THE EXCELL METHOD FOR EFFICIENT GEOMETRIC ACCESS TO DATA

Markku Tamminen
Reijo Sulonen

Helsinki University of Technology
Laboratory of Information Processing Science
02150 Espoo 15, Finland

ABSTRACT

The extendible cell (EXCELL) method provides a data structure for efficient geometric access. It stores geometric data into computer storage blocks corresponding to disjoint variable sized rectangular cells accessible by an address calculation type directory. We describe the method for point files and files of more complicated figures analyzing performance. We report algorithms for the nearest neighbour and point-in-polygon-network problems and describe applications to geographical data bases, hidden line elimination and geometric modeling.

Introduction

By geometric data we mean representations of geometric structures composed of points, line-segments, surface-areas and volumes, central in computer graphics, geographic data management and computer aided design. These applications moreover require geometric access meaning the retrieval of such objects on the basis of geometric relationships like proximity and intersection. Constructing efficient data structures for this is a central problem of computational geometry[17].

This paper analyzes a data structuring approach aiming at efficient geometric access. The following building block tasks of CAD typify our aims:

1. nearest neighbour search[4]
2. the point-in-polygon-network problem[15]
3. range search of points[1] and other geometric objects
4. intersection problems[17,2]
5. geometric analysis by ray-casting[16]
6. checking interference[5] and geometric integrity[13]
7. hidden line and surface problems[18].

We present the extendible cell (EXCELL) approach, a framework for data structures and algorithms for maximally efficient geometric access. It is especially suited for data in external memory. EXCELL is closely related to the quad-tree[27,3,11] and fixed cell[4,9,7] techniques combining their good features. We analyze data structures for sets of points and two-dimensional polygon networks as concrete examples of the approach.

Sample algorithms and application systems for problems 1, 2, 3, 5 and 7 above demonstrate that EXCELL is simple to implement and leads to good expected efficiency. Theoretical results on EXCELL and related methods confirm the empirical inferences.

For simplicity we use two-dimensional examples and terminology (study-area, rectangle,...) even for concepts with wider applicability. We use interchangeably terms "edge" for line-segment and "face" for two- or three-dimensional surface-area (polygon). This presentation relies on detailed analyses reported elsewhere[19,20,21,22,23,24].

Order preserving extendible hashing

It is useful to first consider the simplest case. A one-dimensional data structure corresponding to the geometric problems defined above would support the operation find-next(key) and be called a generalized priority queue.

From recent developments in one-dimensional data structures the extendible hashing (EXHASH) of Fagin et al.[6] seems most promising for us. From our viewpoint order preserving EXHASH can be defined and analyzed as a binary bucket trie[20].

Let us have a set F (file) of real-valued keys of a key-space U. (For simplicity we assume U = [0,1). A binary bucket trie T is the tree obtained by recursively halving U until no interval contains more than b (bucket size) keys (figure 1):

1. To each node n of T corresponds an interval I(n) of U.
2. U itself corresponds to the root of T.
3. If there are more than b keys in the interval I(n) then node n has two sons corresponding to the two halves of I(n).
4. If there are at most b keys in I(n) then node n is a leaf (i.e. has no sons).

To the trie T there corresponds a complete binary tree (here called directory) with 2**d elements. The directory may be represented as an array (figure 1) and the element corresponding to any key of U accessed by index calculation.

EXHASH is a binary bucket trie represented by a directory and data buckets maintained dynamically by bucket splits and directory doublings (and corresponding merges, if necessary): If we add a record to a bucket already containing b records the original bucket interval is split in the middle, the b + 1 records distributed correspondingly to the two resulting buckets and the directory updated. If the depth of the original bucket was d the depth of the whole tree increases

0420-0098/82/0000/0345$00.75 © 1982 IEEE

meaning that the directory has to be doubled. Due to bucket splitting no overflow pages are needed and a record may be retrieved with one disk access (two counting the directory access).

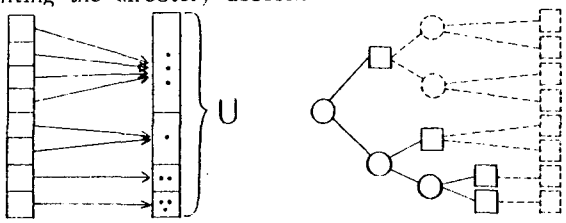


Figure 1. A binary bucket trie (EXHASH structure) with bucket size b = 3 and depth d = 3. The left part represents the directory and the bucket intervals; the right part the corresponding binary tree. Dashed lines represent the complete tree.

Ordinarily hashing means randomization. However, this is not eficient for operations such as find-next. Thus we apply EXHASH directly to the key values and call the result an order preserving address transformation. A main part of our work (chapter 7) has consisted of analyzing how non-uniform key-distribution affects the performance of EXHASH.

EXCELL - EXHASH for multidimensional points

We may apply EXHASH to obtain a multidimensional dynamic geometric file structure, which we have called EXCELL. We assume for simplicity the study area U to be the unit square U = [0,1) X [0,1). Let (x,y) of U have the binary representation

$$x = \sum_{1 \le i < \infty} a_i 2^{-i}$$
$$y = \sum_{1 \le i < \infty} b_i 2^{-i} \qquad (1)$$

and g be the following grid (hash) function

$$g(x,y) = \sum_{1 \le i < \infty} (a_i 2^{-(2i-1)} + b_i 2^{-2i}) \qquad (2)$$

Definition 1. An extendible cell (EXCELL) implementation of a point file (i.e. set) F on U is the structure obtained by applying EXHASH on the interval [0,1) to g(F).

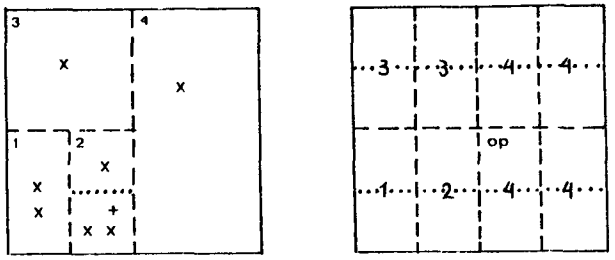
Figure 2 helps interprete this abstract definition.

The data-buckets of EXCELL correspond to rectangles (cells) formed by halving the study area alternatingly in the x- and y- directions. The depth of an EXCELL structure is the maximal number of halvings needed to obtain a cell. At depth d = dx + dy (dx = ⌈d/2⌉ ; dy = ⌊d/2⌋) the grid function (2) distributes the points of F onto an even grid of cells with x-spacing 2**(-dx) and y-spacing 2**(-dy). When d is increased the directory is doubled by halving all directory cells in the x- or y-direction.

The function g is difficult to implement efficiently. Because an EXCELL file has a directory we may, however, reorganize the directory at each doubling without affecting the data buckets. Therefore the grid transformation may be replaced by an array-index-like calculation, for instance:

$$ind(x,y) = 2^{dy} 2^{dx}[x] + 2^{dy}[y]. \qquad (3)$$

In the ind-order the directory forms a two-dimensional array.



x: points
o: cursor
------: cell borders
.....: effect of inserting the new point +

Figure 2. Point EXCELL concepts. The study area is divided into four data cells (=rectangles), none of which contain more than three (=b) points. The directory is an array of elements each corresponding to a rectangle of minimal size and indicating the data cell containing it. The data cell of a cursor point p is retrieved by first calculating the directory array index by the EXCELL grid function. The insertion of the new point (+) requires the cell division and directory doubling operations indicated by short dashes.

Irrespective of the directory calculation cells correspond to buckets induced by function (2).

The main guarantee of EXCELL is:

each leaf cell contains at most b points

the father of each cell (i.e. the rectangle from which the cell was formed by halving) contains at least b + 1 points. } (4)

The EXCELL method for more complicated objects

By more complicated objects we mean line-segments, surface-areas and volumes. The main difference to point files is that an object may intersect several cells of a fixed or extendible grid. We might thus speak of spatial directories, which indicate the data base objects intersecting each geometric cell.

The extendible cell method[21,22] subdivides cells until the contents of each one is described within a specified storage space. Thus it is close to the Warnock hidden surface method[27]. The cell structure is supplemented by a directory identical to that of the previous chapter. The vagueness of this description shows that EXCELL is a general principle specified separately for each problem. We describe an application of EXCELL to two-dimensional polygon networks (figure 3) in somewhat more detail.

Two-dimensional polygon networks are perhaps the most important geometric structure of geographic data management. Also the visible surface-areas of a projection of a three-dimensional solid form a two-dimensional area-partition. The following simple structure has been used for both of these applications[21,23].

We use an edge-oriented representation of polygon

networks (fig. 4). Boundaries are decomposed into elementary edges. The direction of each edge is defined "lexicographically" with the right and left neighbour polygons recorded in the edge record.

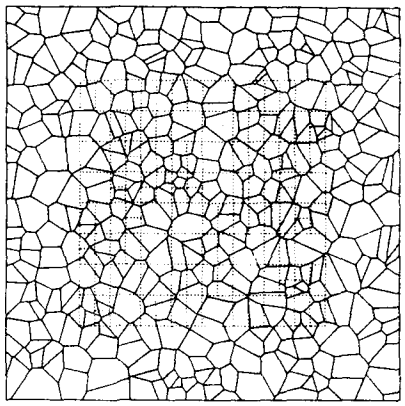


Figure 3. A 400 polygon network generated by the Dirichlet tessellation model (chapter 8). Superposed is an EXCELL structure with b = 10.

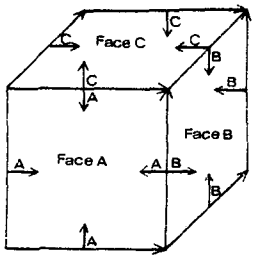


Figure 4. A cube with forward faces A, B and C represented by directed edges and their right/left -neighbour information.

We implement a spatial directory by dividing the (x,y) - study area into non-overlapping rectangular cells corresponding to blocks of storage space. Here we do not utilize pointers between cells and edges but store an edge record (after clipping its geometry) in each cell intersected (figure 5).

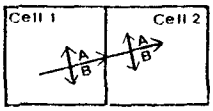


Figure 5. Each edge is clipped and the remaining edgeparts "stored in" the cells intersected.

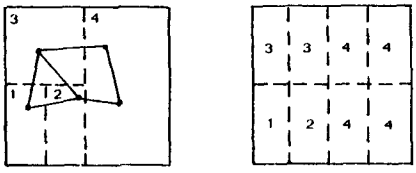


Figure 6. Edge EXCELL concepts. U is divided into four cells (the data part) so that no cell is intersected by more than three edges (o——o). See figure 2 for further explanations.

EXCELL for edges is analogous to point-EXCELL (figure 6). Cell (bucket) size is defined by a maximal number of intersecting edges and cell division by the clipping operation of figure 5.

Access paradigms

We have chosen four geometric queries to reflect the essential access requirements of the tasks mentioned in chapter 1. As the access paradigms for EXCELL point files we use the range query[1] and the nearest neighbour query[4]:

given an arbitrary point (cursor) p of U find the point of file F, which is closest to p. } (NN)

We discuss the implementation in chapter 6.

Our access paradigm for polygon network files is the point-in-polygon-network query:

Given a polygon network N and an arbitrary point (cursor) p of U determine the polygon P(p) of N containing p. } (PIPN)

A geometric model of a solid may be defined by a representation of its boundary, i.e. a set of oriented faces. We may define a three-dimensional face-directory for a geometric volume model by letting each three-dimensional cell contain references to all faces intersecting it. We use "ray-casting"[16] as the access paradigm:

Given points p0 and p1 determine, which face is intersected first by the semi-infinite ray

$$p0 + t(p1 - p0),\ 0 <= t < \text{infinity}.$$ } (RC)

Algorithms

Exact match retrieval for point files is almost identical to that of EXHASH[6]. We present only the insertion algorithm for edges (line-segments) to provide geometric flavour (fig. 7). Insertion of points is just a bit simpler.

Next we study the main geometric access paradigms. The simplest way to implement a range query is to utilize the directory as a two-dimensional array as in the following pair of routines:

```
var ci : set of cell;

procedure open_range (var r : rectangle);
/* open range query for rectangle r */
    ci := []; /* empty set */
    for each directory element; whose cell intersects r do
        ci := ci + [c]; /* set union */

function get_next_in_range() : cell;
/* get next cell in opened range */
    if ci = [] then get_next_in_range := NIL
    else begin choose a cell c of ci;
              ci := ci - [c];
              get_next_in_range := c
         end;
```

```
procedure insert_ex (var e : edge); /* insert edge into EXCELL file */
    while not_null(e) do
        begin c := get_cell_ins(start(e));
              clip_edge(c,e,el);
              insert_cell(c,el) /* el assumed non-degenerate */
        end;

function get_cell_ins (var p : point) : cell;
/* get a cell guaranteed to contain free space */
    c := get_cell(get_dir(p));
    while full(c) do
        begin divide_cell(c);
              c := get_cell(get_dir(p))
        end;
    get_cell_ins := c;

function get_dir (var p : point) : cell_number;
/* retrieve number of cell corresponding to point p */

function get_cell (var n : cell_number) : cell;
/* retrieve cell having number n */

procedure insert_cell (var c : cell; e : edge); /* insert edge into cell */

procedure clip_edge (var c : cell; e,el : edge);
/* clip edge e; store e  c in el and e - el in e */

procedure divide_cell (var c : cell); /* divide a cell */
    if cannot_be_divided(c) then dup_dir();
    if last_divided_in_y_direction(c) then x_halve_cell(c,c1,c2)
    else y_halve_cell(c,c1,c2);
    mod_dir(c1); mod_dir(c2);
    for each edge e in c do
        begin clip_edge(c1,e,el);
              if not_null(el) then insert_cell(c1,el);
              if not_null(e) then insert_cell(c2,e)
        end;
    release(c); /* storage of c is returned to free space */

procedure x_halve_cell (var c,c1,c2 : cell);
/* halve c perpendicularly to x-axis forming new cells c1 and c2 */

procedure dup_dir(); /* duplicate directory in appropriate direction */

procedure mod_dir (var c : cell); /* modify directory part intersecting
rectangle of cell c to contain the number of c */
```

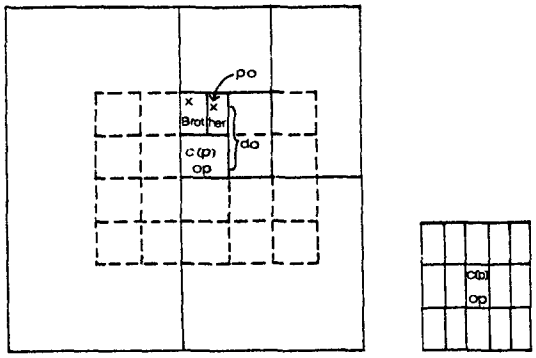
Figure 7. Insertion algorithm for edges.

For the nearest neighbour and point-in-polygon-network queries we utilize what we call brother algorithms relying on the main guarantee (4). We call the brother of a cell the rectangle formed in the same halving operation. Always either a cell or its brother (which may contain several data cells) is non-empty. Figure 8 shows that assuming the maximum coordinate difference metric the size of the area that we must inspect in the NN-query is at most 20 times the size of the cell of the cursor. From the same area we may retrieve all b + 1 nearest neighbours. The loop LOOP may be efficiently implemented by the EXCELL method.

```
function find_nearest (var p : point) : point;
/* find nearest neighbour of cursor p */
    c := get_cell(get_dir(p)); /* get cell c overlapping point p */
    d0 := maximal distance from p to any point of brother(c);
    for each cell c' within a distance less than d0 from p do /* LOOP */
        closest(p,c',p0,d0); /* d0 may change */
    find_nearest := p0;

procedure closest (var p : point; c : cell; p0 : point; d0 : distance);
/* find point p0 closest to p within distance d0 in cell c */
    for each point p' in c do
        if distance(p,p') < d0 then
            begin d0 := distance(p,p');
                  p0 := p
            end;
```

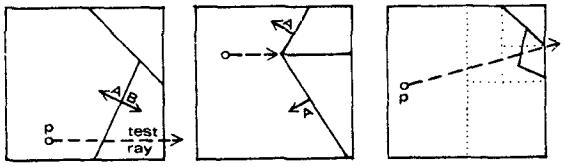
With neighbourhood information stored together with line-segments (figure 4) we may answer a PIPN-query as described in figure 9 and the algorithm find_polygon. The algorithm presented is simplified by considering the line-segment closest to p in a cell.



Part a
x: point
o: cursor

Part b

Figure 8. The brother algorithm. Solid lines show an EXCELL partition with bucket size b = 1. There are at most 20 rectangles of the size of c(p) to be checked. This area is called cnn(p) and indicated by dashes. Parts a and b demonstrate the two possible forms of c(p). In this example all data-cells are retrieved to answer the query.



(a) (b) (c)

Figure 9. The polygon a cursor p belongs to is found by casting a test ray from p and determining on which side of the first intersected segment p lies. To special cases we apply more elaborate tests. Case (a) depicts the ordinary situation, (b) special case and (c) the case of an empty cell.

```
function find_polygon (var p : point) : polygon;
/* find polygon containing cursor p */
    c := get_cell(get_dir(p)); /* get cell of cursor p */
    if not_empty(c) then
        find_polygon_within_cell(c,p)
    else begin c' := find_nephew(c);
               find_polygon := find_polygon_within_cell(c',p);
         end;

function find_polygon_within_cell (var c : cell; p : point) : polygon;
/* search within a non-empty cell */
    find edge e closest to p within c; /* O(b) operations */
    if e unique then find_polygon_within_cell := polygon(p,e)
    /* polygon on the same side of e as p or "p outside" */
    else find_polygon_within_cell := resolve_special_case(p,c,e);

function find_nephew (var c : cell) : cell; /* find first non-empty nephew */
/* check the nephews (offspring of the brother of c) in such an
order that when a non-empty one is found the whole line from p to
any part of the nephew is contained in cells already checked (i.e.
empty) - see figure 11 (*). */
```

The above algorithm demonstrates two-dimensional ray-casting. Next we present a three-dimensional version. Here U denotes the "box" of interest.

```
function first_face_on_ray (var p : point; ray : direction) : face;
/* retrieve first face from point p in direction ray */
    if not in_box(p,U) then /* p arbitrary start of infinite ray */
        begin p := frontface_intersection(p,ray,U);
        /* intersection of ray with a "forward" face of box U */
            first_face_on_ray := first_face_on_ray(p,ray)
        end;
    else begin c:= get_cell(get_dir(p));
            f:= NIL; /* face to be retrieved initialized */
            while in_box(p,U) and f = NIL do
                f := first_face_on_ray_in_cell(p,ray,c)
        end;
    first_face_on_ray := f;

function first_face_on_ray_in_cell
(var p : point; ray : direction; c : cell) : face; /* within-cell search */
    if ray intersects a face then
        first_face_on_ray_in_cell := closest intersected face;
        /* O(b) intersection and polygon containment checks */
    else begin p := backface_intersection(p,dir,c);
    /* intersection with a "backward" face of box c */
    first_face_on_ray_in_cell := NIL
    end;
```

The directory mechanism provides maximally direct access, characteristic of EXCELL compared to other approaches.

Analysis of EXCELL for points

EXCELL cannot claim worst case optimality except for exact match queries. To analyze the expected performance we must postulate a data generation model. Analogously to the one-dimensional model of[6] we have chosen the Poisson point process. Specifically we analyse a varying intensity process with constrained variation (VIPC).

Intensity defines the expected number of points in any subset of U and variation V is defined as the ratio of maximal to average intensity. This is the main parameter. Constrained variation means informally that the intensity function is smooth.

We have been able to perform most analysis of multi-dimensional point-EXCELL using the one-dimensional EXHASH model. The analysis of order preserving EXHASH cannot, however, rely on the uniform randomization assumptions of ordinary hashing. The results on EXHASH are presented in table 1.

The results on EXHASH independent of the order of the directory or the neighbourhood relationship of data intervals apply to EXCELL as defined by (2) and (3). This includes all the results of table 1 except the find-next query. Some further corollaries are:

- the asymptotic expected filling ratio of data buckets is fixed as ln 2 (0.69) in the VIPC model irrespective of V
- the asymptotic expected storage efficiency of quad-tree data buckets is 2/3 of that of the EXCELL data part.

In[24] we showed that under the VIPC model EXCELL behaves asymptotically as if intensity were constant. Also the expected cost of the NN-query is bounded (i.e. O(1)) irrespective of the expected number of points in the file and that the asymptotical probability of having to search more than one cell in the brother algorithm is < 3/sqrt(b) (b: bucket size). This asymptotic independence of intensity variation is the main asset of EXCELL when compared to fixed cell techniques.

	Worst case	Good case	Smooth case
1.Directory storage (elements)	O(CV/b)	$O(C^{1+1/b}/b)$	$O((CV)^{1+1/b}/b)$
2.Datastorage (buckets)	O((C+ClogV)/b)	Clog(e)/b	O(C/b)
3.Average preprocessing (data part)	O(1+logV/b)	O(1+1/b)	O(1+1/b)
4.Exact match query	2	1(buffer)	2
5.Find-next query	O(1+log(VC))	O(1+1/b)	O(1+1/b)
6.Range query	O(1+VR)	O(1+R)	O(1+R)

Table 1. Performance characteristics of EXHASH as measured by the number of storage buckets and bucket accesses. Notation:

C = total (expected) number of records
V = variation of intensity
b = maximal number of records in a data bucket
R = expected number of records retrieved

Average preprocessing means total preprocessing cost divided by C. By smooth case we mean keys generated by a sufficiently regular statistical distribution. Good case refers to a uniform random distribution of keys. Worst case is the worst possible assuming a fixed V. Log is to the base of 2.

Analysis of EXCELL for polygon networks

There exist no standard data generation models for polygon networks as opposed to point files. For empirical and theoretical analysis we have utilized the Dirichlet tessellation (DT) model (figure 3) and the random lines (RL) model (figure 10). The DT formed by the closest point loci generated by a set of uniformly distributed random centers is typical of fairly homogeneous area partitions. The RL-model exhibits extreme variation of polygon size. Its merit is that geometric probability[14] may be utilized to compare theoretically various cell methods.

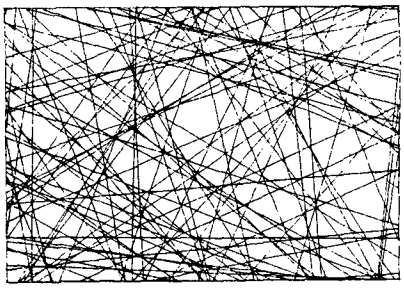


Figure 10. A realization of a 2571 polygon network formed by 100 random lines.

A basic result of geometric probability is:

> Let K1 K be bounded convex sets. The probability that a random line intersects K1 if it is known to intersect K is L1/L, L1 and L being the perimeters of K1 and K. (GP)

Based on (GP) we derive closed formulas for the performance of fixed cell methods in the random lines model. Especially we consider what we have called the optimal fixed cell method (OFC). If there are n line-

segments an OEC structure contains O(n) cells and the expected number of segments intersecting each cell is fixed. This corresponds to the cell structure utilized for point files in[4]. Griffiths[9] and Franklin[7] have utilized a similar structure as a face-directory for the hidden line problem.

While we cannot obtain closed performance formulas for EXCELL (GP) leads to recursions suggesting the functional form of performance equations. The parameters are fitted from empirical simulations. Table 2 compares indicators of the OEC and EXCELL methods.

	EXCELL	OEC
NE	0.67b	0.67b (by definition)
PI	$1+1.2/\sqrt{b}+6/b$	$1+2.0/\sqrt{b}+2.0/b$(th)
PE	$\sqrt{n/b}(1.2+1/\sqrt{b})$	$\sqrt{n/b}(1.2+1.2/\sqrt{b})$(th)
1/EI(1)	$0.67b + 1.3\sqrt{b} + 1.1$	$0.67b + 1.3\sqrt{b} + 1.3$(th)

Table 2. Comparison of empirical formulas for some EXCELL and OEC performance indicators. The expected number of lines intersecting a cell is fixed as 0.67b in both methods. Notation:
- n: total number of line segments in the file
- b: bucket size
- NE: number of segments in a cell
- PI: number of cell intersections per segment
- PE: summed cell perimeter for the whole grid
- EI(l): efficiency of a segment-intersection query ("find data base segments intersecting a random test segment of a standard length"); i.e. the average proportion of intersections found to intersections tested
- th: theoretical

The main result is that for a stochastically homogeneous data distribution the EXCELL and OEC methods perform quite similarly. The greatest difference is the homogeneity of EXCELL: cells are rather uniformly filled and seldom empty whereas in the OEC method the cell contents vary extremely. For non-uniform distributions this difference is accentuated. Also the remarks on the storage efficiency of quad-trees apply here.

The PIPN-query cost depends on the number of cell accesses and within-cell operations. In[21] it was shown that even a cell size of 5 results in only 1.1 expected cell and directory accesses in the DT model. Comparable performance is achieved only by hashing type methods for exact match queries.

Thus EXCELL is especially efficient for well localized geometric access in external storage. Its main problem is the somewhat inefficient storage usage shown by indicators NE and PI of table 2.

Comparison with other methods

EXCELL is a general framework for data structures and algorithms so that comparisons with other methods should be made in a well specified context. However, based on the above sample algorithms and analyses we may draw some overall conclusions.

1. The efficiency of EXCELL is always comparable to that of the optimal fixed cell technique [4,7,9] and more uniform. For smooth data distributions performance depends only on bucket size b.
2. The directory makes the access efficiency of EXCELL clearly superior to that of quad-tree type methods[27,3,11,10]. Use of the directory depends on a time-space tradeoff but the data cells should for storage efficiency always be based on binary cell division as in EXCELL.
3. EXCELL has more efficient access than k-dimensional trees [8] but generally poorer storage utilization.
4. EXCELL generally leads to simpler algorithms than global projection methods [17,2,28]. Also, the latter methods are generally batch-oriented requiring a preprocessing phase. However, sometimes a "naive" use of projection methods[28] is simple and efficient.
5. The object composition tree approach [26,16,13] is more problem specific than EXCELL and may be tailored for great efficiency. Generally it does not contain access efficiency guarantees as EXCELL. Its storage and preprocessing efficiency is better. The two approaches may well be combined by utilizing EXCELL as a "bottom-up" geometric directory to an object composition tree.

These are crude generalizations because comparisons should be based on a specific task and data distribution. The good characteristics of EXCELL are accentuated when data structures reside in secondary storage and its algorithmic framework is simple:

> Because of the maximal size of cell-contents easy and robust naive algorithms may be utilized within cells. Overall efficiency is guaranteed by the directory and by having to consider only a restricted number of cells.

Applications

Geographic data bases initiated our study of EXCELL. The analysis of the PIPN-query shows that EXCELL provides efficient geometric access to a data base. Beside actual spatial queries the main need for geometric access derives from integrity constraints. Guarding these is a main requirement of geographic data management [25].

Integrity checking contains tasks like:
- check if a new point (having approximate coordinates) already exists in the data base
- check if a new edge intersects existing ones.

These require geometric access and make insertion 1-2 orders of magnitude more expensive than queries.

Geographic data processing is traditionally batch-oriented [10]. EXCELL makes possible a data-base environment with on-line algorithms for many tasks. Locality of access, paramount for efficiency in geographic data management, is also a main characteristic of EXCELL.

As seen from the ray-casting example EXCELL may be used to analyse geometric models. In ray-casting for visualization[16] it is important to efficiently find the first intersection on the ray. We have seen how EXCELL supports this.

In[23] we applied EXCELL to the hidden line problem

somewhat in the style of[9]. We obtained an efficient "O(n)" algorithm operating in external storage and thus making possible the treatment of scenes of unlimited complexity even on a mini-computer. The computational experience suggests the same efficiency level as reported state-of-the-art object space algorithms[9,28].

In geometric integrity checking[13] we inspect a new geometric element for non-intersection with existing ones. Interference detection[5] is similar. Efficient geometric access is essential.

Conclusions

EXCELL is above all a practical method for organizing geometric data. Practical means simple to implement and having good expected efficiency. Beside analysis and simulations this has been demonstrated by the operational hidden line system[23].

EXCELL is a general approach with many applications. It is a compromize between traditional full-resolution methods and image processing methods relying on a fixed pixel size. We have not here discussed the numerous ramifications and variations of EXCELL.

We have only referred to the theoretical analysis of EXCELL. However, we consider its results quite illuminating and the general method applicable to many fixed and variable cell methods.

ACKNOWLEDGEMENTS

The Finnish Academy funded this work. We thank Martti Mantyla, Heikki Saikkonen and Matti Tikkanen for valuable comments and help.

REFERENCES

1. J.L. Bentley and J.H. Friedman, A survey of algorithms and data structures for range searching, ACM Comp. Surv., Vol 11, No 4, 1979
2. J.L. Bentley and Th. Ottman, Algorithms for reporting and counting geometric intersections, Report CMU-CS-78-135, Dept. of Comp. Sci., Carnegie-Mellon University, 1978
3. J.L. Bentley and D.F Stanat, Analysis of range searches in quad trees, Info. Proc. Lett., 3(1975)6
4. J.L. Bentley, B.W. Weide and A.C. Yao, Optimal expected-time algorithms for closest point problems, ACM TOMS, vol 6, no 4, 1980
5. J.W. Boyse, Interference detection among solids and surfaces, CACM 22(1979)1
6. R. Fagin, J. Nievergelt, N. Pippenger and H.R. Strong, Extendible Hashing – a fast access method for dynamic files, ACM TODS, 3, 1979
7. W.R. Franklin, A linear exact hidden surface algorithm, ACM Computer Graphics, Vol 14, No 3, 1980
8. J.H. Friedman, J.L. Bentley and A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM TOMS, 6(1977)4
9. J.G. Griffiths, Eliminating hidden edges in line drawings, Comput. Aided Des., 11(1979)2
10. Proceedings of the Advanced Study Symposium on Topological Data Structures and Geographical Information Systems, Harvard University, 1977
11. G.M. Hunter and G. Steiglitz, operations on images using quadtrees, IEEE PAMI-1, 1979
12. D.E. Knuth, The art of computer programming, vol 3: sorting and searching, Addison-Wesley, Reading, Mass., 1973
13. M. Mantyla, Methodological background of the Geometric Workbench, Report-HTKK-TKO-B30, Helsinki University of Technology, Espoo, 1981
14. R.E. Miles A survey of geometrical probability in the plane, Computer Graphics and Image Processing, 12, 1980
15. F.P. Preparata, a new approach to planar point location, SIAM J. Comput., Vol 10, No 3, 1981
16. S.D. Roth, Ray casting for modeling solids, Computer Graphics and Image Processing 18(1982)2
17. M.I. Shamos, Computational Geometry, PhD thesis, Yale University,1978
18. I.E. Sutherland, R.F. Sproull and R.A. Schumacher, A characterization of ten hidden-surface algorithms, ACM Comp. Surv., Vol 6, No 1, 1974
19, 20, 21, 22, 23, 24, 25. M. Tamminen, Helsinki University of Technology, Espoo, Report-HTKK-TKK:
19. A20, The extendible cell method for fast geometric access
20. B29, Order preserving extendible hashing and bucket tries (to appear in BIT)
21. B27, Efficient spatial access to a data base
22. B28, Expected performance of some cell based file organization schemes
23. B34, Hidden lines using the EXCELL method
24. B35, The extendible cell method for closest point problems (to appear in BIT)
25. A23, Management of spatially referenced data
26. R.B. Tilove, Set membership classification: a unified approach to geometric intersection problems, IEEE trans. Computers, Vol C-29, No 10, 1980
27. J.E. Warnock, a hidden surface algorithm for computer generated halftone pictures, Computer Science department, University of Utah, 1969
28. M. Wittram, Hidden-line algorithm for scenes of high complexity, Comp. Aided Des., 13(1981)4