DESIGN METHODOLOGY OF BOUNDARY DATA STRUCTURES

S.R.Ala
Machine Vision Group, Center for Information Engineering,
City University, Northampton Square,London-EC1V OHB, U.K.

Abstract
In this paper we propose a universal data structure (UDS), termed as UDS, which will aid in the design of optimal bondary data structures.We later show, with the aid of some recently published data structures, that any data structure can be expressed as a special case of UDS. We demonstrate, how the application of the optimality concepts of the UDS, can lead us to the discovery of more efficient data structures than popular data structures. We also discuss the two approaches for optimization as proposed by [woo85]. We show that a globally optimal data structure is better than a special purpose optimal data structure.

I. INTRODUCTION

1.1 WHY STUDY DATA STRUCTURE EFFICIENCY?

Memory has become cheaper and processor speeds have galloped so isn't it pointless to continue researching into the storage and time efficiency? But the applications have also become more ambitious (e.g. performance simulation of aircraft models, which even with the fastest cray computers takes days, so not all possible models are tried).

In typical vision applications perceptual features (e.g.a group of parallel lines) are extracted from the image, which need to be matched with model data from CAD data base of the plant. The CAD data base has voluminous data from which we need to retrieve only a small set (e.g. extraction of the direction of all the edges meeting at a given vertex), in real time, to enable the robot to take prompt action to avert potential disasters. In the masonry tasking robot [Cha90] it is equally critical, to extract topological information from the CAD model of the building site, for obstacle avoidance and checking the progress of the brick laying process. The hardware is never powerful enough to keep the user happy [Ros89]. In the quest for real time execution of ambitious applications, data structure efficiency studies assume a more prominent role than the hardware speedup.

1.2 NEED FOR A UNIVERSAL DATA STRUCTURE:

Ever since the discovery of the classic Winged edge data structure [Bau75], there have been innumerable variations of it proposed. The word "edge" found qualified with all inconceivable adjectives, and we were inundated with newly coined data structures (e.g. Hybrid Edge [Kal89], Bridge Edge [Yam85], Half-Edge [Man88], Split Edge [Eas82], Doubly connected edge or DCEL [Pre85], Quad Edge [Gui85], Vertex-Edge and Face-Edge [Wei85]).

It is one the aims of this work to unite the multitude of these data structures under a common umbrella and present a comprehensive, yet clear view of the boundary data schemata. Also once we optimise the UDS, we would have answered the optimality problem of all the combinatorially possible data structures.

1.3 PREVIOUS WORK

The first discovery was the Winged Edge (WE) by Baumgart, sprung from the needs of model based computer vision. Edges were more reliably detectable than vertices and faces, from images. Hence winged edge was biased towards the edge, with all edge oriented topological information explicitly

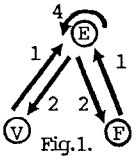


Fig.1.

stored as illustrated in Fig.1 (Note: In Figures 1, 3 and 4 the numbers on the arcs like $v \rightarrow^{1} E$ imply that for each vertex, one topologically adjacent edge is explicitly stored). Though it originated from the needs of computer vision it soon became sine-qua-non of all geometric modeling. However it underwent several modifications to meet the needs of various applications.

With regard to face, an edge has the dual roles of describing the face boundary as well as connecting two adjacent faces. In the winged edge a single edge record encodes the information about these dual roles. This necessitated an additional check for the direction of traversal of each edge , as each edge occurs in the traversal of two faces. To obviate from this time consuming check the concept of split edge (SE) was developed. An edge was thought to have two halves, each half partaking in describing one face only. Each half edge record has a pointer to the other half, to maintain the adjacency information of the two neighboring faces. Also a half edge has only one vertex associated with it, unlike the edge record of the WE, which has a two vertex array field. The WE and Split edge aimed at modeling solids or polyhedrons, in which each edge occurs in two faces.

However architectural applications involve not only solids but also unidirectional lines and polygons (e.g. a landscape needs a polygon to model not a two sided face). Kalay [Kal89] and Mantyla [Man88] augmented the SE by including an additional edge record, which has an array field of two half edge pointers. This information was contained in the opposite edge half field of each half edge record, in the SE. Thus half edge record, which was termed as segment record in [Kal89], can be used to model single sided polygons without redundancy. Thus two records were used in the SE (two half edge records per each edge record of WE) and three records (two half edge records and one edge record connecting the two half edges) in Hybrid Edge [Kal89] and Half-edge [Man88].

The original WE was extended to cater for multiply connected faces by the inclusion of the loop entity by Braid [Bra80]. Each face is a list of loops. Each loop is described by a set of vertices or edges. Note that the half edge described above aptly describes the loop as both loop and half edge are unidirectional, unlike the bidirectional face and edge. Multiple shell objects can be handled by the addition of the Cavity entity. Body and cavity have relation analogous to face and loop. Thus the three basic entities were augmented by three new entities: loop, cavity and half edge.

Woo [Woo85] studied the combinatorial analysis of the data structures and proposed a new data structure [Woo84],termed as Symmetric data structure (SDS). SDS has been proved to be more efficient than WE in both space and time and consequently became a popular choice of many current implementations. It has been extended to represent hierarchical feature based geometric modelers [Flo88,Fal89] and 3-D triangulation [Bru89]. A literal chinese version of [Woo84, 85] can be found in [Sun89]!

Winged triangle [Pao89] represents dimension independent polyhedra through simpilical decomposition. Non-manifold 3-polyhedra are mapped to a set of manifold polyhedral surfaces. [Hof88] is another example of current non-manifold data representation.

1.4 ORGANIZATION

In section 2 we propose the Universal data structure, which can be regarded as a complete generalization of the boundary data schema. We will also show that any data structure can be viewed as a

particular subset of the UDS. Section 3 attempts to optimise the UDS and shows how the optimality principles of the UDS can profitably be employed in the design of alternative data structures. In section 4 we investigate the two approaches for the design of an optimal data structure: global and special purpose optimal data structure, which were suggested by Woo..

II. UNIVERSAL DATA STRUCTURE

2.1 NOTATION

An upper case letter like E, denoting the entity, may refer either to the set of all edges or its cardinality, depending on the context. To refer to a particular instance of an entity we use a lower case letter.
The notation for a relation or mapping is an arrow between two entities, (e.g. f→V is the set of vertices adjacent to a given face 'f', $N_{f\to V}$ refers to the cardinality of the set,'N' without any subscript refers to average number of topological neighbors for any relation). [Woo84] has proved that N ≤ 6, for any solid.
Note: '*N*' will have a different meaning in Section 3.3.
The notation O(N) means 'of the order of N'

2.2 ENTITIES OF THE UDS

The three basic entities vertex, edge and face are augmented with the five entities Universe, Body, Cavity, Loop and Segment (denoted by U, B, C, L and S respectively). Universe is at the apex of the hierarchy of the UDS entities, as shown in Fig.2. Its meaning will be clear in Sect. 2.5.2.

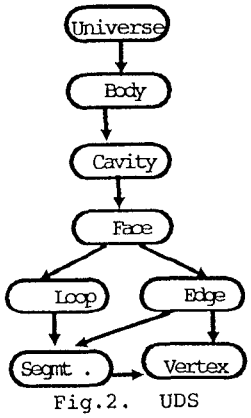


Fig.2. UDS

(Note that UDS is modelled by the complete graph of eight vertices while Fig.2 shows the hierarchy only).
Each body has its own set of cavities, faces, loops, edges, segments and vertices. Objects with internal voids can be conveniently represented by the 'C' entity.

2.3 RELATIONS

The possible number of relations in the UDS are 8×8 = 64, some of these relations, for example V→U, are trivial, since there is only one universe.

WE and several of its other derivatives have assumed that each edge has exactly four adjacent edges and each edge is shared exactly by two faces. These assumptions are equivalent to $N_{e\to E} = 4$ and $N_{e\to F} = 2$. They do not hold good for all cases of UDS. Since there is only one Universe and any edge has exactly two vertices and two segments, the following conditions hold good in any case of UDS.
$N_{b\to U} = N_{f\to U} = N_{l\to U} = N_{e\to U} = N_{s\to U} = N_{v\to U} = 1$, $N_{e\to V} = 2$ and $N_{e\to S} = 2$.

2.4 EULER'S FORMULA AS APPLIED TO UDS

Let F^i, V^i, E^i, C^i, H^i and L^i be the total number of faces, vertices, edges, internal cavities, through holes and loops respectively for the i'th body. If we restrict all the bodies to polyhedra with each edge being shared by two faces, we have

$$F^i+V^i-E^i = 2+2C^i - 2H^i + L^i, 1\le i\le B$$

For the UDS as a whole we have

$$\sum^B(F^i+V^i-E^i) = \sum^B(2 + 2C^i - 2H^i + L^i)$$

$$F+V-E = 2B + 2C - 2H + L \qquad (1)$$

Since E = 2S, the above Euler equation can also be formulated in terms of S

2.5 SPECIAL CASES OF UDS

We show that any data structure can be represented as a special case of UDS, with example data structures drawn from current

literature. The Half-Edge structure [Man88] has been chosen because it incorporates several of the key concepts behind the development of the UDS. It encodes all the entities of the UDS except the 'Cavity'. The other two, being the most specialized forms that have appeared in the literature, pose the greatest challenge to be represented as subsets of the UDS.

2.5.1 CASE 1: Winged Triangle (WT)

This [Pao89] is based on the triangulation of the faces and it is claimed that it is more compact than WE in the linearized representation of solids with curved boundaries.

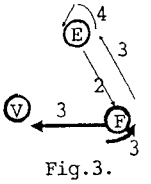


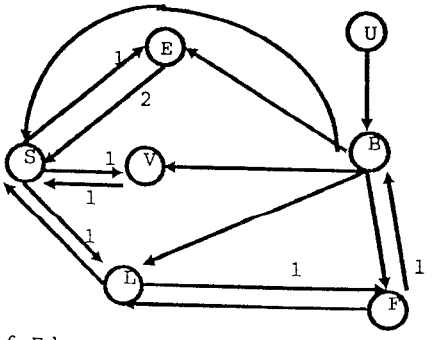
Fig.3.

By restricting UDS such that

(1) all faces are triangles,
(2) entities are limited to F, V,
(3) relations stored are limited to F→V and F→F (shown in bold arcs in Fig.3 and others shown in light arcs) and
(4) $N_{f\to F} = N_{f\to E} = N_{f\to V} = 3$, $N_{e\to E} = 4$, $N_{e\to F} = 2$ (these are illustrated by the numbers on the arcs in Fig.3)

we get WT as a special case of the UDS.

2.5.2. Case 2: Half-Edge data structure



Half-Edge as a special case of UDS
Fig.4.

The complete data structure from [Man88] is reproduced in the Appendix. Note that the 'Solid' and 'Half-Edge' correspond to the 'Body' and 'Segment' of the UDS. Accordingly in Fig.4, we use B and S, to represent the 'Solid' and 'Half-Edge'. The relation U→B is encoded by the *nextf* field of *solid* struct shown in the Appendix.

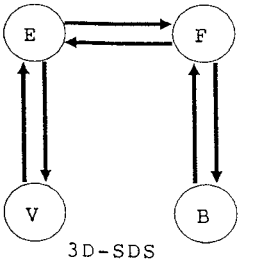
By restricting UDS such that

(1) all the entities except C are considered.
(2) relations stored are limited to B→F, B→L, B→E, B→S, B→V, F→B, F→L, L→S, L→F,S→E, S→L, S→V, E→S and V→S
(3) $N_{f\to B} = N_{l\to F} = N_{s\to E} = N_{s\to L} = N_{s\to V} = 1$ $N_{v\to S} = 1$, $N_{e\to S} = N_{e\to F} = N_{e\to V} = 2$, $N_{e\to E} = 4$

we get Half-edge data structure as a special case of the UDS.

2.5.3. CASE 3: 3D Symmetric data structure

An interesting extension of SDS to 3D triangulation was proposed [Bru89]. It uses an extra primitive T, the tetrahedron, in addition to the three primitive topological elements V, E, F. The 3D SDS stores 6 of the 16 adjacency relations of a 3D tessellation.



3D-SDS
Fig.5.

By restricting UDS such that

(1) all the bodies considered are tetrahedra,
(2) entities are limited to B, F, E, V,
(3) relations stored are limited to B→F, F→B, F→E, E→F, E→V and V→E and
(4) $N_{f\to E} = N_{f\to V} = 3$, $N_{f\to B} \le 2$, $N_{b\to B} \le 4$ $N_{b\to F} = N_{b\to V} = 4$, $N_{b\to E} = 6$, $N_{f\to F} \le 6$

we get the 3D-SDS as a special case of the UDS, as shown in Fig.5.

III. OPTIMIZATION OF UDS

3.1 METHODS OF OPTIMIZATION

We have 64 possible relations for the 8 entities of the UDS. It may be recalled

that WE has used the concept of fractional relations-whereby WE stored only one of the neighbors for the V→E and F→E relations. We thus have 64×2=128 potential number of possible relations for the 8 entities, including the fractional relations. The obvious approach for the optimization of UDS is the application of backtracking algorithm, involving comparison of the storage and time for all possible data schemata of UDS. The number of possible data structures for n entities is 2^{n^2}. The backtrack algorithm will have a time complexity $O(p(n)2^{n^2})$, where p(n) is a polynomial function of 'n'. For UDS n = 8, so we clearly have an algorithm not amenable even to the fastest CPU available. We therefore abandon the exhaustive enumeration approach and look for ways to reduce the search space for the optimal solution.

We can use the classical OR techniques [Wag75] for formulation and solution of the UDS optimization. An alternative approach uses the graph theoretic techniques [Har72]. We discuss in detail both these methods.

3.2 OR APPROACH

3.2.1. Problem formulation

Given a storage of M and the frequencies p_i of different queries, find the set of relations which have to be stored to minimize the total time required for the execution of the queries.

Let x_i = 1, if the i'th relation is explicitly stored
= 0, otherwise.

Let $1 \le i \le 64$ and $65 \le i \le 128$ represent the full relations and fractional relations respectively.

Note: In the above notation, x_i relation corresponds to its fractional relation x_{i+64}.

Let m_i = space required for the i'th relation.

Let t_i = time required for the execution of the i'th query once.

The classical OR formulation is

Minimize $\sum_i p_i t_i$ subject to (2)

$$\sum_i x_i m_i \le M \quad (3)$$

$$t_i = t_i(x_1, x_2, \ldots, x_{128}),\ 1 \le i \le 128$$

$$x_i\, x_{i+64} = 0,\ 1 \le i \le 64 \quad (4)$$

$$x_i = 1 \text{ or } 0,\ 1 \le i \le 128 \quad (5)$$

We have equation (4), since we can't have both the fractional relation and the corresponding full relation stored simultaneously.

A simple illustration of the formulation may be found in Section 4.2.

3.2.2. *Computation of m_i and t_i*

For clarity, we restrict UDS to be having only three entities: V, E and F. The possible relations, their index values i, for use in x_i, m_i and t_i and the storage m_i are tabulated below.

Rel.	V→V	V→E	V→F	E→V	E→F	E→E	F→V	F→E	F→F
i	1	2	3	4	5	6	7	8	9
m_i	2E	2E	2E	2E	2E	4E	2E	2E	2E

Note: m_i in the table were obtained as discussed in [Woo85]. (6)

Each t_i is a function of x_1, x_2,..., x_9.

Let us consider t_2 corresponding to V→E. It depends on x_2, x_3 and x_8 only (storing the other relations makes no difference).

Let k = time required for one operation (e.g arithmetic comparison). k includes CPU & main memory access time.

In C 'language like' code

```
If ( V→E is stored explicitly)
    t2 = 1 × k
else If ( V→F and F→E are stored explicitly)
    t2 = N^2 × k
/* An explanation may be found in [Ala90]*/
    else t2 = E × k
```

$$t_2 = x_2O(1) + (1-x_2)x_3x_8\ O(N^2) + (1-x_2)(1-x_3)(1-x_8)\ O(E) \quad (7)$$

It is thus possible to express each t_i in terms of $x_1,\ldots,x_{128}$.

3.2.3. Solution

Substituting for t_i in equation (2) and m_i in equation (3), with expressions similar to those obtained in equations (6) and (7) and applying Operations Research techniques (e.g. Non-linear programming) yields the *optimal* values, for the x_i, $1 \le i \le 128$.

3.3 GRAPH THEORETIC APPROACH

3.3.1 Problem formulation

The problem can be formulated as the selection of a sub-graph (sG) from the complete graph (CG) = (N,A) such that

$$sG = (sN \mid sN \subseteq N,\ sA \mid sA \subseteq A).$$

N and A denote the set of nodes (e.g. entities V, E) and arcs (e.g. relation E→F) respectively.

3.3.2. Optimality conditions

For the sG to be optimal

(I) its transitive closure $T(sG)$ must be CG.

$T(sG) = CG \Leftrightarrow$

1. $sN = N$
2. $\forall\ n_i \mid n_i \in N$ must be reachable from all nodes $n_j \mid n_j \in N$.

In other words the graph must be strongly connected.

(II) must have the least weighted path length. It can be easily shown that

1. $|A| = |N|^2$.
2. (a) There are $2^{|N|^2}$ possible sub-graphs of CG.

(b) CG contains $\binom{n}{i}(i-1)!$ cycles having number of arcs = i, where n = |N|

$$\text{Total number of cycles} = \sum_{i=2}^{n} \binom{n}{i}(i-1)! > (n-1)!$$

3. The transitive closure of an $mSG = (sG \mid sN = N)$, which has a Hamiltonian or an Eulerian trail is the CG.

The observation 3 leads to the following necessary conditions for an optimal sub-graph $mSG = (N,SA \mid SA \subset A)$

It must be a minimal sub-graph satisfying the two conditions below

1. Has a hamiltonian cycle or an Eulerian trail.
2. Has no self loops.

Let n^h and n^e be the number of Hamiltonian and Eulerian sub graphs of CG.
It can be proved that

$$n^h = (|N|-1)! \quad (8)$$

$$n^e = |N|!/2 \quad (9)$$

Thus we drastically reduce the search space for the optimal sub-graph from $2^{|N|^2}$ to $(1+N/2)(|N|-1)!$.

3.3.3 COMPARISON BETWEEN EULERIAN and HAMILTONIAN SUB GRAPHS

Lemma
Hamiltonian cycle is the cycle with the least path length for connecting a given number of points with directed arcs.

Proof
The min*imum* number of undirected arcs to connect n vertices is n-1 (analogous to acyclic chain). The min*imum* number of directed arcs to connect n vertices is n (analogous to a closed polygon). Also it is possible to construct a hamiltonian cycle of directed arcs around n vertices and the path length is n. Thus of all possible cycles for the CG, hamiltonian cycles require the minimum number of arcs.

We derive the formulae for the path lengths and *sum of the path lengths for each of the nodes to each of the nodes* for the two.

(1)Path lengths

It can be easily proved that for a hamiltonian sub-graph HSG(hN,hA)

$$|hA| = |hN|$$

It can be easily proved that for a Eulerian sub-graph ESG(eN,eA)

$$|eA| = 2\ (|eN|-1)$$

$$|eA|/|hA| = 2 - 2/|N| \qquad (10)$$

For any boundary graph $|N|$ varies from 3 to 8 (UDS). Thus the ratio of number of relations of a Eulerian to Hamiltonian sub graph varies from 4/3 to 7/4

(2) Total of the sum of the path lengths for each of the nodes to each of the nodes

Let p_{ij} = path length from node i to j for $1 \le i \le |N|, 1 \le j \le |N|$.

p_i = Sum of the path lengths for node i from each of the nodes

$$= \sum_{j=1}^{|N|} p_{ij} \text{ for } 1 \le i \le |N|$$

p = Total of the sum path lengths for all nodes $= \sum_{i=1}^{|N|} p_i$

We now derive formulae for p^h, p^e the totals for hamiltonian and eulerian sub graphs respectively.

$$p^e_{ij} = i-j\ ,\ j<i$$
$$= j-i\ ,\ j>i$$
$$= 2\ ,\ j=i$$

$$p^e_i = \sum_{j=1}^{i-1} (i-j) + \sum_{j=i+1}^{n} (j-i) + 2 \text{ for } 1 \le i \le |N|,$$
$$|N| = n$$

$$p^e = (n^3+5n)/3$$

$$p^h = n.n.(n+1)/2$$

$$p^h - p^e = (n^3+3n^2-10n)/6 \qquad (11)$$
$$= 4, \quad n=3$$
$$= 12, \quad n=4$$
$$= 44, \quad n=6$$

We discuss, how the above formulae can guide us to evaluate the Hamiltonian and Eulerian boundary data sub-graphs. The number of relations (path length in graph theoretic notation) $|A|$ in the sub-graph is a crude measure of the storage. The total of the path length for each of the nodes from each of the nodes is a crude indicator of the total time for all types of possible queries. Thus formulae (10) and (11) are crude indicators of relative storage and time respectively of Eulerian and Hamiltonian boundary data schemata.

However note that the number of relations $|A|$ doesn't constitute a valid criterion for the amount of storage, as some of the relations may require different storage than the rest (e.g.In the WE, E→E requires 4E storage while all other 8 relations require 2E each). The best criteria for the storage is thus the weighted path length of the graph, but to keep the discussion simple the formulae were restricted to crude measures.

3.4 APPLICATION OF THE OPTIMALITY CONCEPTS

We consider application of the above discussion to develop an alternative to the SDS, which is shown to require 25% less storage, in case 1. In cases 2 and 3 we evaluate the WT and the 3D-SDS, which were shown to be special cases of the UDS in Section 2.5.

3.4.1 CASE 1: An alternative to the SDS

From (8) the number of possible hamiltonian sub graphs with 3 entities is (3-1)! = 2.

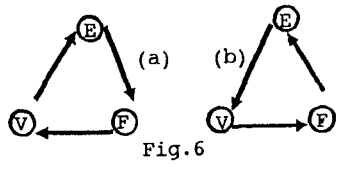


Fig.6

These two data structures are shown in Fig.6. It may be observed that they are both Δ shaped, one can be obtained from the other by reversing the direction of the arcs, hence we refer the first one as Δ and the other one as reverse Δ. Reverse Δ, which stores explicitly F→E,E→V, V→F is a hierarchical representation, each relation, except the last, being a mapping from an entity to its immediately lower level entity.

However, Δ is more efficient for extension

to multiply connected faces [Ala90]. It can be shown that the remaining 6 relations can be obtained from the three relations stored explicitly in time O(N). Woo has proved that the average value of N is less than 6. Thus clearly all the queries can be answered in nearly constant time as N « E.

From equation (9) the number of possible eulerian sub graphs with 3 entities is 3!/2 = 3.These three data structures are illustrated in Fig.7. Of these, the SDS [Woo85],illustrated in Fig.7(a), is the only one which has a hierarchical structure described before.

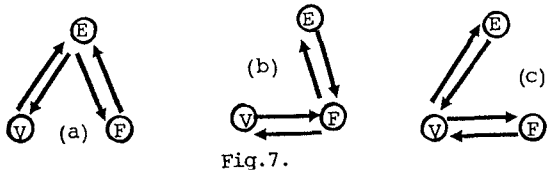


Fig.7.

Comparison between Δ and SDS
From equation (10) we find that the storage ratio for SDS and Δ is 4/3. It is clear that both have constant time access, but Δ requires 75% storage of SDS.

3.4.2 CASE 2: Winged Triangle
The number of triangles T is linear in E, hence the 2D-triangulation doesn't incur huge penalty like the 3D triangulation, in Case 3 below, which results in T, the number of tetrahedra, being quadratic in the number of vertices or edges.

Comparison of WT and Δ for space
The storage required for WT is 12(E-F+L) [Pao89]. The storage for Δ which has a total of L loops in its faces is 6E+F+L [Ala90]. Δ is space efficient if 6E > 13(F-L) + 2L. In certain situations (e.g: A cube with a square through hole has 10 faces, 2 loops, 24 edges and 16 vertices) Δ may be more space efficient than WT.

Comparison of WT and Δ for time
The two data structures are compared with respect to access time in Section IV.

3.4.3. CASE 3: Data structures for 3D-Triangulation

2D-Triangulation versus 3D-Triangulation:
It has been proved [Pre85] that a 3D triangulation of V points has $O(V^2)$ vertices and edges.Thus T and F are both $O(V^2)$, and E is O(V).Thus 3D SDS will have $O(V^2)$ storage complexity. Since E is O(V) the storage complexity can also be expressed as $O(E^2)$. The 3D-SDS has a quadratic storage complexity, compared to linear space complexity of Δ and SDS.

Extension of Δ to represent the 3D-Triangulation
Extension of Δ to represent the 3D-Triangulation will require an extra entity T,the tetrahedron. From (8) the number of possible Δ extensions are equal to the number of different hamiltonian sub graphs with 4 entities i.e.(4-1)! = 6. Of these 6 data structures, the schema of storing T→F,F→E, E→V and V→T is a hierarchical representation each relation being a mapping from an entity to its immediately lower level entity. This is similar to the reverse Δ, which is illustrated in Fig.8. The reason for not choosing Δ schema is that we don't have multiply connected faces in a 3D-triangulation.

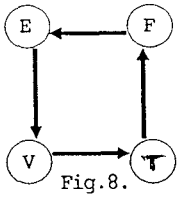


Fig.8.

It can be shown that the 12 relations,which are not explicitly stored, can be obtained from the four explicitly stored relations. As an example, let us obtain the E→F for the edge e. We first obtain the two end-points v_1, v_2 of the edge e by using the explicitly stored relation e→V. We then obtain $\{v_1 \rightarrow T\} \cup \{v_2 \rightarrow T\} = T_e = \{T_1,..,T_p\}$. For each of these $t \in T_e$ we find their faces. We thus have at most 4p faces, which may have e as one of their edges. We obtain for each of these faces 3 edges by the explicitly stored relation F→E. We thus have at most 12p edge comparisons to make. Thus clearly any query requires constant time as 12p « E

Extension of SDS to represent the 3D-Triangulation
From equation (9) the number of possible SDS extensions are equal to the number of different eulerian sub graphs with 4

entities i.e.4!/2 = 12. It may be noted that the 3D-SDS proposed in [Bru89] is one of these 12 possible extensions of the SDS. It has the hierarchical structure described in the preceding paragraph. Extraction of the 10 relations which were not stored explicitly, requires a constant number of operations for any constant relation or a number of operations which is linear in the output size for any variable relation.

Comparison between Δ and SDS for 3-D triangulation

Note that T= O(V) and F = O(V). Hence although Δ explicitly stores four relations the storage due to the two relations T→F and F→E is quadratic in V, while that due to the other two relations, E→V and V→T is linear in V. Hence the main storage is because of the 2 quadratic storage relations. Similarly in the SDS extension,it is because of 3 quadratic relations.

It is clear that both have constant time access, but Δ has approximately 2/3rd storage of SDS.

IV.GLOBALLY OPTIMAL VERSUS SPECIAL PURPOSE OPTIMAL DATA STRUCTURES

4.1. SPECIAL PURPOSE DESIGN

Woo suggested two approaches for the design optimal data structures. The first approach is the design of a general purpose data structure that is globally optimal. This was the approach adopted in the optimization of the UDS in Section 3.3. The second approach is the design of a special purpose data structure for a specific set of applications. This approach requires knowledge of p_i (the distribution of the topological queries), which may then be used for problem formulation and optimization, as described in Section 3.2.

For example, in the WT, the authors state that edges are never used by set operations, only in line drawing do they find some use. Based on this assumption the WT stores F→V and F→F only. Thus face based queries can be answered instantaneously, but the other 7 queries require linear time.

In a CIM environment several users will need access to information other than F→V and F→F. (e.g. in computer vision edge oriented queries are quite common). With WT each of these queries will require linear time, clearly an undesirable situation.

4.2. AN EXAMPLE

Let us assume that an application requires 8000, 1000, 999 and 1 numbers of queries of F→V, F→E, F→F and E→F respectively. This is representative of an application predominantly face based, but also exhibiting the realistic situation of an infinitesimal frequency of 0.01%, of the occurrence of the edge based queries.

Given the above number of queries let us find the best strategy of storing. Assume all the data resides in main memory, which is limited to 6E.

We can formulate the problem as in Section 3.2.1., with the restriction of 3 entities only, similar to Section 3.2.2., in which the meaning of x_i,m_i and t_i may be found. Proceeding along the lines of Section 3.2.3

$$\text{minimize } (t_5+8000t_7 +1000t_8 +999t_9) \quad (12)$$

subject to

$$(m_1x_1+ m_2x_2+ m_3x_3+ m_4x_4 + m_5x_5+m_6x_6+ m_7x_7+ m_8x_8+ m_9x_9) \le 6E \Rightarrow$$

(substituting for m_i from the table in the section 3.2.2.)

$$x_1+x_2+x_3+x_4+x_5+ 2x_6 + x_7+x_8+ x_9\le 3 \quad (13)$$

$$x_i= 1 \text{ or } 0 \text{ for } i = 1..9 \quad (14)$$

In equation (12), ignoring t_5 (in tune with the claim of Paoluzzi et.al.[Pao89]) we obtain the optimal solution as, $x_7 = x_8 = x_9 = 1$, $x_1 = x_2 = x_3 = x_4 = x_5= x_6 = 0$

This optimal solution stores the two relations F→V and F→F, of the WT, and an additional relation F→E, requiring 6E total storage. The three face based queries

require negligible time, however E→F requires file inversion, hence t_5 is O(E). Thus total time, which is approximately t_5, is O(E).

To sum up, time for Δ is O(N) which is independent of E while the time for WT is a multiple of E. As disused by Bently [Ben86] the coefficient of E even if very small does make it more expensive than a constant time Δ, in the asymptotic case. WT runs 99.99% in constant time, but is still not better than Δ, even in the event of occurrence of a small small probability (1 in 10,000) of a query (E→F in the example). Also *a special purpose data structure* like WT, *requires knowledge of query frequencies* which involves extensive data gathering and statistical analysis over a wide variety of user applications [Wei85].

V. CONCLUSIONS

A data structure which generalizes boundary data schema was proposed. Examples drawn from currently popular data structures were used to demonstrate the versatility of the proposed Universal data structure (UDS). Methods of optimization of UDS and their application, which lead to the discovery of efficient data structures were also discussed. It was shown that a global optimization approach based on UDS is superior to a special purpose data structure designed,for a specific set of applications.

REFERENCES

[Ala90] Ala S R, Data structures for computer vision, Internal report, Center for Information Engg, City Univ., London.

[Bau75] Baumgart B G, A polyhedron representation for computer vision, AFIPS National Computer Conference, 589-596, 1975.

[Ben86] Bentley J L, Programming Pearls, Addison-Wesley, 1986.

[Bra80] Braid I C, Notes on a geometric modeler, CAD group Document n.101 University of Cambridge, UK, 1980.

[Bru89] Bruzzone E, Defloriani L and Puppo E, Manipulating 3-D Triangulations, 3rd international Conf., FODO 1989 Proceedings, Foundations of data organization and algorithms edited by Litwin W, Schek H J, Springer Verlag, 339-353.

[Cha90] Chamberlain D A, West G A and Speare P R, A masonry tasking robot, J. of Mechatronics Systems Engineering, Kluwer Publishers, Dec. 1990.

[Eas82] Eastman C M, Introduction to Computer aided design Course Notes Carnegie Mellon University, Pittsburgh, PA, USA, 1979.

[Fal89] Falcidieno B and Giannini F, Automatic Recognition and Representation of Shape-Based Features in a Geometric Modeling System, Computer Vision Grap. and Image Proc., 48, 1989, 93-123.

[Flo88] Floriani L D and Falcidieno B, A hierarchical boundary model for solid object representation, ACM Trans. Graphics, 7, 1, 1988.

[Gui85] Leonidas Guibas and Jorge Stolfi, Primitives for the Manipulation of general subdivisions and the computation of Vornoi diagrams, ACM Transactions on Graphics, 4, 2, 1985, 74-123.

[Har72] Harary Frank, Graph Theory, Addision Wesley, 1972.

[Hof88] Hoffman C M, Hopcraft J E and Karasick M S, Towards implementing robust geometric computations, ACM Sym. Computational Geometry, 1988.

[Kal89] Kalay Y E
(1) The hybrid edge: a topological data structure for vertically integrated geometric modeling, Computer-aided design, 21, 3, 1989, 130-140.
(2) Modeling objects and environments, Wiley, New York, 1989.

[Man88] Mantyla, M., An Introduction to Solid Modeling, Computer Science press, 1988.

[Pao89] Paoluzzi A, Ramella M and Santarelli A, Boolean algebra over linear

polyhedra, Computer-aided design, 21, 8, 1989, 474-484.

[Pre85] Preparata F P, Shamos M I, Computational geometry: an Introduction, Springer Verlag, 1985.

[Ros89] Rosenthal D, More Haste-Less Speed, Proceedings of EUUG Spring'89-Brussels, 3-7 April, 1989, 123-130.

[Sun89] Sun Jiaguang, Chen Yujian , Data Structures for Geometric Modeling , Chinese Journal of Computing, 12, 3, 1989, 181-193.

[Wei85] Weiler K., Edge-based Data Structures for Solid Modeling in Curved-surface Environments, IEEE Computer Graphics and Applications, 5, 1, 1985, pp.21-40.

[Wag75] Wagner H M, Principles of Operations research, Prentice Hall, 1975.

[Woo84] Woo T C and Wolter J D, A Constant Expected Time, Linear Storage Data Structure for Representing Three-Dimensional Objects, IEEE Transactions on Systems, Man, and Cybernetics, 14, 3, 1984, pp.510-515.

[Woo85] Woo T C , A Combinatorial Analysis of Boundary Data Structure Schemata, IEEE Computer Graphics and Applications, 5, 3, 1985, pp.19-27.

[Yam85] Yamaguchi F and Tokieda T, Bridge edge and triangulation approach in solid modeling in Kunii T L(ed) Frontiers in computer graphics, Springer Verlag , 1985.

APPENDIX

HALF-EDGE DATA STRUCTURE
FROM [Man88]

```
struct solid
{
        Id              solidno;
        Face            *sfaces;
        Edge            *sedges;
        Vertex          *sverts;
        Solid           *nextf,*prevf;
};
struct face
{
        Id              faceno;
        Solid           *fsolid;
        Loop            *flout, *floops;
        vector          feq;
        Face            *nextf, *prevf;
};

struct loop
{
        HalfEdge        *ledg;
        Face            *lface;
        Loop            *nextl, *prevl;
};
struct edge
{
        HalfEdge        *he1, *he2;
        Edge            *nexte, *preve;
};
struct halfedge
{
        Edge            *edg;
        Vertex          *vtx;
        Loop            *wloop;
        HalfEdge        *nxt, *prv;
};
struct vertex
{
        Id              vertexno;
        HalfEdge        *vedge;
        vector          vcoord;
        Vertex          *nextv,*prevv;
};
```

ACKNOWLEDGEMENTS

I would like to thank T.J. Ellis and D. Chamberlain for support. Thanks also to the referees whose suggestions greatly improved the paper.