アルゴリズム　17－5
(1990. 9.28)

Quad Treesに対する比較実験と考察

乗松　誠志*　　佐藤　政生**　　大附　辰夫*

＊　早稲田大学理工学部
＊＊拓殖大学工学部

あらまし　　4分木（Quad Tree）は、2次元平面におけるデータ処理に有効なデータ構造である。2分木と同様にその構造が単純なことから、地理情報処理、ＶＬＳＩのレイアウトパターンデータ管理等広い分野に応用されている。本稿では、4分木を、その管理方法の違いにより数種類に分類する。さらに、その中から4種類を選び出し計算機上に実装し、性能比較実験を行う。その結果、改良型4分木が、メモリ使用量、木構造構築時間、領域探索時間の点で最も優れていることが示される。

Experimental Comparisons among Quad Trees

Seiji Norimatsu*　Masao Sato**　Tatsuo Ohtsuki*

*School of Science and Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169, Japan
**Faculty of Engineering, Takushoku University
815-1 Tatemachi, Hachioji-shi, Tokyo 193, Japan

Abstract　Quad Tree is an effective data structure for processing two-dimensional data. Because of its simplicity, it has been applied to many fields such as geometrical information processing, VLSI pattern data management, etc. Quad Trees are classified in this paper in terms of data management method. Four of them are implemented on a computer and their performance data of experimental programs are compared. As a result, it is shown that modified quad tree is the most effective in terms of memory space requirement and the time needed for tree construction and area search.

１．はじめに

Quad Tree（以下、４分木と訳す）は、1974年にFinkel, Bentleyらによって提案されたデータ構造である[1]。４分木は、２種類の異なったキー値（２次元平面に存在する点のｘ，ｙ座標）を取り扱うためのデータ構造であり、図１の様に、領域を再帰的に４分割していくことからその名がつけられている。

この４分木を用いれば、２次元平面上の点や領域を管理することができる。図１の４分木はFinkel,Bentleyらが提案した点を管理する４分木であるが、この４分木を図２の様に領域に適用し、地理情報処理、ＶＬＳＩのパターンデータ管理等の２次元データ処理に用いることが可能である。[2]

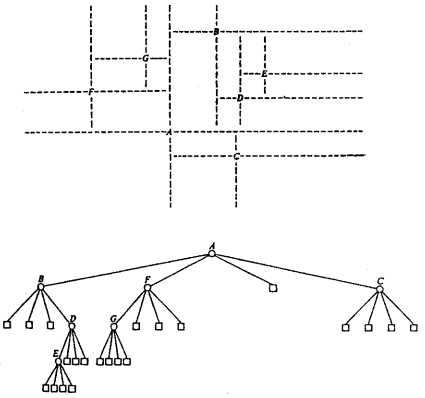


図１　Finkel,Bentleyが提案した４分木[1]

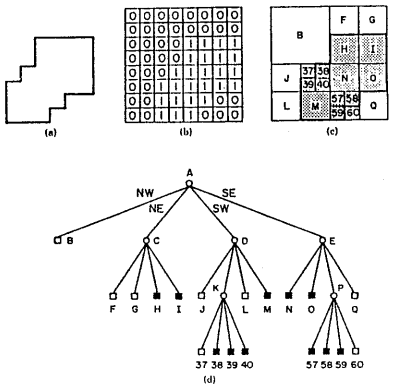


図２　４分木を用いた領域管理[2]

４分木によって長方形データを管理することは、図３の様に全てのオブジェクト（長方形データ）が分割線にかかるまで領域を再帰的に４つに均等分割していくという方法で実現される。分割線に交差するオブジェクトはノードに線形連結リストでつなぐか（これを分割線リスト型４分木と呼ぶことにする）、あるいは図４に示される様な２分木構造にしておく。

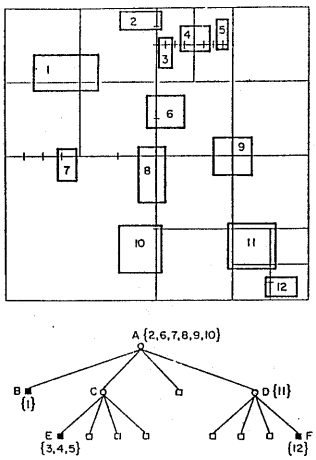


図３　４分木を用いたパターンデータ管理[2]

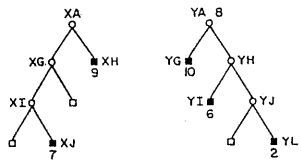


図４　ノードＡにおける２分木[2]

しかし、ＶＬＳＩのレイアウト設計などでは膨大な数のオブジェクトを扱うため、全てのオブジェクトが分割線にかかるまで分割を繰り返すのは効率的でない。そこで、分割された副領域中に存在するオブジェクトの数がある一定値以下になれば分割を停止するのが一般的である。その一定値のことをしきい値（Threshold Value）という。副領域中のオブジェクトは線形連結リストで子のノードにつながれることになる。結果としてできる４分木は適応性のある４分木（Adaptive Quad Tree）と呼ばれる。以下、４分木とは、このしきい値を導入した、適応性のある４分木であることとする。

長方形データ管理によく用いられるデータ構造として４元木（４－dimensional Tree）と呼ばれるものがある。1984年にRosenberg[3]が４元木と分割線リスト型４分木を計算機上に実装し、領域探索時間、メモリ使用量について比較実験している。その結果の詳細は割愛するが、大まかにいえばメモリ使用量では分割線リスト型４分木の方が少ないが、探索速度では４元木の方が優っている

ということであった。

また、1985年にBrown[4]が、分割線のリストを用いずに、２つ以上の副領域にまたがるオブジェクトはそのままそれらの副領域にいれてしまうという手法を提案した。これを重複保持型４分木（Multiple Storage Quad Tree）と呼ぶ。Brownは重複保持型４分木と４元木について、メモリ使用量、及び領域探索時の訪問されたノード数について比較実験を行っている。その結果、メモリ使用量は重複保持型４分木の方が少なく、探索時の訪問ノード数は探索領域（ウィンドゥ）が小さいときにはほぼ同等、ウィンドゥが大きくなると４元木の方が少ないということであった。

重複保持型４分木の致命的な欠点として、探索の際のオブジェクトの重複列挙を避けるために、２回の探索を必要とすることである。これを改善するために、1989年にWeytenら[5]が、４分リスト型４分木（Quad List Quad Tree）なるものを提案している。

Weytenらは、重複保持型４分木と４分リスト型４分木を計算機上に実装、比較実験している。その結果、メモリ使用量は４分リスト型４分木がわずかながら多いが、探索速度ははるかに４分リスト型４分木の方が優っていた。

また、1989年にPitaksanonkulら[6]が、分割線にかかるオブジェクトについて、４元木と同様に、ある基準に基づいて１つの副領域に入れてしまうという方法で４分木を構築した場合、４元木より探索速度の面で優ることを計算機実験により示した。この様にしてできた４分木を、改良型４分木（Modified Quad Tree）と呼ぶことにする。以上の分類に基づいた４分木の分類表を図５に示す。

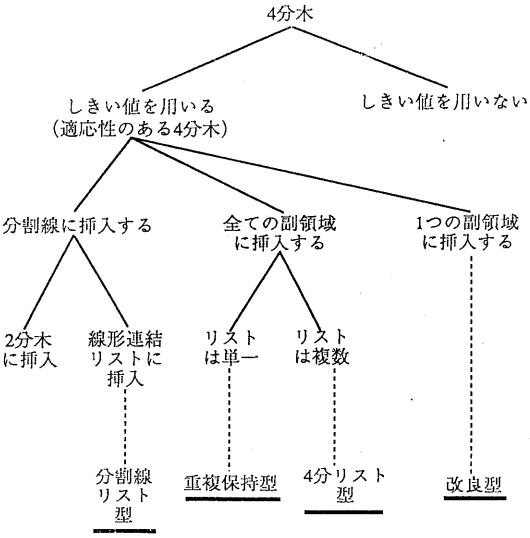


図５　　４分木の分類表

この様に、現在までに様々な手法の４分木が提案され、４元木との比較実験が行われてきた。しかしながら、４分木同士についての比較実験というのは、Weytenらが重複保持型と４分リスト型について行った以外、現在まで全く行われていない。４元木との比較のみならず、他のデータ構造との比較のためにも、４分木同士の比較を行い、４分木同士の優劣を確かめることは意義のあることだと言える。そこで、本稿では、図５に下線で示す４種類の４分木について計算機上にＣ言語で実装、メモリ使用量、木構造構築時間、領域探索について比較実験した結果について述べる。

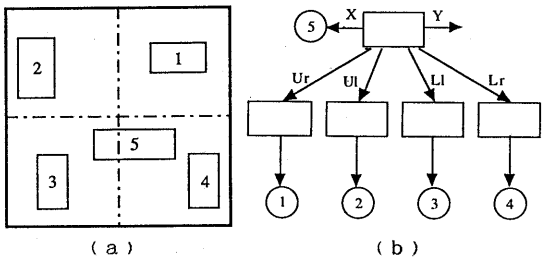
なお、本稿で扱う４分木の前提条件をここで整理しておく。

- 領域の分割の方法は、オブジェクト（パターンデータ）の分布の粗密にかかわらず一定（radix型）であるとする。
- 適応性のある４分木である。
- パターンデータの形状は長方形に限定する。長方形以外の複雑な図形を扱う場合には、通常その図形を長方形に分割するか、外接長方形で代表させる方法がとられる。本稿では、この様な前処理がすでに行なわれたものとして議論を進める。

２．各４分木の詳細

２－１　分割線リスト型４分木（ＮＯＲＭＡＬ）

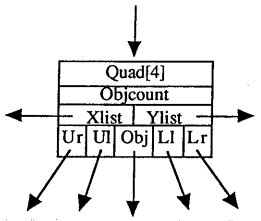
これは、分割の過程において、分割線に交差するオブジェクトはノードに連結リストでつなぐという方法である（図６）。



（ａ）　　（ｂ）

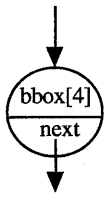
図６　（ａ）領域の例　（ｂ）（ａ）に対する分割線リスト型４分木

図６（ｂ）のノードの型を詳細に記述すると、図７の様になる。



Quad[4] : 副領域の境界を表す４座標
Objcount : Objに挿入されているオブジェクトの数
Xlist,Ylist : X,Y分割線に交差するオブジェクトへのポインタ
Ur,Ul,Ll,Lr : ４つの子へのポインタ
Obj : 葉のノードにおけるオブジェクトへのポインタ

図７（ａ） 分割線リスト型４分木のデータ型



bbox[4] : オブジェクトの境界４座標
next : 次のオブジェクトへのポインタ

図７（ｂ） オブジェクトのデータ型

次に、木構造構築、及び領域探索のアルゴリズムの概略を以下に示す。

◎木構造構築

木構造構築を行う関数qtree_buildには、次の２つの値が渡る。

・根のノードqroot（Objには全長方形オブジェクトの線形連結リストの先頭へのポインタ、Objcountには全オブジェクト数がそれぞれ入っている）へのポインタ
・根の領域の４座標

```
qtree_build(Q,rect) {
  if(Q==NULL) return;
  if(Q->Objcount<=しきい値) return;
  Q->Quad=rect;
  rectを４つの象限に分割;
  全てのQ->Objについて do {
       if(Objectが第１象限に含まれれば) {
           Q->Ur->Objにそれを挿入;
           Q->Ur->Objcount++;
       }
       else if(Objectが第２象限に含まれれば) {
           Q->Ul->Objにそれを挿入;
           Q->Ul->Objcount++;
       }
       else if(Objectが第３象限に含まれれば) {
           Q->Ll->Objにそれを挿入;
           Q->Ll->Objcount++;
       }
       else if(Objectが第４象限に含まれれば) {
           Q->Lr->Objにそれを挿入;
           Q->Lr->Objcount++;
       }
       else if(ObjectがＸ分割線に交差していれば)
           Q->Xlistにそれを挿入;
       else if(ObjectがＹ分割線に交差していれば)
           Q->Ylistにそれを挿入;
  }
  Q->Obj=NULL ;
  Q->objcount=-1 ;

  qtree_build(Q->Ur,第１象限の境界座標);
  qtree_build(Q->Ul,第２象限の境界座標);
  qtree_build(Q->Ll,第３象限の境界座標);
  qtree_build(Q->Lr,第４象限の境界座標);
}
```

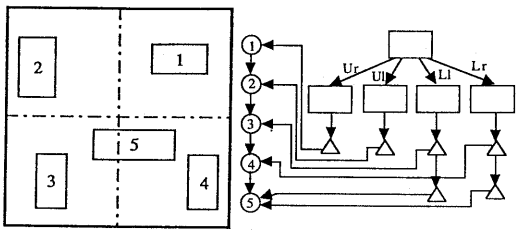
◎領域探索

qには最初に木構造の根へのポインタ、windowには探索領域の境界の４座標が渡る。

```
regionsearch( q,window ) {
   if(qが葉ならば) {
        Objをたどり、windowと交差するものを列挙 ;
        return ;
   }
   else {
        Xlist,Ylistをたどり、windowと交差するものを
        列挙;
   }
   if(q->Ur->Quadとwindowが交差)
        regionsearch(q->Ur,window) ;
   if(q->Ul->Quadとwindowが交差)
        regionsearch(q->Ul,window) ;
   if(q->Ll->Quadとwindowが交差)
        regionsearch(q->Ll,window) ;
   if(q->Lr->Quadとwindowが交差)
        regionsearch(q->Lr,window) ;
}
```

2－2　重複保持型４分木（ＭＵＬＴＩ）

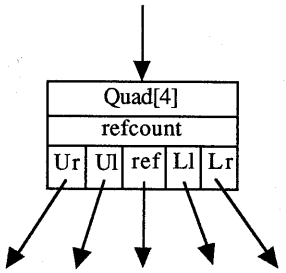
分割の過程において、２つ以上の副領域に交差するオブジェクトはそのままその副領域に入れてしまう手法である（図９）。重複するオブジェクトはそのコピーを作成し葉に保持しても良いが、メモリのオーバーヘッドが極めて大きい。そこで、オブジェクト実体へのポインタのみからなるリファレンスを作成し、葉に保持する。



（ａ）　　（ｂ）

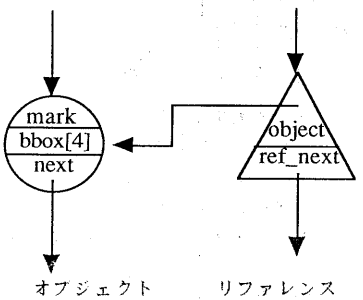
図８　（ａ）領域の例　（ｂ）（ａ）に対する重複保持型４分木

図８（ｂ）のノードの型を詳細に記述すると図９の様になる。



Quad[4] ： 副領域の境界を表す４座標
refcount ： refに挿入されているリファレンスの数
Ur,Ul,Ll,Lr ： ４つの子へのポインタ
ref ： 葉のノードにおけるリファレンスへのポインタ

図９（ａ）　重複保持型４分木のデータ型



mark ： 重複チェックのためのフラグ
bbox[4] ： オブジェクトの境界４座標
next ： 次のオブジェクトへのポインタ
object ： オブジェクト実体へのポインタ
ref_next ： 次のリファレンスへのポインタ

図９（ｂ）　オブジェクトとリファレンスのデータ型

さらに各アルゴリズムについて述べる。

◎木構造構築

分割線リスト型４分木のqtree_buildに準ずる。但し、分割線リストは用いず、オブジェクトが交差する全ての副領域にリファレンスを作成し、リストに挿入していく所が異なる。

◎領域探索

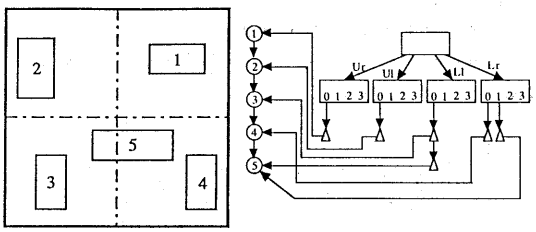
分割線リスト型のregionsearchの分割線リスト探索を無くし、探索対象をリファレンスとするだけでは不十分である。なぜなら、このまま探索を行なえば、同じオブジェクトが重複して列挙されるからである。

そこで、オブジェクトにフラグ（mark）を用いる。これは最初全て０である。もしそれが列挙されたならばフラグを１にし、２度と列挙しない様にする。こうすることにより、重複列挙を避ける。

１回の領域探索が終る毎に、領域探索とほとんど同様な手続きを用いて、領域探索で立ったオブジェクトのフラグは全て０に戻す。

2－3　４分リスト型４分木（ＱＬＱＴ）

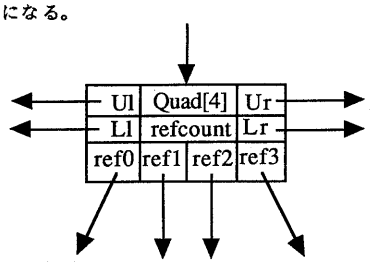
基本的には２－２の重複保持型４分木と同じである。しかし、葉におけるリファレンスリストを４種類に分類し、領域探索時のアルゴリズムに工夫をすることによって、重複列挙を避けている。



（ａ）　　（ｂ）

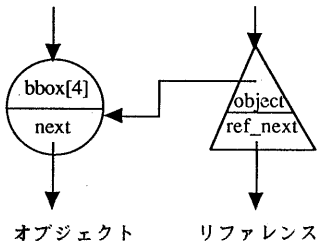
図１０　（ａ）領域の例　（ｂ）（ａ）に対する４分リスト型４分木

図１０（ｂ）のノードの型を詳細に記述すると図１１の様になる。



Quad[4] : 副領域の境界を表す４座標
refcount : refに挿入されているリファレンスの数
Ur,Ul,Ll,Lr : ４つの子へのポインタ
ref0 :
ref1 :
ref2 :
ref3 :
} 葉のノードにおける リファレンスへのポインタ（４個）

図１１（ａ） ４分リスト型４分木のデータ型



bbox[4] : オブジェクトの境界４座標
next : 次のオブジェクトへのポインタ
object : オブジェクト実体へのポインタ
ref_next : 次のリファレンスへのポインタ

図１１（ｂ） オブジェクトとリファレンスのデータ型

各アルゴリズムについて以下に述べる。

◎木構造構築

重複保持型４分木の木構造構築に準ずる。
但し、葉において、リファレンスを表１に示すような４種類のリストに分類する。

オブジェクトと副領域の底辺が	オブジェクトと副領域の左辺が	リストタイプ
交差しない	交差しない	0
交差しない	交差する	1
交差する	交差しない	2
交差する	交差する	3

表１ 葉におけるリファレンス分類表

◎領域探索

先のregionsearchと同様、まず深さ優先探索で、ウインドゥと交差する葉のノードに到達する。その後、windowの左辺と交差する葉については、そのノード中の０，１リストを調べる。windowの下辺と交差する葉については、そのノード中の０，２リストを調べる。windowの左下隅を含む葉については、そのノード中の０，１，２，３リストを調べる。その他の葉については、そのノード中の０リストを調べる。この探索により、オブジェクトは必要かつ十分に探索され、１回だけ列挙される。この証明については[5]を参照されたい。

２－４ 改良型４分木（ＭＯＤ）

２つ以上の副領域に交差するオブジェクトについては、ある基準に基づき、１つの副領域に含めてしまうのがこの方法である。ここでは、オブジェクトの左下隅を含む副領域に含めることにする（図１２）。

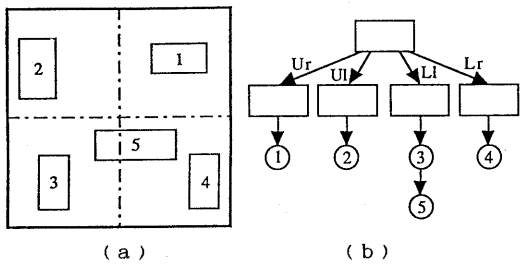
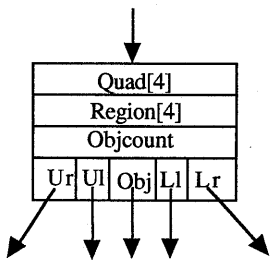


図１２（ａ）領域の例 （ｂ）（ａ）に対する改良型４分木

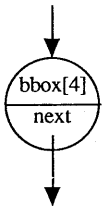
図１２（ｂ）のノードの型を詳細に記述すると図１３の様になる。



Quad[4] : 副領域の境界を表す４座標
Region[4]:修正された副領域の境界座標
Objcount : Objに挿入されているオブジェクトの数
Ur,Ul,Ll,Lr : ４つの子へのポインタ
Obj : 葉のノードにおけるオブジェクトへのポインタ

図１３（ａ）改良型４分木のデータ型

ここでRegin[4]とは、そのQuadに保持されている全てのオブジェクトを外包する最小の長方形の境界座標であり、Quad[4]に代わり、探索時のキー値となる。



bbox[4] ： オブジェクトの境界４座標
next ： 次のオブジェクトへのポインタ

図１３（ｂ） オブジェクトのデータ型

各アルゴリズムを以下に述べる。

◎木構造構築

分割線リスト型４分木のqtree_buildに準ずる。但し、分割線リストは用いず、オブジェクトの左下隅を含む副領域のリストにオブジェクトを挿入していくところが異なる。

◎領域探索

分割線リスト型４分木のregionsearchに準ずる。但し、分割線リストの探索は必要ない。また、探索時の分岐の条件は、Region[4]とウインドゥが交差するか否かとなる。

３．時間・空間複雑度

オブジェクトが平面に均一に分布していると仮定したときの各木の空間複雑度、及び各操作の時間複雑度を以下に示す。但しオブジェクト数（重複保持型４分木、４分リスト型４分木ではリファレンス数）をｎ，しきい値をＳとする。

☆メモリ使用量

$O(n/S+n)$

☆木構造構築時間

$O(n\log_4(n/S))$

☆領域探索時間

点探索等の、ウィンドゥが比較的小さいとき

$O(\log_4(n/S)+S(+L))$

但し、Ｌは分割線リスト型４分木での、分割線リストに入るオブジェクト数のうち、探索時に訪問されるオブジェクト数であり、Ｌは高々$n^{1/2}\log n$に比例した個数であることが言える。

４種類何れの木の場合も、ウインドゥが大きくなるにつれ、領域探索の時間複雑度はＯ（ｎ）に近づく。

４．実験結果及び考察

２章で示した４種類の４分木をＣ言語で実装したものをＳｕｎ４／１１０上で動作させ、実験を行った。その結果を表２～６、図１４～２３に示す。なお、領域全体の大きさは100000*100000であり、そこに長方形オブジェクトを乱数によってほぼ均一分布になるように発生させた。木構造構築時間は、既に計算機の主記憶上にある全オブジェクトの線形連結リストから木構造を構築するまでの時間である。領域探索時間は、ウィンドゥの大きさを固定し、乱数で位置を変えながら探索を数千回試行したものの平均である。なおウインドゥの大きさは25000*25000、5000*5000、0*0（即ち点探索）の３種類である。いずれの木の場合も、しきい値を１０と１００の２つの場合について実験を行った。

実験結果より次の様なことが明らかになった。

まず、全ての木について領域探索時間に着目した場合、ウィンドウが25000*25000の時に、しきい値が１００の方が１０の場合よりも領域探索時間が短くなる傾向が見られる（表４、図１８、図１９）。この様にウィンドゥが大きい場合は、しきい値が大きい方が探索すべき葉のノード数が少なくて済む分、有利であると思われる。ウインドゥが小さいときには領域探索時間はしきい値と共に増大していることが読み取れる（表６、図２２、図２３）。これは３章で示した理論的な時間複雑度とも合致している。今回の実験ではしきい値は２種類しか用いていないが、より多くのしきい値を用い領域探索時間を計測する実験は興味深い。

次に各々の木の性能比較を行う。

メモリ使用量は、重複保持型と４分リスト型は分割線リスト型と改良型の約２倍であることがわかる（表２、表３、図１４、図１５）。改良型４分木のみが持つ付加的な領域情報（図１３（ａ）におけるRegion）も、さほどのメモリ使用量のオーバーヘッドを生じていない様である。

木構造構築時間は、重複保持型と４分リスト型は、分割線リスト型と改良型よりも２～６倍程度要している（表２、表３、図１６、図１７）。

領域探索時間は、重複保持型の探索時間が他の３つに比べ長い（表４～６、図１８～２３）。これは、重複列挙を避けるための２回の探索が影響しているからである。また、分割線リスト型は、ウインドゥの大きいうちは探

索時間は比較的短いが、ウインドゥが小さくなるにつれて他の３つに比べ探索時間が長くなってくる（表６、図２２、図２３）。これは、３章で示したＬの影響が増大して来るからである。重複保持型と４分リスト型では、探索時間に約２～４倍の差があることが分かり（表４～６、図１８～２３）、Weytenらが以前実験した結果と合致する。改良型と４分リスト型は、他の２つに比べいかなるウィンドウの大きさ、オブジェクトの数についても探索時間は短い（表４～６、図１８～２３）。

今回の実験によれば、メモリ使用量、木構造構築時間、領域探索時間を総合的に判断したとき、４種類の４分木の中では改良型が最も優っているといえるであろう。

参考文献

[1]R.A.Finkel and J.L.Bentley,"Quad-trees--A data structure for retrieval on composite keys,"Acta Inform.,vol.4,no.1-9,1974.

[2]H.Samet,"The Quadtree and Related Hierarchical Data Structures",Computing Surveys,Vol.16.No.2, June 1984.

[3]J.B.Rosenberg,"Geopraphical data structures compared: A study of data structures supporting region queries,"IEEE Trans.Computer-Aided Design, vol.CAD-4,pp.53-67,Jan.1985.

[4]R.L.Brown,"Multiple Storage Quad Trees: A Simpler faster alternative to bisector list quad trees, "IEEE Trans.Computer-Aided Design,vol. CAD-5,pp.413-419,July 1986.

[5]L.Weyten and W.D.Pauw,"Quad List Quad Trees: A Geometrical Data Structure with Improved Performance for Large Region Queries, "IEEE Trans. Computer-Aided Design,vol.8,No.3,pp.229-233,March 1989.

[6]A.Pitaksanonkul et al.,"Comparisons of Quad Trees and 4-D Trees:New Results,"IEEE Trans. Computer-Aided Design,vol.8,No.11,November 1989.

オブジェクト数 n（個）	分割線リスト型４分木（NORMAL）		改良型４分木（MOD）		重複保持型４分木（MUTLI）			４分リスト型４分木（QLQT）	
	使用メモリ（bytes）	木構造構築（秒）	使用メモリ（bytes）	木構造構築（秒）	リファレンス数（個）	使用メモリ（bytes）	木構造構築（秒）	使用メモリ（bytes）	木構造構築（秒）
５１２	15096	0.03	18240	0.02	855	28356	0.07	28804	0.14
１０２４	32356	0.06	37780	0.09	1674	54764	0.23	55228	0.32
２０４８	62456	0.11	75600	0.19	3513	116404	0.51	118868	0.65
４０９６	126764	0.27	153580	0.40	6912	226540	1.25	230028	1.44
８１９２	251116	0.62	308760	0.87	13380	459612	3.87	468268	3.06
１６３８４	505124	2.38	607300	2.15	27091	898484	5.87	911620	6.49

表2　しきい値＝１０のときの各４分木のメモリ使用量及び木構造構築時間

オブジェクト数 n（個）	分割線リスト型４分木（NORMAL）		改良型４分木（MOD）		重複保持型４分木（MUTLI）			４分リスト型４分木（QLQT）	
	使用メモリ（bytes）	木構造構築（秒）	使用メモリ（bytes）	木構造構築（秒）	リファレンス数（個）	使用メモリ（bytes）	木構造構築（秒）	使用メモリ（bytes）	木構造構築（秒）
５１２	11144	0.02	11280	0.02	596	18012	0.05	16204	0.05
１０２４	21800	0.05	22000	0.04	1170	35244	0.14	31484	0.17
２０４８	44360	0.07	45120	0.12	2397	71748	0.34	64468	0.34
４０９６	87192	0.14	88240	0.25	4737	141380	0.89	126388	0.80
８１９２	175976	0.35	179040	0.57	9536	284940	2.17	255436	1.86
１６３８４	348968	0.80	355840	1.26	18948	567052	3.39	507564	4.07

表3　しきい値＝１００のときの各４分木のメモリ使用量及び木構造構築時間

オブジェクト数（個）	しきい値＝１０				しきい値＝１００			
	NORMAL	MOD	MULTI	QLQT	NORMAL	MOD	MULTI	QLQT
	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）
５１２	1.26	1.03	3.35	1.37	1.41	1.27	3.38	1.41
１０２４	2.07	1.57	4.84	2.00	3.70	2.06	5.18	2.55
２０４８	3.71	2.74	9.22	4.02	4.78	3.29	7.39	3.53
４０９６	6.68	5.47	16.55	6.65	6.56	5.78	14.09	5.85
８１９２	11.56	9.33	32.96	11.52	10.50	9.39	26.85	9.78
１６３８４	30.40	17.90	60.89	21.21	18.07	15.36	38.25	18.50

表４　領域探索における各４分木の探索時間（ウインドゥの大きさ25000*25000）

オブジェクト数（個）	しきい値＝１０				しきい値＝１００			
	NORMAL	MOD	MULTI	QLQT	NORMAL	MOD	MULTI	QLQT
	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）
５１２	0.68	0.34	0.81	0.42	0.78	0.39	1.08	0.56
１０２４	0.85	0.39	1.03	0.48	1.18	0.78	1.85	0.70
２０４８	1.36	0.53	1.50	0.75	1.59	0.98	2.43	0.95
４０９６	1.86	0.68	2.27	0.83	2.11	1.32	2.78	1.40
８１９２	2.61	1.07	3.27	1.25	3.54	1.48	4.28	1.70
１６３８４	4.98	1.64	5.27	2.15	4.49	2.26	5.42	2.55

表５　領域探索における各４分木の探索時間（ウインドゥの大きさ5000*5000）

オブジェクト数（個）	しきい値＝１０				しきい値＝１００			
	NORMAL	MOD	MULTI	QLQT	NORMAL	MOD	MULTI	QLQT
	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）	探索時間（msec.）
５１２	0.55	0.20	0.45	0.21	0.66	0.43	0.75	0.40
１０２４	0.70	0.21	0.50	0.22	0.75	0.50	0.83	0.45
２０４８	0.84	0.24	0.55	0.26	0.96	0.58	0.95	0.52
４０９６	1.28	0.26	0.62	0.27	1.49	0.59	1.20	0.62
８１９２	1.67	0.28	0.70	0.28	1.90	0.60	1.52	0.64
１６３８４	2.23	0.32	0.91	0.32	4.36	0.67	1.93	0.74

表６　領域探索における各４分木の探索時間（ウインドゥの大きさ0*0、即ち点探索）

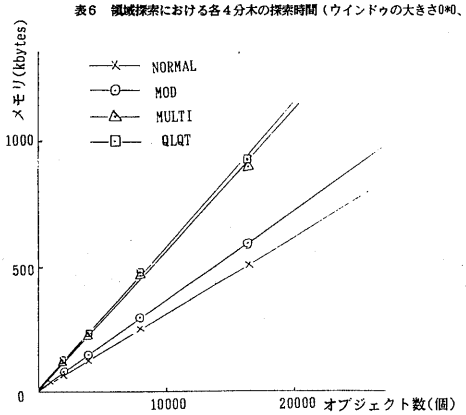


図１４　各４分木の使用メモリ量
（しきい値＝１０）

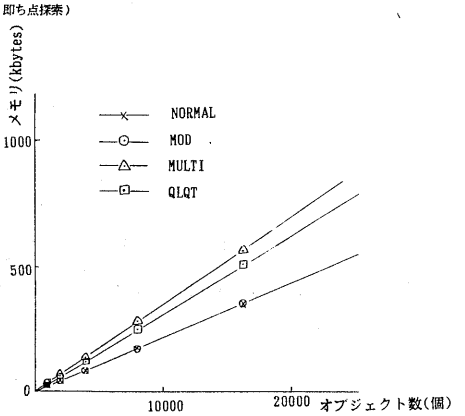


図１５　各４分木の使用メモリ量
（しきい値＝１００）

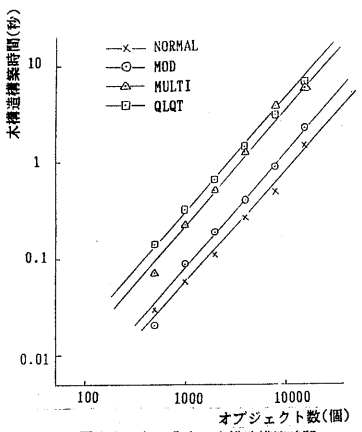


図１６　各４分木の木構造構築時間
（しきい値＝１０）

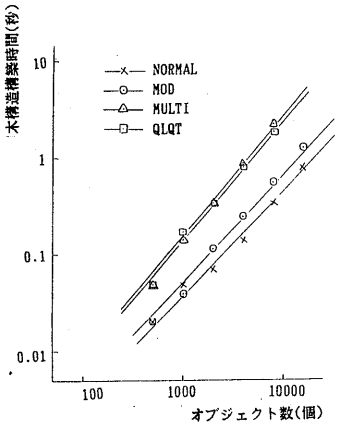


図１７　各４分木の木構造構築時間
（しきい値＝１００）

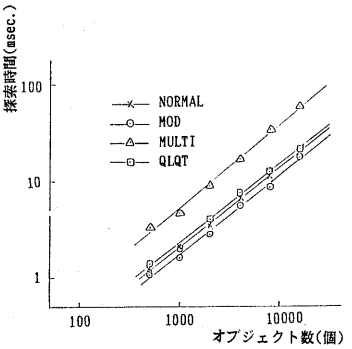


図１８　各４分木における領域探索時間
（ウィンドウ＝25000*25000，しきい値＝１０）

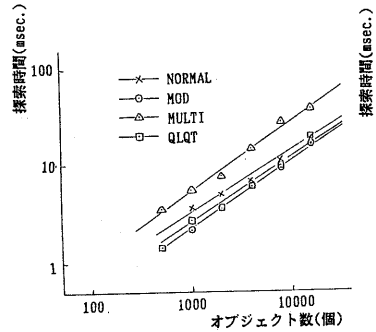


図１９　各４分木における領域探索時間
（ウィンドウ＝25000*25000，しきい値＝１００）

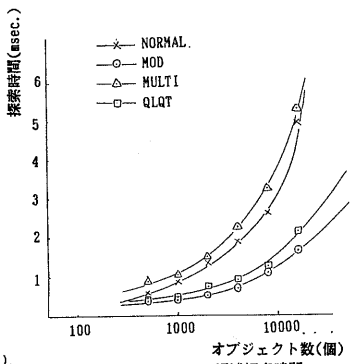


図２０　各４分木における領域探索時間
（ウィンドウ＝5000*5000，しきい値＝１０）

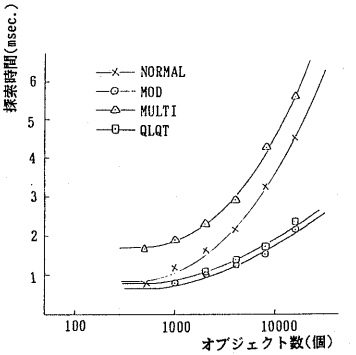


図２１　各４分木における領域探索時間
（ウィンドウ＝5000*5000，しきい値＝１００）

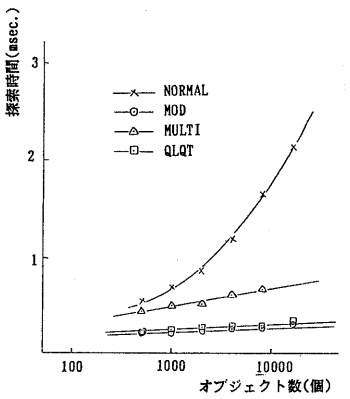


図２２　各４分木における点探索時間（しきい値＝１０）

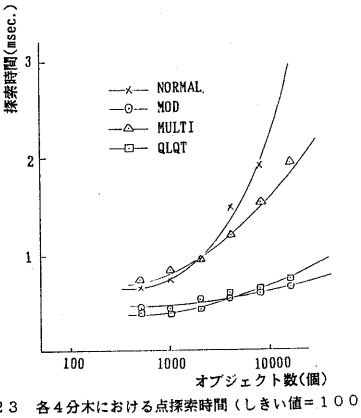


図２３　各４分木における点探索時間（しきい値＝１００）