄어드는 대신 루프 속에서의 명령은 2배이다. 일반적으로 루프의 반복과 관련되는 조건의 검사는 대입 명령문보다 컴퓨터 내부에서의 소요 시간이 더 크다. 따라서 단순히 알고리즘적 측면만을 고려한다면 직선을 생성하기 위해 선택해야 할 픽셀이 많으면 많을수록 수정된 알고리즘은 원래의 알고리즘보다 분명히 개선된 효과를 얻을 수 있다. 그러나 뒤의 실험 결과는 이러한 기대와 조금 다르게 나타나는데 이것은 픽셀의 주소를 구하는 시간보다 픽셀의 정보를 저장하기 위해 해당 픽셀의 주소에 접근하는 하드웨어적인 시간이 더 많이 소요되기 때문이라고 생각된다.

4.3 실 행

아래 <표 1a>와 <표 1b>는 직선의 시작점과 끝점을 각각 (0,0)과 (10,5)로 하여 원래의 브레제남 알고리즘과 수정된 알고리즘을 사용하여 실행을 한 결과이다. 두 표를 비교해 보면 결과는 같다는 것을 알 수 있다.

〈표 1〉 수정 브레제남 알고리즘(b)과 원 알고리즘(a) 간 비교

Original BLGA start point (0, 0) end point (11, 5) (x , y)	Modified BLGA start point (0, 0) end point (11, 5) (x1 , y1) (x2, y2)
(0 , 0)	(0 , 0) (11, 5)
(1 , 0)	(1 , 0) (10, 5)
(2 , 1)	(2 , 1) (9, 4)
(3 , 1)	(3 , 1) (8, 4)
(4 , 2)	(4 , 2) (7, 3)
(5 , 2)	(5 , 2) (6, 3)
(6 , 3)	
(7 , 3)	
(8 , 4)	
(9 , 4)	
(10 , 5)	
(11 , 5)	
OK (a)	OK (b)

<표 2a>와 <표 2b>는 기울기가 $\frac{1}{2}$인 직선을 가지고 실행을 해 본 결과이다. 이 경우에도 두 알고리즘의 실행 결과는 서로 같다는 것을 알 수 있다. 다만 x 좌표가 7 일 때 y좌표는 각각 3과 4로서 조금 다르다. 이것은 기울기가 $\frac{1}{2}$이기 때문에 직선이 픽셀의 중심을 지나는 것으로 어느 픽셀을 선택하여도 상관이 없는 것이다.

〈표 2〉 기울기가 1/2일 때 수정 브레제남 알고리즘(b)과 원 알고리즘(a) 간 비교

Original BLGA start point (0, 0) end point (8, 4) (x , y)	Modified BLGA start point (0, 0) end point (8, 4) (x1 , y1) (x2, y2)
(0 , 0)	(0 , 0) (8, 4)
(1 , 1)	(1 , 1) (7, 3)
(2 , 1)	(2 , 1) (6, 3)
(3 , 2)	(3 , 2) (5, 2)
(4 , 2)	(4 , 2) (4, 2)
(5 , 3)	
(6 , 3)	
(7 , 4)	
(8 , 4)	
OK (a)	OK (b)

5. 실험 결과 및 분석

본 논문에서 제안하는 방식(이하 수정 (브레제남) 알고리즘)의 타당성 여부를 입증하기 위해 다음 두 가지 종류의 실험을 진행하였다. 수정 브레제남 알고리즘의 가장 큰 장점은 한 개의 결정 파라미터(decision parameter)를 사용해 동시에 2개의 픽셀 위치를 결정하므로, 원 브레제남 알고리즘에 비해 실행 속도가 크게 향상된다는 점이다.

5.1 실행 속도 비교 및 결과 분석

따라서, 첫 번째 실험에서는, 원 알고리즘과 수정 알고리즘 간 실행 속도를 비교해 보았다(<표 3>, (그림 5)~(그림 7)). 이를 위해 각각에 대해 픽셀 개수가 100,000개에서 최대 20,000,000개로 이루어진 단일 선분(single line segment)을 그리는 데 걸리는 시간을 측정하였다. 순수하게 픽셀 위치 계산에 걸리는 시간만을 측정하기 위해, 윈도우 화면에 선분을 그리기 위한 그래픽 함수(예: g.drawLine(x1,y1, x2,y2) ;)호출은 시간 측정에서 배제하였다.

기울기가 1일 때, 픽셀 개수 변동에 따른 두 알고리즘간 실행 속도 비교 결과를 <표 3>에 나타내었다. 두 개의 알고리즘 모두 java1.2 언어를 사용하여 구현하였고, pentium-II PC(300MHz)에서 실행하였으며, 시간 측정 단위는 ms(10^{-3}초)이다. 한편, 상대 시간만 측정하면 되므로 측정 시간(start time 및 stop time)은 시간 변화량이 있는 값(유효 숫자 5자리)만을 표시하였다.

<표 3>에서 알 수 있는 것처럼, 픽셀 개수가 1,000,000개 이하일 경우 수정 알고리즘은 거의 시간이 걸리지 않는다. 평균 속도를 비교할 경우, 픽셀 개수가 1,000,000~8,000,000개일 경우 수정 알고리즘의 속도는 원 브레제남 알고리즘의 15%에 불과하며, 10,000,000~20,000,000개일 경우에도 원 알고리즘의 52%에 지나지 않는다.

	평균 속도(단위:ms)		B/A (백분율)
픽셀 개수	원 브레제남 (A)	수 정 알고리즘(B)	
100,000~800,000	520	-	0
1,000,000~8,000,000	548	83	15.1%
10,000,000~20,000,000	660	345	52.3%

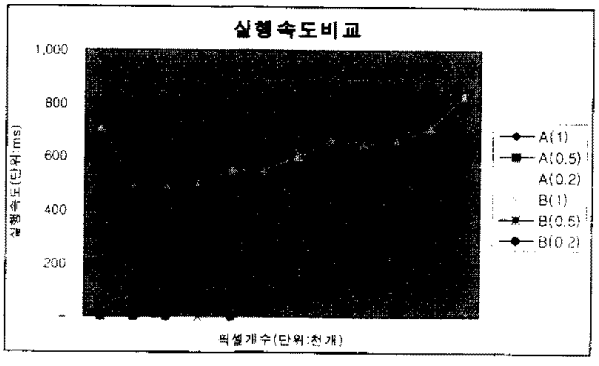
한편, 기울기 변동에 따른 두 알고리즘 간 실행 속

도 변화도 측정하였다. 기울기가 각각 1, 0.5, 0.2일 때 두 알고리즘 간 성능(실행 속도) 분석 결과를 (그림 5)에 나타내었다. 또한, 각 알고리즘 별로 기울기 변동에 따른 성능 실험 결과를 각각 (그림 6) 및 (그림 7)에 나타내었다. 원 브레제남 알고리즘의 경우(그림 6) 기울기 변동에 따른 속도 변화가 매우 심한 반면, 수정 알고리즘의 경우(그림 7) 실행 속도는 기울기 변동에 거의 무관함을 알 수 있다.

〈표 3〉 기울기 1일 때 원 브레제남 알고리즘 및 수정 알고리즘 실행 속도 비교

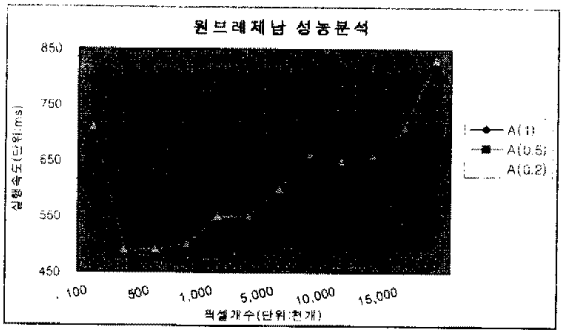
(단위 : milliseconds)

	원 브레제남 알고리즘			수정 브레제남 알고리즘			Z백분율 비교
	start time (A1)	stop time (A2)	차(A2 A1)	start time (B1)	stop time (B2)	차(B2-B1)	(B2-B1)/(A2-1)
100,000	65830	66320	490	66320	66320		0
200,000	63340	63830	490	63830	63830	-	0
500,000	75550	76100	550	76100	76100	-	0
800,000	38290	38840	550	38840	38840		0
1,000,000	77820	78310	490	78310	78310	-	0
2,000,000	50170	50720	550	50720	50770	50	9.1
5,000,000	64650	65200	550	65200	65310	110	20.0
8,000,000	63380	63980	600	63980	64150	170	28.3
10,000,000	41830	42490	660	42490	42710	220	33.3
12,000,000	39770	40430	660	40430	40710	280	42.4
15,000,000	91560	92220	660	92220	92610	390	59.1
20,000,000	46890	47550	660	47550	48040	490	74.2



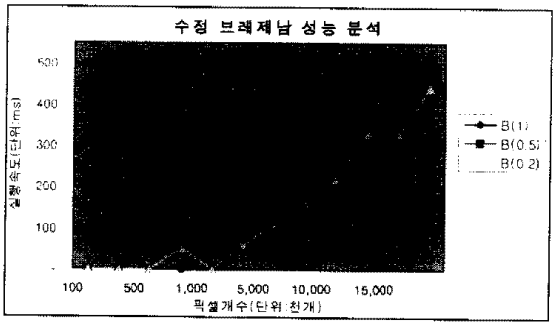
(그림 5) 기울기 변동에 따른 원 브레제남 알고리즘 및 수정 알고리즘 실행 속도 비교

단, A:원 브레제남 알고리즘, B:수정 브레제남 알고리즘을 가리키며, 괄호 안의 수치는 기울기를 가리킨다.



(그림 6) 기울기 변동(1,0.5,0.2)에 따른 원 브레제남 알고리즘 성능(실행 시간) 분석

단, A(1):기울기 1일 때, A(0.5):기울기 0.5일 때, A(0.2):기울기 0.2일 때를 가리킴



(그림 7) 기울기 변동(1,0.5,0.2)에 따른 수정 브레제남 알고리즘 성능(실행 시간) 분석

단, B(1):기울기 1일 때, B(0.5):기울기 0.5일 때, B(0.2):기울기 0.2일 때를 가리킴

5.2 선분 모양 및 실행 속도 비교 실험

두 번째 실험에서는, 수정 알고리즘에서 생성한 선분의 모양이 원 브레제남 알고리즘에 비해 어떤 변화가 있는지 여부를 알아보고자 하였다. 즉, 수정 브레제남 알고리즘의 경우 x축 증분(또는 y축 증분) 선택이 선분의 양 끝점에서 동시에 일어나므로 최종적인 선분의 모양에 어떤 영향을 미치는가를 확인하는 실험이다. 이를 위해 같은 기울기를 갖는 수백 개 이상의 직선을 그리고, 이에 따른 선분의 모양 및 실행 속도를 측정하였다. 이때, 시간 측정에는 그래픽 함수 호출 및 실행 시간이 모두 포함되었다. 기울기에 따른 직선 모양의 변동 유무를 측정하기 위해 기울기를 다양하게 변화시켜 보았다.

<표 4>에는 본 논문에서 제안한 알고리즘의 실행 속도를 요약하였다. 실험에 사용된 데이터의 픽셀 개수는 최저 10만개에서 최대 100만개까지이다. 데이터 2에

대해 본 논문에서 제안한 알고리즘을 적용하여 생성된 직선들을 (그림 9) 에 나타내었다. (그림 8) 에는 동일한 데이터에 원 브레제남 알고리즘을 적용하여 생성된 직선들을 나타내었다. (그림 9) 에서 3개의 꺾인 점(breaking point)들이 보이는 것은 aliasing 효과에 따른 것이며, 원 브레제남 알고리즘을 적용했을 때 직선 모양 역시 이러한 꺾인 점이 나타난다. 따라서, 생성된 직선 선분 모양에는 전혀 차이가 없음을 알 수 있다.

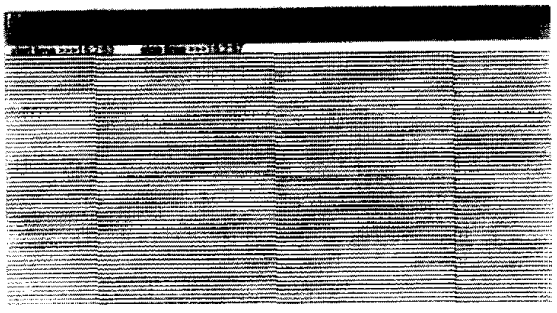
〈표 4〉 실험 결과(기울기 < 1)

	픽셀 개수	라인 개수	기울기	실 행 시간(초)
데이터 1	100,000	100	0.01	2
데이터 2	200,000	200	0.005	4
데이터 3	500,000	500	0.0025	9
데이터 4	1,000,000	1,000	0.0001	17

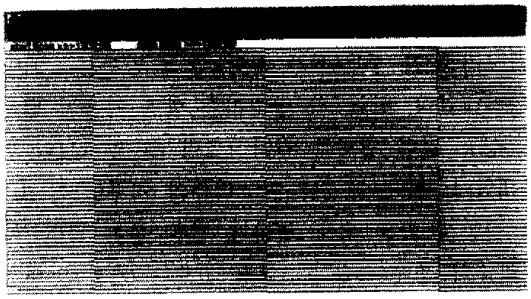
〈표 5〉 픽셀 개수에 따른 실행 시간 비교 결과

	픽셀 개수	본 논문 방식A (단위 ms)	원 브레제남 알고리즘 B (단위 ms)	B-A (단위 ms)
데이터 1	100,000	1720	1820	100 (5.5%)
데이터 2	200,000	3460	3630	170 (4.9%)
데이터 3	500,000	8770	9060	270 (3.0%)
데이터 4	1,000,000	17470	17790	320 (1.8%)

<표 5>에는 원 브레제남 알고리즘과 본 논문에서 제안한 방식간 실행 속도 차를 비교한 결과를 나타내었다. <표 4>의 실행 시간과 차이가 나는 것은 1/1000초 단위 측정을 위해 sleep 명령을 사용했기 때문이다. 픽셀 개수가 많아짐에 따라 실행 시간차는 다소 줄어드는 것을 알 수 있으나, 2~5%의 실행 속도 개선이 이루어짐을 확인할 수 있었다.



(그림 8) 원 브레제남 알고리즘을 적용하여 생성된 직선들



(그림 9) 수정 브레제남 알고리즘을 적용하여 생성된 직선들

6. 결 론

한 직선 선분은 선분의 중간 점에 대하여 대칭의 성질을 가진다. 우리는 이 대칭성에 착안하여 직선의 생성을 위한 픽셀의 선택 시 한번에 두 개의 픽셀을 선택할 수 있는 수정된 브레제남 직선 생성 알고리즘을 제시하였다. 직선을 그릴 때 선분의 양 끝점에서 동시에 중점을 향해 움직이면서 한꺼번에 대칭 관계에 있는 두 개의 픽셀을 선택하게 하였다. 이 때 두 픽셀의 선택을 위해 각각의 결정 파라미터를 사용하지 않고 한 개의 파라미터만을 사용하였다. 즉 $p_k<0$이어서 왼쪽에서 y 좌표가 그대로 유지된다면 오른쪽에서도 역시 y좌표가 그대로 유지된다. $p_k \geq 0$ 이어서 왼쪽에서 상향된 y좌표의 픽셀이 선택되었다면 오른쪽에서는 이에 대응하여 하향 y 좌표의 픽셀이 선택되는 것이다. 이러한 선택 방법을 뒷받침하기 위한 이론적 증명을 제시하였다. 결론적으로 본 논문에서 제안한 수정 브레제남 알고리즘은 픽셀 선택과 관련된 루프의 반복 실행 회수가 약 1/2로 줄어들기 때문에 직선을 그리기 위한 시간을 단축할 수 있다.

순수하게 픽셀 위치 계산에 걸리는 시간만 측정해 본 결과 본 논문에서 제안한 알고리즘의 실행 속도는 픽셀 개수가 100만~800만개 범위일 때 원 브레제남 알고리즘의 15%에 불과함을 알 수 있었다. 또한, 수정 알고리즘은 기울기 변동에 상관없이 일정한 실행 속도를 유지함을 알 수 있었다. 한편, 윈도우에 선분을 그리기 위한 그래픽 함수 호출에 걸리는 시간 등 제반 시간을 모두 포함시킨 성능 측정 결과, 최대 100만개의 픽셀을 갖는 직선 선분에 적용했을 때 원 브레제남 알고리즘에 비해 2~5%의 속도 개선 효과를 얻음을 확인할 수 있었다. 한편, 생성된 선분 모양은 원 브레제남 알고리즘과 전혀 차이가 없음도 확인하였다.

참 고 문 헌

[1] J. D. Foley , A. Vandam , "Fundamentals of Interactive Computer Graphics," Addition Wesley, pp.432-436, 1984.

[2] Cornel K. Pokorny , Curtis F. Gerald , "Computer Graphics : The Principles behind the Art and Science," Franklin, Beedle and Associates, pp.31-54, 1989.

[3] Brons r. "Theoretical and Linguistic Methods for Describing Straight Lines." In Fundamental Algorithm for Computer Graphics, Nato ASI Series, Vol. F17, New York : Spring Velag, 1985.

[4] Castle CMA. Pitteway MLV : An Application of Euclid's Algorithm to Drawing Straight Lines." In Fundamental Algorithms for Computer Graphics, Nato ASI Series, Vol F17, New York : Spring Velag, 1985.

[5] P. E. Danielsson, "Incremental Curve Generation," IEEE Trans, Comput, Vol.C-19, pp.783-793, 1970.

[6] B. W. Jordan Jr. et al., "An Improved Algorithm for the Generation of Non-parametric Curves," IEEE Trans. Comput., Vol.C-22, No.12 pp.1052-1060, 1973.

[7] T. Kamae, M. Kosugi, and T. Hoshino, "Dot Display of Simple Graphics" (in Japanese), Trans. I. E. C. E. Japan, Vol.56-A, No.7 pp.401-408, 1973.

[8] Bresenham, J. E, "Algorithm for Computer Control of a Digital Plotter," IBM Syst. J. 4.1, pp 26 30. 1965.

[9] Robert F. Sproull , "Using Program Transformations to Derive Line-Drawing Algorithms," ACM Trans. on Graphics. Vol.1, No.4, pp.259 273, 1982.

[10] David F. Rogers, "Procedural Elements for Computer Graphics", McGraw-Hill,Inc., pp.29-42. 1985.

[11] Yasuhito Suenaga, Takahiko Kamae, Tomonori Kobayashi, "A High-Speed Algorithm for the Generation of Straight Lines and Circular Arcs," IEEE Trans. on Computers, Vol.C 28, No.10, 1979.

[12] Donald Hearn, M.Pauline Baker, "Computer Graphics": 2nd edition, Prentice Hall,1997.

이 상 락

e-mail : srlee@lion.inchon.ac.kr

1971년 서울대학교 과학교육과 졸업(이학사)

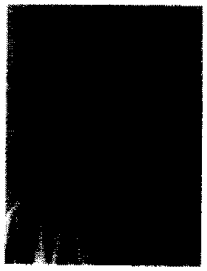
1984년 광운대학교 대학원 전자계산학과 졸업(이학석사)

1995년 광운대학교 대학원 전자계산학과 졸업(이학박사)

1983년~1988년 한국교육개발원 연구원

1988년~현재 인천대학교 조교수

관심분야 : 컴퓨터그래픽스의 기하모델링, 애니메이션 등



홍 윤 식

e-mail : yshong@lion.inchon.ac.kr

1983년 한양대학교 전자공학과 졸업(학사)

1985년 한국과학기술원 전기 및 전자공학과 졸업(공학석사)

1989년 한국과학기술원 전기 및 전자공학과 졸업(공학박사)

1989년~1991년 LG반도체(주) 선임연구원

1991년~현재 인천대학교 전자계산학과 부교수

1997년~현재 LG정보통신(주) 단말연구소 S/W개발단 기술자문교수

1994년~1995년 U. C. Irvine visiting scholar

관심분야 : High-Level Synthesis, HCI, mobile computing