A Fast Image Scaling Method Using Bresenham's Line Drawing Algorithm

김진홍

1. 서론

게임이나 이미지 관련 응용 프로그램들은 자유로운 배율의 이미지 스케일링 기능을 필요로 한다. 게다가 멀티미디어 장비들의 발전으로, 상용되는 이미지들의 해상도가 급격이 높아지고 있으며, 이러한 이미지들에 대해 실시간 스케일링처리가 요구되어질 때도 있다.

특히, 만화영화제작시 컴포지션(다양한 2D 이미지들을 혼합하여 한 장면을 구성하는것) 단계에서 가상 카메라의 이동으로 고 해상도(1000픽셀 이상) 화면의 스케일링이 요구되는데, 스케일링 부분을 고속 처리하여 컴포지션의 처리 시간을 단축하면 고 효율의 작업 처리가 가능해진다. 그리고 실시간 처리로 스케일링 작업시 보다 인터랙티브해질 수 있다. 또한, 근래 게임들은 상황에 따라 전체 게임 화면의 이미지가 자유롭게 스케일링될 필요가 있는데, 이 경우에도 스케일링의 실시간 처리 능력이 요구된다.

우리는 이미지를 스케일링하기 위해 일반적으로 방정식을 이용한다. 하지만 일반 방정식을 이용해 2D 이미지 스케일링을 실현하면 필연적으로 요구되는 부동소수점 연산 때문에 FPU(Floating Point Unit)가 없거나 FPU를 에뮬레이션 해야하는 환경에서는 실시간 처리에 부적합할 정도로 속도가 느려져 실용 가치가 떨어진다. 이것을 보완하기 위해 여러 가지 방법들이 개발 적용되고 있겠지만 본 연구에서는 선 그리기 알고리즘인 Bresenham 알고리즘을 이용하여 스케일링시 속도저하에 가장 큰 요인이 되는 부동소수점 연산을 제거함으로써 실시간에 적합한 성능향상을 기대해 본다.

2. 스케일링 기본 방법

스케일링은 개념적으로 간단하다. 여기서 1차원으로 나열된 픽셀들을 예로, 스케일링에 대해 줌-인(Zoom-In)의 경우만 간단히 짚고 넘어가자.

원본이미지의 각 픽셀 원소를 배율에 맞도록 반복 출력하면 확대가 된다. 즉, 배율이 2배이면 각 점마다 수평 수직으로 두 개씩 반복 출력하면 두배 확대된 이미지가 만들어진다.

오직 정수의 배율만 사용한다면 원본이미지의 픽셀을 단순 반복 처리할 수 있지만, 실수배율이 요구될 시에는 원본이미지의 각 픽셀의 반복처리수가 다를 수 있으므로 각 픽셀마다 적당한 반복횟수를 결정해야하는 문제가 발생한다. 그러나 보통 반복횟수를 구하는 대신 원

본 픽셀 좌표들을, 확대 후 생겨날 복사본 픽셀 좌표에 매핑 시키는 방법을 사용한다. 매핑 시에는 직선의 방정식과 유사한 간단한 방정식을 사용한다.

S : x pixel-coordinate of a source image
D : x pixel-coordinate of a destination image
* S, D는 정수

$$S = (int)\frac{1}{\text{배율}} * D$$

< 매핑함수 >

위의 과정을 2차원적으로 실행하면 2D이미지를 확대하는 솔루션이 된다. 위의 방법은 상당히 간단하고 연산이 정밀하지만 부동소수점 연산을 요구하기 때문에 컴퓨터 상에서는 잠재적으로 느릴 수 밖에 없다.

3. Bresenham알고리즘의 소개

3.1 목적

현재 우리가 사용하는 Graphics System은 거의 대부분이 픽셀들을 나열하여 화면을 구성하는 Raster Screen이다. 이러한 시스템에서의 line은, 임의의 두 점을 수학적인 직선으로 연결했을 때 그 선에 가장 가까이 위치한 픽셀들을 선택해 칠함으로써 구현된다. 그 픽셀들을 선택하는데 가장 일반적인 방법은 직선의 방정식 y = ax + b 를 이용해 얻어진 값을 반올림이나 버림 해 이용하는 것이다. 하지만 기울기에 해당하는 a값 때문에 부동소수점 연산이 필요하게 되므로 선이 구현되는 속도가 느려진다. Bresenham 알고리즘에서는 그 직선의 방정식을 변형해 정수연산만을 포함하도록 함으로써 컴퓨터에서 잠재적인 부동소수점처리속도의 문제를 제거해 성능 향상을 이룬다.

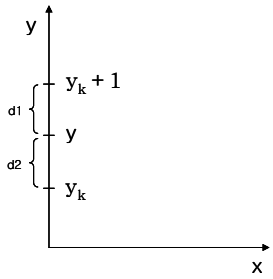
3.2 Bresenham 알고리즘

직선의 방정식을 이용한 x_k다음점의 y위치는 $y = m(x_k+1) + b$ 이다.

선을 그을 두 지점을 x1, y1, x2, y2값으로 표현하면

$m = \frac{(y2-y1)}{(x2-x1)}$, 즉 $m = \frac{\triangle y}{\triangle x}$ 이다.

위 공식에 의해 x_k에 해당하는 y의 값을 계산하면 실수가 나올 수 있고 그렇게 되면 <그림 1>과 같이 y_k와 y_k+1 두 점 사이에서 둘 중 하나를 선택해야 하는 문제가 발생한다.



<그림 1>

Bresenham 알고리즘의 기본 아이디어는 실제 계산된 y와 y_k 그리고 y와 y_k+1 사이의 거리를 비교해 더 짧은 쪽의 픽셀을 선택하는 것이다.
y와 y_k의 거리를 d1, y와 y_k+1의 거리를 d2라고 하면;
$d1 = y - y_k$
$d2 = y_k+1 - y$ 이다.

만일 d1 - d2 에서 음수가 나오면 d1이 더 작은 수이므로 y_k가 더 가깝다는 의미이므로 y_k를 선택하고, 양수면 y_k+1을 선택한다. 여기서 이 부호를 판정하는 식을 P_k라 하자.
$P_k = d1-d2 = 2m(x_k + 1) - 2y_k + 2b - 1$
P_k에 △x를 곱함으로써 기울기 m을 나눗셈이 아닌 정수 곱셈의 형태로 변형한다. 바로 이 부분에서 실수계산이 사라진다! (△x는 양수이므로 부호에는 영향이 없다) 그러면,
$P_k = 2\triangle y \cdot x_k - 2\triangle x \cdot y_k + (2\triangle y + 2\triangle x \cdot b - \triangle x)$
위의 밑줄 친 부분은 계산상 사라지게 되는 상수이므로 c라 놓고 정리하면,
$P_k = 2\triangle y \cdot x_k - 2\triangle x \cdot y_k + c$ 가 된다.
그리고 다음 점의 부호는 $P_{k+1} = 2\triangle y \cdot x_{k+1} - 2\triangle x \cdot y_{k+1} + c$ 이며 recursive하게 P_{k+1}을 구하면, $P_{k+1} - P_k = 2\triangle y \cdot x_{k+1} - 2\triangle x \cdot y_{k+1} + c - (2\triangle y \cdot x_k - 2\triangle x \cdot y_k + c)$ 이고, 정리하

면 $P_{k+1} = P_k + 2\triangle y(x_{k+1} - x_k) - 2\triangle x(y_{k+1} - y_k)$ 이 된다.

여기서 $x_{k+1} - x_k$ 의 값은 항상 1이 되므로 결국, $P_{k+1} = P_k + 2\triangle y - 2\triangle x(y_{k+1} - y_k)$ 이다.

우리는 이 P_{k+1}이라는 부호 판정용 식을 이용해 다음 x_k위치에 해당하는 y_k위치를 찾아내는 것이다.

P_{k+1}이 음수이면 다음 y의 위치(y_{k+1})가 y_k+1보다 y_k에 더 가깝다는 의미이므로 바로 전의 y_k 값을 그대로 유지한다. 그렇게 되면 y_{k+1}이 y_k와 같게 되므로 $y_{k+1} - y_k = 0$이 되어 다음 점의 부호 판정값 P_{k+1}은, $P_{k+1} = P_k + 2\triangle y$ 이 된다.

P_{k+1}의 결과 값이 양수이면 y_k에 더 가깝다는 의미이므로 y_k+1을 선택하고, y_{k+1}과 y_k값의 차이가 1이 되므로 부호 판정값 P_{k+1}은, $P_{k+1} = P_k + 2\triangle y - 2\triangle x$ 이 된다.

마지막으로 P_k의 초기 값은, $P_0 = 2\triangle y - \triangle x$ 가 된다.

지금까지의 수식으로 선 그리기를 수행하는 과정은 아래와 같다.

1. P_0를 구한다.

2. $P_k < 0$이면 다음 점은 x_k+1, y_k이고 $P_{k+1} = P_k + 2\triangle y$,
   아니면 다음 점은 x_k+1, y_k+1이고 $P_{k+1} = P_k + 2\triangle y - 2\triangle x$ 이다.

3. 2번 과정을 $\triangle x$번 반복한다.

3.3 스케일링 적용

실수배 확대 시에는 전에 언급했던 바와 같이 원본이미지의 각 점을 몇 번씩 반복 처리해야하는지 결정해야 하는 문제가 발생한다. 여기서는 '원본이미지의 각 점을 몇 번씩 반복 출력해야 하는가' 라는 문제를, '원본이미지의 처음 점을 선택해 반복 출력하다가 언제 원본이미지의 다음 점을 선택해야 하는가'라는 것으로 생각을 바꾸었다. Bresenham 알고리즘은 다음 점을 선택하는 조건을 마련해 주므로 이 주제에 적용 가능하다.

직선의 방정식은 상수 b를 제거하면 위의 <매핑함수>와 유사한 것을 알 수 있다. 직선의방정식에서 y는 <매핑함수>의 S와 같고, 기울기 a는 $\frac{1}{\text{배율}}$ 과 같으며 그에 곱해지는 x는 D와 같다. 여기서 $\frac{1}{\text{배율}}$은, 원본 이미지의 폭을 sw라 하고 확대 후 이미지의 폭을 dw라 하

면 $\frac{sw}{dw}$ 라고 정의할 수 있다. 여기서 배율을 사용자가 직접 정의하는 것보다, 원본 이미지의 크기와 확대 후의 이미지의 폭을 사용해 정의하면 실제 사용하기가 훨씬 편리하다. 그러므로

S = (int) $\frac{1}{\text{배율}}$ * D 을 y = $\frac{sw}{dw}$ x 라 할 수 있다.

위 식을 이용해 다음 y위치를 구하면, y = $\frac{sw}{dw}$ (x_k + 1) 이다.

Bresenham알고리즘과 같이 P_k, P_{k+1}, P_0를 구하면,

$P_k = 2sw \cdot x_k - 2dw \cdot y_k + c$
$P_{k+1} = P_k + 2sw - 2dw(y_{k+1} - y_k)$
$P_0 = 2sw - dw$ 이다.

확대를 수행하는 과정은 Bresenham 알고리즘과 같고, 이것을 2차원적으로 적용하면 이미지 확대 알고리즘이 완성된다.