한국시뮬레이션학회 '99추계학술대회 논문집 1999.10.9 한양대학교

실시간 시뮬레이션을 위한 검색 기법들의 속도 연구

윤 석 준†, 강 현 주†

A Study on Real-time Speed of Searching Algorithms

Sugjoon Yoon, Hyounjoo Kang

Abstract

실시간 시뮬레이션에서 주어지는 다양한 종류의 불연속적인 파라미터 값을 가지는 데이터 테이블에서 실시간의 제약 하에서의 검색을 수행하기 위해서는 최적의 기법이 요구된다. 실시간의 제약 하에서 최적 기법의 기준이 되는 것은 보통의 알고리즘들과는 달리 평균 속도가 아니라 worst case에서의 속도가 된다. 검색 알고리즘 들은 iteration을 거치게 되므로 총 탐색에 걸리는 시간은 iteration의 수(logical speed)와 1 probe를 수행하는 데 걸리는 시간(실제 수행 속도)의 곱으로 정의된다. 본 연구에서 총 탐색에 걸리는 시간을 이론적으로 계산한 검색 속도는 기존의 수행한 수치비교시험의 결과와 대체로 일치하였고, 이분 검색법이 iteration의 수와 실제 수행시간 모두에 있어서 가장 우수하다. 한편, 검색하고자 하는 파라미터 값의 dynamics를 이용하여 주어진 데이터 테이블 내의 검색 영역을 축소시키는 dynamic-window 개념을 도입하여 검색 알고리즘의 속도를 향상시킬 수 있었다. 이 개념의 도입은 데이터 테이블의 형태에 민감한 보간 검색법(interpolation method)과 그 응용 기법들에 대해 탁월한 효과를 나타내었다. 결론적으로 일반적 데이터 테이블에 있어서는 이분 검색법이 logical speed와 실제 수행속도가 우수하고, dynamic-window개념을 도입한 보간 검색법과 그 변형들은 logical speed가 탁월하게 향상된다.

1. 서 론

동체의 시스템 또는 운동에 대한 시뮬레이션에서는 다양한 종류의 불연속적인 파라미터 값들이 데이터 테이블의 형태로 주어지게 되며, 매 적분 스텝마다 이 데이터들을 이용하여 필요한 함수 값들을 산출하기 위한 함수발생(function generation)[1] 작업들이 수행된다. 적분 스텝의 크기가 고정되는 일반적인 실시간 시뮬레이션의 제약 하에서 함수발생 과정을 수행하기 위해서는 최적의 검색 기법이 요구된다.

본 연구에서는 기존에 사용되어 왔거나 새롭게 연구되고 있는 검색 알고리즘들의 이론적·시험적 속도를 비교함으로써 실시간성을 갖는 최적

의 검색 기법을 찾는 것이다.

실시간 시뮬레이션에서 사용되는 대부분의 데이터들은 불연속적이면서 등 간격이 아니다. 또한 검색하게 되는 파라미터 값들이 데이터 테이블 내에 바로 존재하지 않는다. 따라서, 불연속적으로 저장된 키(key) 값들이 아니라 파라미터 값을 포함하는 단위 구간에 대한 검색을 수행해야 한다. 실시간 시뮬레이션에서는 시뮬레이션 시작 이전에 off-line으로 데이터의 정렬이 가능하기 때문에 정렬된 데이터를 대상으로 하는 검색 방법들이 본 연구의 비교 대상이 된다. 이용한 조건들에 적합한 검색 알고리즘들이 이분 검색(bisection search)[2], 보간 검색(interpolation search)[3], fast search[3], modified regular falsi 등이며, 실시간 시뮬레이션 특히 HILS(Hardware-in-the-Loop Simulation) 분야에서 사용되어왔다.

앞에서 언급한 알고리즘들에 관해 수치시험을 통한 최적의 기법[9]은 이미 밝혀진 바 있으나, 특정 data set에 대한 시험적 비교만으로는 이를 최적의 기법이라 단언할 수없기 때문에 이론적인 알고리즘의 속도연구를 통해 이를 재검토 하고자 한다.

일반적으로 알고리즘들을 선택하는 기준으로는 이해와 코딩, 오류 정정의 용이와 빠른 수행 속도를 들 수 있지만, 지금부터 검토하고자 하는 알고리즘들에 대해서는 실시간성이 가정 우선적인 제약 조건이 된다. 따라서 수행속도가 알고리즘 선택의 첫 번째 기준이 된다.

알고리즘의 속도를 나타내는 방법에는 여러 가지가 있지만, 평균속도보다 worst case에서의 속도가 실시간성의 제약조건을 만족시키기 위한 기준이 되기 때문에 수행시간의 상한을 나타내는 $Big\,O$ 계산값을 각 알고리즘의 속도로 정의할 수 있다. 검색 알고리즘의 경우에는 $Big\,O$ 계산값이 검색에 필요한 iteration의 수를 나타내게 된다.

또한 검색에 필요한 iteration의 수인 logical speed가 아니라, 알고리즘들을 하드웨어 상에서 실행시켰을 때 검색을 마치는 실제 수행시간도 알고리즘을 선택하는 중요한 기준이 된다.

이분 검색법이 n개의 데이터에 대한 worst case에서의 $Big\,O$ 계산값으로 $O(\log_2 n)$을 갖으며, 이는 수치시험 결과와 마찬가지로 검토 대상인 알고리즘들 중 가장 빠르다. 실제 수행시간에 대해서도 단순한 알고리즘을 가진 이분 검색법이 다른 검색법들에 비해 3배정도 빠르게 나타났다.

한편, 검색하고자 하는 파라미터 값이 갖고있는 dynamics를 이용하여 주어진 데이터 테이블 내의 검색 영역을 축소, 직선화 시키는 기법을 기존의 4가지 검색 알고리즘에 결합시켜 검색 속도를 향상시킬 수 있다. 검색 알고리즘들 중 보간 검색법과 이를 응용한 방법들이 직선형 데이터에 대해 빠른 검색 속도를 보이므로, Dynamic-window[7] 개념을 도입함으로써 탁월한 효과를 얻을 수 있다.

2. 실시간 시뮬레이션의 제약

실시간 시뮬레이션에서 사용되는 대부분의 데이터들은 불연속적이면서 등 간격이 아니다. 또한 시뮬레이션 시작 이전에 off- line으로 데이터의 정렬이 가능하다. 따라서 불연속적이며 간격

이 불규칙적인 정렬된 데이터 테이블이 검색 대상이 된다.

검색하게 되는 파라미터 값들 대부분이 주어진 데이터 테이블 내에 바로 존재하지 않게 된다. 따라서 키 값이 아니라 단위 구간에 대한 검색을 수행해야 한다. 실시간 시뮬레이션에서는 적분 스텝의 크기가 고정되어 있기 때문에 정해진 시간 내에 검색을 수행해야 하며, 초기값에 상관없이 검색이 보장되어야 한다.

3. 검색 알고리즘

본 절에서는 n개의 data를 가진 1차원 데이터 테이블을 대상으로, 기존의 검색 알고리즘들인 이분 검색법, 보간 검색법, fast search와 수치해석에서 대수방정식의 근을 구하는 방식의 변형 기법인 modified regular falsi 기법의 알고리즘들을 소개한다.

Bisection method(이분 검색법)

이분 검색은 원하는 파라미터 값을 포함하는 단위 구간을 찾을 때까지 데이터 테이블의 크기를 되풀이하여 이등분하는 방식이다. 다음은 n개의 데이터에 대한 알고리즘으로 i번째 데이터를 $D(i)$로 표시하며 구하는 파라미터 값을 N으로 표시한다:

Algorithm

$l=1,\ r=n$

For $j=0,1,2,\cdots$ *until satisfied, do*:

Set $m=(l+r)/2$

If $D(m+1) < N$, *set* $l=m+1$

Else if $D(m) > N$, *set* $r=m$

Otherwise 'N' *exist in the inteval* $[l, r]$

Interpolation method (보간 검색법)

보간 검색은 대수방정식의 근을 구하는 수치해석 기법인 Regular falsi의 변형으로 주어진 데이터 테이블 내의 파라미터 값들이 선형적으로 분포됨을 가정하여 양 단을 잇는 직선상에서 파라미터 값을 포함하는 단위 구간을 검색하는 방법이다. 데이터 테이블의 형태에 따라 검색속도 차이가 심하게 나타나, 직선형의 경우 탁월한 속도를 보이지만 최악의 경우는 순차 검색을 하게 된다.

Algorithm

$l=1,\ r=n$

For $j=0,1,2,\cdots$ *until satisfied, do*:

Set $m=\dfrac{(N-D(L))*(r-l)}{(D(r)-D(l))}+1$

If $D(m+1) < N$, *set* $l=m+1$

Else if $D(m) > N$, *set* $r=m$

Otherwise 'N' *exist in the inteval* $[l, r]$

Fast Search

Fast Search는 보간 검색법의 변형으로, 구간의 크기를 정하는 임의의 수를 이용하여 각 loop에서의 구간을 줄여 키를 포함하는 단위 구간에 대한 검색속도를 증가시키는 방법이다. 최악의 경우 보간 검색법은 순차 검색을 하기도 하는데 Fast Search는 이를 회피할 수 있다. 따라서 최악의 경우에도 검색 속도가 보간 검색법보다 빠르다.

Algorithm

$l=1,\ r=n$

For $j=0,1,2,\cdots$ *until satisfied, do*:

Set $m=\dfrac{(N-D(L))*(r-l)}{(D(r)-D(l))}+1$

If $D(m+1) < N$, *set* $l=m+1$

If also $|m+1, r-1| < |m, l+1|$,

set $r = r+2$

Else set $r = r+1$

Else if $D(m) > N$, *set* $r = m$

If also $|m+1, r-1| < |m, l+1|$,

set $l = l+2$

Else set $l = l+1$

Otherwise 'N' exist in the inteval $[l, r]$

Modified regular falsi

Modified regular falsi 방식은 수치해석에서 대수방정식의 근을 찾는 방법으로 이용되어 왔다. 그 원리는 보간 검색법의 변형으로 구간의 양 끝 점 중 하나의 크기를 반으로 줄여서 키를 포함하는 구간의 크기를 좁혀 검색 속도를 증가시키는 방법이다.

Algorithm

$l = 1$, $r = n$, $L = D(1)$, $R = D(n)$

For $j = 0, 1, 2, \cdots$ *until satisfied, do*:

Set $m = \frac{(N - D(L)) * (r - l)}{(D(r) - D(l))} + 1$

If $D(m+1) < N$, *set* $l = m+1$ and

$L = D(m+1)$, $R = (D(r) + N)/2$

Else if $D(m) > N$, *set* $r = m$ and

$R = D(m)$, $L = (D(l) + N)/2$

Otherwise 'N' exist in the inteval $[l, r]$

Dynamic-window Search

검색 기법에 대한 연구로는 원래의 주어진 데이터 테이블에서 검색 대상이 되는 영역을 축소함으로써 검색 속도를 향상시키기 위한 dynamic-window search[7]도 있다. 검색하는 데이터가 임의의 dynamics를 갖고 매 적분 프레임마다 변한다면, 이 정보를 활용하여 주어진 데이터 테이블 중 일부의 영역으로 검색 대상을 축소할 수 있을 것이다.

4. Worst case에서의 검색 속도

앞서 실시간 시뮬레이션에 주어지는 데이터 테이블의 특성에 대해 언급한 바 있다. 이런 데이터 테이블의 검색에 보통의 검색 알고리즘을 이용하면 2개의 키를 찾아야 하는데, 실시간 시뮬레이션에서 이러한 알고리즘들은 검색 속도가 느려져 유용하지 못하다. 이러한 조건에 맞추기 위해서는 구간, 즉 2개의 키를 하나의 단위로 생각하여 각 검 알고리즘을 수정해야 한다.

앞에서 실시간 시뮬레이션에 주어지는 데이터 테이블에 적합한 수정된 4가지 검색 알고리즘들(bisection, interpolation, fast, modified regular falsi)을 소개했다. 지금부터는 이 알고리즘들의 속도를 비교, 계산함으로써 실시간성이 만족되는 최적의 기법을 찾도록 하겠다.

특정 다수의 data set을 대상으로 한 수치시험을 통해 검색 알고리즘을 선택하는 것은 불합리적일 뿐만 아니라 시간적, 비용적 면에서 불가능하다. 따라서 가능한 모든 형태의 data set을 고려하여 이론적으로 알고리즘을 선택하는 방법과 근거를 제시하고자 한다.

알고리즘을 선택하는 기준은 크게 다음의 2가지로 나눌 수 있다.

1. 이해와 코딩, 오류 정정의 용이
2. 수행속도가 빠른 알고리즘

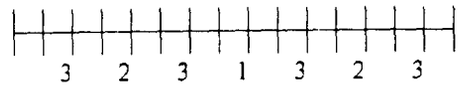
프로그래머가 사용하는 시간적 비용이 수행하는 비용을 훨씬 초과하는 경우가 대부분이기 때문에 보통의 프로그램을 작성하는데 있어서는 1번째

조건을 가장 중요하게 생각하게 된다. 그러나 앞에서 검토한 4가지 검색 알고리즘들은 logic이 단순하므로 1번째 조건의 제약은 받지 않는다. 반면 실시간 시뮬레이션의 제약중 하나인 실시간성을 만족시키기 위해서는 2번째 조건인 수행속도가 검색 방법을 선택하는 최우선의 기준이 된다. 또한 적분 스텝이 고정되어 있기 때문에 모든 경우의 평균이 아니라 worst case에서의 속도로 비교해야만 한다.

알고리즘의 속도는 나타내는 방법은 매우 다양하다. 입력의 크기에 따른 수행시간 $T(n)$[10], $T(n)$의 상한을 나타내는 O-표기법 ($Big\ O$)[10]과 하한값을 나타내는 Ω-표기법 ($Omega$)[10] 등이 있다. 앞에서도 언급했듯이 실시간 시뮬레이션에는 제한된 시간 내에 검색결과를 내야 하기 때문에 알고리즘을 선택하기 위해서는 알고리즘의 평균속도나 하한속도가 아니라 worst case에서의 상한속도 ($Big\ O$)가 기준이 된다. $Big\ O$의 계산값은 알고리즘을 수행하는데 필요한 최대한의 연산식의 개수로 정의 되며, 검색 알고리즘에 있어서는 검색에 필요한 iteration의 수가 된다. 다음은 각 검색 방법에 대한 $Big\ O$를 계산한 것이다.

Bisection method(이분 검색법)

Bisection search는 데이터의 형태와는 무관하게 데이터 개수에만 영향을 받게 된다. 앞에서 언급한 것과 같이 구간을 단위로 검색을 행한다면, n개의 데이터를 가진 경우에는 $n-1$개의 구간이 생기고, 각 iteration마다 구간의 수가 반으로 준다. 각 구간에 포함된 수를 검색하기 위한 iteration의 수를 생각해 보면 1번에 찾는 경우는 1개구간이고, 2번에 찾는 경우는 2개구간이 되는데, 이를 일반화시키면 k번에 찾는 경우는 2^{k-1}개의 구간이 된다. <그림1>은 각 구간에 포함된 수를 검색하는데 필요한 iteration의 수를 나타낸다.



<그림1> 이분검색에 필요한 iteration의 수

또 각각의 iteration에서는 1번의 계산만이 실행되므로, 앞서 설명했듯이 iteration의 수가 O-표기법 ($Big\ O$)으로 사용된다. 이를 이용해 worst case의 iteration의 수를 다음과 같이 계산 할 수 있다.

k가 worst case의 iteration의 수라면, 1번부터 k번까지 찾게되는 데이터의 수가 $n-1$이 되어야 하기 때문에 다음 식을 얻는다.

$$\sum_{i=1}^{k} 2^{i-1} = n-1$$

위의 식의 k에 관해 풀어보면,

$$\sum_{i=1}^{k} 2^{i-1} = \frac{1(1-2^k)}{1-2} = n-1$$

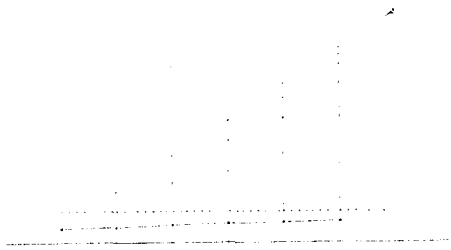
$$\rightarrow \quad 2^k = n$$

$$\therefore\ k = \log_2 n$$

그러나 iteration의 수는 정수가 되어야 하므로 정확하게는 $\log_2 n$의 올림값이 worst case에서의 iteration 수가 된다. 이를 O-표기법($Big\ O$)으로는 $O(\log_2 n)$으로 표현한다.

Interpolation method(보간 검색법)

Interpolation search는 데이터의 형태에 많은 영향을 받게 된다. 데이터의 두 점을 잇는 직선을 이용하여 검색하므로 데이터의 형태가 직선형일 때에는 검색속도가 매우 빠르지만, <그림2>와 같이 최악의 경우 순차검색을 하게된다.

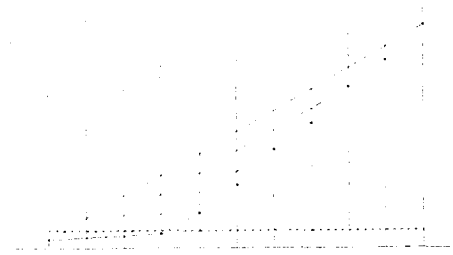


<그림2> 보간검색의 worst case

Interpolation search도 bisection search와 마찬가지로 각각의 iteration마다 1번의 계산을 하게되므로 iteration의 수를 계산하면 된다. 따라서 데이터 형태가 worst case인 경우에 검색 대상이 되는 구간의 크기가 1씩 감소하므로 검색하는데는 데이터의 개수만큼의 iteration이 필요하다. 이를 *O*-표기법(*Big O*)으로는 $O(n)$으로 표현한다.

Fast method

Fast search는 interpolation search를 보완한 방법이다. 두 점을 잇는 직선에 의해 찾게되는 한 점과 고정된 점의 *index*를 줄이는 방법으로 속도를 향상시키기 위해 그림과 같이 각 iteration 마다 구간의 수가 줄어드는 효과를 얻을 수 있다. <그림3>은 worst case의 예이다.



<그림3> fast search의 worst case

Worst case의 경우 1번의 iteration으로 3개의 구간을 줄일 수 있다. 따라서 전체 data의 개수가 n이라면 다음과 같이 iteration의 수를 구할 수 있다.

$$n-3\times i=0 \quad \rightarrow \quad i=n/3$$

위의 방법은 단위구간의 크기를 고정시키는 방법이고, 기존의 fast search는 단위구간의 크기를 매 probe 마다 바꾸어 줌으로써 구간의 크기를 평균 n에서 $\sqrt{n}$으로 줄여가면서 검색을 하게 된다. 구간의 크기는 $n^{\frac{1}{2^0}} \rightarrow n^{\frac{1}{2^1}} \rightarrow n^{\frac{1}{2^2}} \rightarrow n^{\frac{1}{2^3}}$와 같은 형태로 줄어들게 된다. 기존의 방법의 속도는 다음과 같이 계산할 수 있다.

$$n^{\frac{1}{2^k}}=2 \rightarrow \frac{1}{2^k}\log_2 n=1$$

$$\rightarrow \frac{1}{2^k}=\frac{1}{\log_2 n} \rightarrow 2^k=\log_2 n$$

$$\therefore k=\log_2 \log_2 n$$

또한 worst case에서는 구간의 크기가 n에서 $\sqrt[k]{n}$으로 감소하기 때문에 다음의 계산과정을 통해 $O(\log_2 n)^2$을 계산할 수 있다.

$$n^{k^{\frac{1}{2}}} = 2 \rightarrow k^{-\frac{1}{2}} \cdot \log_2 n = 1$$

$$\rightarrow k^{-\frac{1}{2}} = \frac{1}{\log_2 n} \rightarrow k^{\frac{1}{2}} = \log_2 n$$

$$\therefore k = (\log_2 n)^2$$

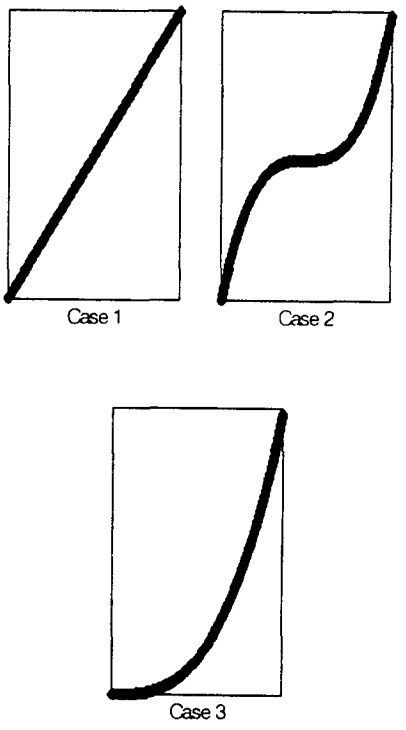
Modified regular falsi

Modified regular falsi를 이용한 검색도 interpolation search를 보완한 방법이다. 따라서 두 점을 잇는 직선을 이용하게 되는데 두 점 중 한 점의 값을 줄여줌으로써 순차검색이 되는 것을 방지, 검색의 속도를 증가시키는 방법이다. 아래의 3가지 수치시험에서는 가장 우수한 검색 속도를 보이지만, 이론적인 속도(worst case)는 interpolation search와 같다. 데이터의 형태가 연속이 아니기 때문에 고정된 점의 값을 줄여주는 것의 효과가 전혀 나타나지 않기 때문이다. 다음 그림<그림4>은 그 예이다.

<그림4> modified regular falsi의 worst case

따라서 worst case인 경우에 검색하는데는 필요한 iteration의 수는 interpolation search와 같게 된다. 이를 $O(n)$으로 표현된다.

5. 수치 시험과 실제 수행시간



<그림 5> 수치시험 데이터 테이블의 유형

<그림5>와 같이 각각 1000개의 키가 등 간격으로 배치된 3종류의 데이터 테이블에 대해 이분 검색법, 보간 검색법, fast search, modified regular-falsi 등의 검색법들을 시험하였다.

<표1>에 표시된 수치들은 단위 구간을 찾는데 소요되는 총 탐색 iteration의 수를 나타내며, 작을수 록 검색속도가 빠름을 의미한다. Time은 1 probe 수행하는데 걸리는 시간(초)을 의미하기 때문에 iteration의 수와 time의 곱이 총 탐색에 걸리는 시간이 된다.

<표1>의 결과에서 나타나듯이 이분 검색법의 총 검색 iteration의 수는 데이터의 형태에 상관없이 데이터 키의 개수에만 좌우된다. 그러나 보간 검색 방법들은 데이터의 형태에 따라 많은 영향을 받게 된다. 직선형의 경우 데이터의 개수에 관계없이 빠른 속도로 검색을 하나, 굴곡이

심한 데이터의 경우 데이터의 개수에 관계없이 검색 속도가 급격히 감소한다. 또 실제 실행시간에 있어서도 이분 검색법이 기타 다른 방법들에 비해 약 3배 정도 빠르게 나타난다. 따라서 총 iteration의 수와 실제 실행 시간의 곱으로 표현되어지는 총 탐색에 걸리는 시간에 있어서 이분 검색법이 가장 빠르다.

<표1> 기존 검색 기법들의 수치시험 비교

	Bisection		Interpolation		Fast		Modified	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Case 1	9.0	10	1	1	1	1	1	1
Case 2	9.0	10	16.8	65	16.3	64	10.1	48
Case 3	9.0	10	21.8	98	21.5	97	9.9	37
Total	9.0	10	13.2	98	12.9	97	7.0	48
Time	0.00031		0.00092		0.00109		0.00109	

* Time은 1 probe를 수행하는데 걸리는 시간(초)이다.

다음으로 dynamic-window개념을 도입한 이분 검색법과 보간 검색법을 비교하였다. 싸인 함수 형태로 분포된 1000×1000개의 등 간격 키 값를 포함하는 2-D 데이터 테이블에서 200개의 파라미터 값을 검색하는데 소요되는 탐색 iteration의 수를 비교하였다.

<표2>의 결과와 같이 dynamic-window개념을 도입하면 데이터 테이블의 검색 대상 영역이 축소되는 효과를 얻을 수 있어, 이분 검색법이나 보간 검색법 모두 탐색 iteration의 수가 축소된 영역의 크기만큼 향상됨이 시험적으로도 입증되었다. 특히, 보간 검색법의 경우 검색 영역을 축소하는 효과 뿐 아니라 데이터 분포의 형태를 직선형으로 변환시킴으로써 탐색 iteration의 수를 줄여 검색 속도를 향상시키는데 탁월한 효과를 나타내었다.

결과적으로 탐색 iteration의 수가 이분 검색법에 비해 보간 검색법이 3배 정도 빠르게 되어, dynamic-window 개념의 도입하면 총 검색에 걸리는 시간이 비슷해 진다.

<표2> Dynamic-window search 기법을 도입한 검색법들의 수치시험 비교

	Bisection		Interpolation	
	Mean	Max	Mean	Max
Simple	8.95	10	6.95	13
Dynamic-window	3.16	6	1.21	2

6. 결 론

기존 검색 알고리즘들에 대한 수치시험결과에서 나타났듯이 최저(worst case) 검색 속도에 대한 이론적인 *Big O* 계산값도 이분 검색법이 가장 빠르게 나타났다. 또 알고리즘의 실제 수행시간에 있어서도 이분 검색법이 탁월하게 빨랐다.

Dynamic-window 개념을 도입시키면 검색 영역의 축소와 직선화 효과를 얻을 수 있어 검색속도를 향상시킬 수 있다. 특히 보간 검색법들이

직선형 데이터 검색에 탁월하므로 dynamic-window 개념을 도입할 경우 logical speed를 향상시키는데 보다 우수한 효과를 얻을 수 있어서 실제 수행시간을 함께 고려한 경우에도 우수하다는 결론을 내릴 수 있다.

후 기

본 연구는 과학기술부의 선도기술개발사업인 감성공학기술개발사업의 위탁과제로 수행되었으며, 지원에 감사 드립니다.

참고 문헌

[1] Kuo-Chi Lin, "The Use of Function Generation in The Real Time Simulation of Stiff Systems, The University of Michigan, 1990.

[2] 박정호, (컴퓨터)알고리즘, 상조사,1996.

[3] F. Warren Burton and Gilbert N. Lewis, A Robust Variation of Interpolation Search, Information Processing Letter, 10(4,5), pp.198-201, 5 July 1980.

[4] Gilbert N. Lewis , Nancy J. Boynton and F. Warren Burton, Expected Complexity of Fast Search with Uniformly Distributed Data, Information Processing Letter, 10(1), pp.4-7, 27 October 1981.

[5] Samuel D. Conte and Carl de Boor, Elementary Numerical Analysis an Algorithmic Approach (3th), McGraw-Hill, 1965.

[6] G. Gonnet, J. George and L. Rogers, An Algorithmic and Complexity Analysis of Interpolation Search, Acta Informatica 13, pp.39-52, Springer-Veriag 1980.

[7] Leon S. Levy, "An Improved List-Searching Algorithm, Information processing letters, 15(1), pp.43-45, 19 August 1982.

[8] Yehoshua Perl, Edward M. Reingold, "Understanding The Complexity of Interpolation Search", Information processing letters, 6(6), pp.219-222, December 1977.

[9] 윤석준, 강현주, "실시간 시뮬레이션을 위한 최적의 검색기법연구", 시뮬레이션학회 춘계 학술대회 논문집, pp.204-211, 1999.

[10] 민용식, C++ 데이터 구조론, 교우사, 1999.