(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 February 2001 (01.02.2001)

PCT

(10) International Publication Number
WO 01/08263 A2

(51) International Patent Classification[7]: H01R

(21) International Application Number: PCT/SG00/00109

(22) International Filing Date: 27 July 2000 (27.07.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
9903605-5 28 July 1999 (28.07.1999) SG

(71) Applicants *(for all designated States except US)*: **NATIONAL UNIVERSITY OF SINGAPORE** [SG/SG]; 10 Kent Ridge Crescent, Singapore 119260 (SG). **LI, Xuetao** [CN/SG]; through Intro, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260 (SG). **TAN, Tiow, Seng** [MY/SG]; through Intro, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260 (SG).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **WOON, Tong, Wing** [SG/SG]; through Intro, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260 (SG).

(74) Agent: **APPLIED RESEARCH CORPORATION**; Kent Ridge, P.O. Box 1016, Singapore 911101 (SG).

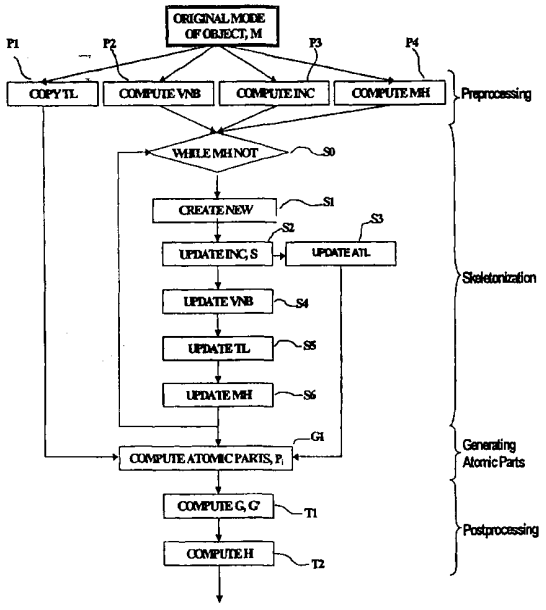
(81) Designated States *(national)*: JP, US.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published:
— *Without international search report and to be republished upon receipt of that report.*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR GENERATING ATOMIC PARTS OF GRAPHIC REPRESENTATION THROUGH SKELETONIZATION FOR INTERACTIVE VISUALIZATION APPLICATIONS



(57) Abstract: This invention presents a method to extract atomic parts of a graphics model using its skeleton. A skeleton is a fully collapsed body of the model, and is obtained through a novel way to contract edges of the model. For each connected part of the skeleton, an atomic part or feature which is a part of the model that is distinctively autonomous from its connected or neighboring body is formed. Next, atomic parts can be connected into a hierarchy depending on the eventual interactive visualization applications. The operation of the method includes the steps of interactively computing, displaying and recording skeleton, atomic parts, and object hierarchies in response to user commands to, for example, modifying skeleton, atomic parts or object hierarchies. Object hierarchies are useful to various applications such as object scene management, view-dependent simplification, mesh-mapping, morphing, and building bounding volume hierarchies.

WO 01/08263 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR GENERATING ATOMIC PARTS OF GRAPHIC REPRESENTATION THROUGH SKELETONIZATION FOR INTERACTIVE VISUALIZATION APPLICATIONS

FIELD OF THE INVENTION

The present invention relates generally to computer graphics and solid modeling, and more particularly, to methods and apparatuses for representing and displaying an object of its skeleton, atomic parts, and hierarchy.

BACKGROUND OF THE INVENTION

The quest for ease of modeling in three-dimensional computer graphics has led to the development of several well-known techniques, such as constructive solid geometry, free-form deformation, non-uniform rational B-splines, or more recently, implicit surfaces. Today's graphics hardware, however, is only capable of dealing with polygons efficiently. Hence, models are often tessellated before display for efficient rendering.

Related to 3D modeling is 3D model acquisition. The difficulty of obtaining a good 3D model has led to development in this area, using methods such as laser range scanners and turntable techniques. Such methods produce massive point sets representing points on the model's surface and require further processing such as 3D triangulation. The result is again a polygon mesh.

A tradeoff has to be established between having a high quality model with a high polygon count, and fast rendering with fewer polygons. The ever-increasing number crunching capability of today's processors tends to push the envelope of polygon count for efficient rendering, making it feasible to use more complex models.

Working on the premise of polygon mesh, however, leads to many difficulties. A polygon mesh is inherently unstructured, making it expensive to perform geometric operations such as intersection tests in collision detection and ray tracing. In the present invention, we are interested in object representation as in its object hierarchy.

The object hierarchy is most natural in terms of the human concept of shape. This has to do with the fact that cognition works best for hierarchically organized systems. In fact, 3D modelers often exhibit this phenomenon unknowingly when they organize an object in a top-down fashion. Although the object hierarchy is a natural representation of shape, it is often non-unique and designer-specific. Even so, the variations are usually minor and do not affect the conceptualization of the model on the whole. The present invention describes an algorithm for determining first a collection of atomic parts of an input polygon mesh and for determining a unique hierarchy from the atomic parts.

It is conceivably easy to obtain an object hierarchy from certain representations of models, for instance, constructive solid geometry models. However, the same cannot be said of geometric models, particularly B-rep models, which are by far the most prevalent. UCOLLIDE [TAN99] is a collision detection system that makes use of simplification to compute bounding volume hierarchies. The novelty of this work lies in the use of cluster-based simplification [LOW97] for extracting shape, and the use of this shape information for computing bounding volume hierarchies. Traditionally, bounding volume hierarchies are generated by top-down partitioning or bottom-up merging. Bottom-up methods only work well for organizing objects in a scene, and not polygons per se. Top-down methods perform poorly when the object to be partitioned consists of many sparsely arranged parts. Hence, we can conclude that it is hard to achieve optimal or near optimal bounding volume hierarchies using either method.

[TAN99] takes a radically different approach. Using simplification and shape analysis, the major components of an object can be extracted. Further shape analysis on the simplified model yields the components of the model. Partitioning on each component can then be done using traditional top-down methods. It is easy to see why this method is superior to previous ones like [GOTT96]. Firstly, the sum of bounding volumes in [TAN99] is smaller, due to more intelligent partitioning. Since the nodes of the hierarchy are organized by components, it is also easier to zoom into a particular region during intersection tests.

For complex models with a numerous interconnected parts, it is insufficient to use only one simplified model or also called level-of-details (LOD) for shape analysis. This is because it is difficult to pinpoint one LOD that captures all the essential

features of a model. By using a few LODs, the decomposition of a model can be guided along each node of the parent LOD and a hierarchy is naturally obtained. Although reasonably good results can be obtained using this method, there are some issues that remain to be addressed:

(i) the association of polygons in a lower LOD to a higher one may not be straightforward;

(ii) it is not clear what is the desired number of LOD and how to choose them;

(iii) different LODs produce different results; and

(iv) vertex clustering can cause topology change in the model. How this affects decomposition is unclear.

In addition, simplification using vertex clustering is not incremental. The algorithm needs to be invoked a number of times for LOD generation and this can cause a performance penalty. In light of all these issues, an alternative formulation of shape extraction is developed.

SUMMARY AND OBJECTS OF THE INVENTION

The invention described herein satisfies the above-identified needs and overcomes the limitations of the prior invention by [TAN99]. The present invention describes a new system and method to effectively generate object hierarchies, for purposes of various interactive applications such as:

(i) organize an arbitrary mesh for scene management and view-dependent simplification;

(ii) provide an alternative structure to one given by modeler;

(iii) visualization of complex models;

(iv) mesh-mapping; for morphing, establishing correspondence; and

(v) building bounding volume hierarchy (BVH) for intersection tests in collision detection and ray tracing

Specifically, disclosed herein is a method for execution by a data processor that generates effective object hierarchies. The method includes the following steps:

(1) **Preprocessing**. This is to prepare data structures for subsequent processes.

(2) **Skeletonization**. This is the process of deriving a skeleton of the input model,

where skeleton is a fully collapsed body of the model.

(3) **Generating Atomic Parts**. This is the process of obtaining various features (which is a part of the model that is distinctively autonomous from its connected or neighboring body) from the skeleton.

(4) **Postprocessing**. This process connects various atomic parts obtained into a hierarchy.

Further scope of applicability of the present invention will become apparent from the detailed description given hereinafter. However, it should be understood that the detailed description and specific examples, while indicating preferred embodiments of the invention, are given by way of illustration only, since various changes and modifications within the spirit and scope of the invention will become apparent to those skilled in the art from this detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will become more fully understood from the detailed description given hereinafter and the accompanying drawings which are given by way of illustration only, and thus are not limitative of the present invention, and wherein:

FIG. 1 is a block diagram of an exemplary raster graphics systems;

FIG. 2 is a simplified diagram of a graphics processing system according to the invention;

FIG. 3 is a flowchart that depicts the steps of the method of the invention;

FIG. 4 is an edge contraction example utilizing method as described in the invention; and

FIG. 5 is a flowchart that depicts the operation of the method of the invention.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 1 illustrates an exemplary raster graphics system that includes a main (Host) processor unit 100 and a graphics subsystem 200. The Host processor 100 executes an application program and dispatches graphics tasks to the graphics subsystem 200. The graphics subsystem 200 outputs to a display/storage device

300 connected thereto.

The graphics subsystem 200 includes a pipeline of several components that perform operations necessary to prepare geometric entities for display on a raster display/storage device 300. For the purposes of describing the invention, a model of the graphics subsystem is employed that contains the following functional units. It should be realized that this particular model is not to be construed in a limiting sense upon the practice of the invention.

A Geometric Processor unit 210 performs geometric and perspective transformations, exact clipping on primitives against screen (window) boundaries, as well as lighting computations. The resulting graphics primitives, e.g. points, lines, triangles, etc., are described in screen space (integral) coordinates.

A Scan Conversion (Rasterization) unit 220 receives the graphics primitives from the geometric processor unit 210. Scan converter unit 220 breaks down the graphics primitives into raster information, i.e. a description of display screen pixels that are covered by the graphics primitives.

A Graphics Buffer unit 230 receives, stores, and processes the pixels from the Scan Conversion unit 220. The graphics buffer unit 230 may utilize conventional image buffers and a z-buffer to store this information.

A Display Driver unit 240 receives pixels from the Graphics Buffer unit 230 and transforms these pixels into information displayed on the output display device 300, typically a raster screen.

FIG. 2 is a simplified diagram of a graphics processing system according to the invention. As shown in **FIG. 2**, an input device 10 inputs graphics data to be processed by the invention. The CPU 100 processes the input data from input devices 10 by executing an application program. CPU 100 also dispatches graphics tasks to the graphics subsystem 200 connected thereto. The output results (skeleton, atomic parts, and object hierarchy) may then be stored and/or displayed by display/storage devices 300.

Having described an exemplary graphics processing system that is suitable for

use in practicing the invention, a description is now provided of a method of extracting atomic parts from a polygon mesh and the arrangement of these parts into a hierarchy.

For the purposes of describing the invention, it is assumed that objects are described in polygonal boundary representations (b-reps). Models with curved surfaces are tessellated, i.e. approximated with planar triangles. This is to facilitate display as well as edge contraction, as will be described in the following section.

FIG. 3 shows a dataflow representation of the algorithm.

PREPROCESSING STEP

The purpose of preprocessing is to compute the necessary data structures required for subsequent processing. In particular, the following steps are performed: (P1) making a copy of the original tessellated triangle list, (P2) computing vertex neighborhood graph, (P3) computing incident edge table, and (P4) computing min heap for all the edges in the model. Notice that some of these steps, such as (P2) and (P3), are not necessary if input object representation comes with such information.

For the purpose of describing the preprocessing step, it is assumed that the original model is represented by a vertex table VT and a triangle list TL. The vertex table VT contains the supporting vertices used in the model. The triangle list TL contains the triangles used in the model and each triangle is represented as an ordered list of references (indices) to the vertex table VT. In addition, there may be other augmented structures such as facet normal table and vertex normal table, but these are normally not used. The contents of the vertex table VT and triangle list TL are volatile during the execution of the algorithm. In another embodiment, the original model can be represented as an edge-based data structure (instead of vertex-based as assumed in the description) such as the winged-edge data structure and quad-edge data structure [GUIB85]. It is straightforward for those skilled in the art to do the necessary adaptation from the following description to work on edge-based data structures.

(P1) Making a copy of the original tessellated triangle list

It is essential to make a copy of the original tessellated triangle list. This is because the edge contraction step performs in-place updating on the triangle list as new vertices are added. In one embodiment of this operation, the triangle list is used for rendering of intermediate results. When edge contraction is completed for all the edges in the model, the tessellated triangle list is restored so that the original model can be rendered.

(P2) Computing vertex neighborhood graph VNB

The vertex neighborhood graph VNB is used to store connection information between vertices. More specifically, if vertex v0 and v1 are neighbors, e.g. (v0,v1) of an edge of a triangle, then $v1 \in VNB(v0)$ and $v0 \in VNB(v1)$. The vertex neighborhood graph VNB is used extensively during the edge contraction step. A simple way of implementing VNB is to use a two-dimensional array where each row represents the list of vertices related to the vertex index. A more efficient method is to use an adjacency list.

(P3) Computing incident edge table INC

The incident edge table INC is another data structure used in edge contraction. The elements of the incident edge table INC are defined as follows: for any valid edge (v0, v1) in the model, INC(v0, v1) gives the list of triangles that are incident to this edge. The incident edge table INC is implemented using a hash table. A simple hashing formula could be given as follows:

$$\text{Key} = \begin{cases} v0 * \textit{displacement\_factor} + v1 & \text{if } v0 < v1 \\ v1 * \textit{displacement\_factor} + v0 & \text{otherwise} \end{cases}$$

displacement_factor is an arbitrary integer larger than the number of vertices in the model

(P4) Computing min heap MH for all the edges in the model

The min heap is a data structure whereby the smallest element or the element with the smallest weight is always at the top of the heap. Thus, it is easy to retrieve the smallest weight item from the min heap. It is also efficient to remove the smallest weight item whilst maintaining the property of the min heap. After computing

incident edge table INC, the list of all edges is easily known. The min heap is then obtained by adding all edges using the length of each edge as its weight. Thus, edges are retrieved in increasing order of length. In another embodiment of this method, the weight of each edge is computed based on some pre-calculated value of its vertices. These are variations that serve to improve the results and do not detract from the objective of this operation. The min heap is implemented using a linear array.

SKELETONIZATION STEP

Simplification is the process of obtaining a simpler form of a model while preserving certain attributes, such as appearance, as much as possible. In the context of geometric models, a simpler model is one with fewer numbers of triangles and vertices. While the applications of simplification are mainly in the arena of level-of-detail modeling, there have been ingenious applications like Collision Detection [TAN99], Progressive Mesh [HOPP96] and Skeletonization [DEUS99]. The approach taken by the former is based on the vertex clustering principle while the other two use edge contraction. Hoppe employs edge contraction for simplification and compression, with emphasis on appearance preservation. Deussen *et. al.*, in turn implements a simpler version of edge contraction for the purpose of defining intersection planes for non-photorealistic rendering.

The process of skeletonization in this invention is different in its methodology of edge contraction. Furthermore, the resulting skeleton is not the end product; rather the crux of the operation lies in the association of triangles into parts forming the skeleton.

A skeleton is defined as the result of a collapsed model. Edge contraction refers to the process of collapsing edges into vertex. For the purpose of describing the invention, assume that the current edge to be collapsed is represented by (v0,v1) where v0 and v1 are the vertices of its endpoints. **FIG. 4** illustrates this process. Unless otherwise stated, edge contraction is performed until the min heap MH of all edges is empty (S0). An additional data structure is required in this method to

store associated triangles and this is known as the associated triangle list ATL (which is initialized to empty set at the beginning of the skeletonization step). The detailed steps in edge contraction consist of: (S1) creation of new vertex, (S2) updating of associated triangle list ATL, (S3) updating of incident edge table INC, (S4) updating of vertex neighborhood graph VNB, (S5) updating of triangle list TL, (S6) updating min heap MH and skeleton S.

(S1) Creating new vertex

Given v0 and v1 are vertices to be collapsed, a straightforward method of creating a new vertex vn would be (v0+v1)/2. However, this does not take into consideration the relationship of v0, v1 and the skeleton S. In the present invention, the new vertex is given the coordinate of v0 or v1 if either v0 or v1 belong to an existing edge in skeleton S. The new vertex vn is appended at the end of the vertex table VT.

(S2) Updating of associated triangle list ATL

Since v0 and v1 are vertices of at least one triangle, the collapsing of v0 and v1 would cause at least one triangle to be removed from the model. In addition, the change of vertex numbering, i.e. v0 $\rightarrow$ vn and v1 $\rightarrow$ vn, violates the validity of the associated triangle list ATL. The purpose of this step is to keep track of the removed triangles, as well as to update the vertex number in the associated triangle list ATL. The following algorithm exhaustively enumerates the types of updating involved:

(i) for each vertex v* in VNB(v0) \ {v1}, rename ATL(v0, v*) as ATL(vn, v*)

(ii) for each vertex v* in VNB(v1) \ {v0}, rename ATL(v1, v*) as ATL(vn, v*)

(iii) for each triangle T in INC(v0, v1) with v2 be its third vertex, do

store T to ATL(v2, vn)

store ATL(v0, v2) and ATL(v1, v2) to ATL(v2, vn)

store ATL(v0, v1) to ATL(vn, vn)

store ATL(v0, v0) and ATL(v1, v1) to ATL(vn, vn)

(S3) Updating of incident edge table INC

The collapsing operation (v0, v1) $\rightarrow$ vn requires that the incident edge table INC

be updated. The algorithm for performing this update is given as follows:

(i) for each vertex v* in VNB(v0) \ {v1}, rename INC(v0, v*) as INC(vn, v*)

(ii) for each vertex v* in VNB(v1) \ {v0}, rename INC(v1, v*) as INC(vn, v*)

(iii) for each triangle T in INC(v0, v1) with v2 be its third vertex, do

delete T from INC(v2, vn)

(iv) remove INC(v0, v1)

(S4) Updating of vertex neighborhood graph VNB

As in step (S3), it is mandatory to maintain the validity of vertex neighborhood graph VNB after the collapsing operation. The algorithm for this update is as follows:

(i) add VNB(vn) = VNB(v0) $\cup$ VNB(v1) \ {v0, v1}

(ii) delete VNB(v0) and VNB(v1)

(iii) for each vertex v* in VNB(vn), update VNB(v*) = VNB(v*) \ {v0, v1} $\cup$ {vn}

(S5-S6) Updating of triangle list TL, min heap MH and skeleton S

The triangle list TL is updated so that collapsed vertices are renamed to vn. Note that if a high fidelity rendering is required, the affected normals in facet normal table and vertex normal table are to be recomputed. For the sake of efficiency, the updating of min heap MH and skeleton S are also performed under this step. It is not efficient to update MH directly, as this would involve examining the entire heap. Instead, only affected edges are reinserted into the min heap MH. When an edge is retrieved, its validity can be ascertained by checking against the incident edge table INC.

The algorithm for performing this update is as follows:

```
for each vertex vi in VNB(vn), do
    if INC(vn, vi) is not empty
        for each triangle T in INC(vn, vi), do
            find and rename vertex v0 or v1 in T to vn
        let wt = f (vi, vn)      where f could be the length function
        heap_insert(MH, edge(vi, vn), wt)
    otherwise
        add (vn, vi) to S
```

GENERATING ATOMIC PARTS

Having obtained the skeleton and the data structure ATL, the next step would be to compute the list of atomic parts associated with the model (step G1). An atomic part is defined as the collection of triangles that represent a connected skeleton. It is obtained from the following derivation:

Let R be a relation on S such that aRb (i.e. a related to b) if there exists a path from a to b in S.

R is an equivalence relation under S.

Obtain the set of equivalence classes of R, called Q_i where i=1 to m.

For each Q_i, an atomic part $P_i = \cup$ ATL(vj, vk) where (vj, vk)$\in S_i$ and vj, vk$\in Q_i$

POSTPROCESSING STEP

The objective of post-processing is to derive the object hierarchy representing the original model M by organizing the atomic parts in a meaningful way. The object hierarchy is defined as a rooted tree where the root node is the original model M and each level below it is a breakdown of the node into mutually exclusive parts. The leaf nodes are atomic parts obtained in previous step. Hierarchy construction makes use of the relationship between connected atomic parts and this relationship is defined by the connection graph G as follows:

$$G = \{ (P_i, P_j) \mid \exists\, t_p \in P_i,\ t_q \in P_j \text{ and } t_p, t_q \text{ shares an edge} \}.$$

For models containing disjointed parts, the connection graph G is also disjointed. In that case, each connected subgraph is treated as an independent connection graph and is dealt with separately. Spatial analysis (see [TAN99]) could later be applied to join them up into a hierarchy. For the rest of the discussion on hierarchy building, G is assumed connected.

G is undirected and could possibly contain cycles. The presence of cycles could create problems for some hierarchy construction methods. Hence it is necessary to first transform the graph into a tree. One way of doing this, in step T1, is to compute the minimal spanning tree (MST) G' of G. The cost of each edge in G could be the some function, such as cosine, of the angle formed by orientations of

the endpoints. The cost function is chosen as such because a sharp angle formed by the parts indicate that they probably should not be joined in the first place.

One way of computing an orientation would be to use the axis of least inertia, which is fairly expensive to compute in three-dimensions. Alternatively, the orientation is approximated by the largest eigenvector in the principal component analysis (PCA) of the object.

Having obtained G' for each connected subgraph of G, hierarchy construction can begin (step T2). Three types of hierarchical construction method are disclosed in the following paragraphs: (1) unrooted folding method (2) rooted folding method (3) common edge method. The use of a particular method may depend on the eventual applications of the constructed hierarchy.

(1) Unrooted folding method

This method applies to the default configuration, in which *G'* is unrooted. Nodes of degree n ($n > 1$) with at least ($n - 1$) leaf neighbors are tagged. At each step of the algorithm, tagged nodes are collapsed to form a new node in the object hierarchy *H*. The algorithm for building *H* is as follows:

```
add P1, P2, ..., Pm to H
while | G' | > 1
        add nodes of degree n with at least (n-1) leaf neighbors to a queue, Q
        while P = dequeue(Q) is not empty
                create a new node M' in H to be parent of leaf neighbors of P
                create a new node M'' in H to be parent of M' and P
                delete leaf neighbors of P from G'
                rename P to M'' in G'
```

(2) Rooted folding method

The unrooted folding method gives a simple way of constructing the hierarchy, without a priori knowledge of the model. However, the hierarchy constructed this way may be lopsided and not really reflect the natural partitioning of the model. A better way would be to pick a representative node in *G'* to be the root. A representative node could be the node with the largest bounding volume (representing a dominant part), or one with the highest degree (representing a

center of focus), or could be selected manually by the user. Hierarchy construction would be similar as before, except that it is now done bottom-up. The rooted folding method for building H is as follows:

add $P_1, P_2, \ldots, P_m$ to H
while $|G'| > 1$
    let h be the height of G'
    add nodes at level (h-1) to Q
    while P = dequeue(Q) is not empty
        create a new node M' in H to be parent of leafs of P
        create a new node M'' in H to be parent of M' and P
        delete leafs of P from G'
        rename P to M'' in G'

(3) Common edge method

For geometric objects with many interconnected parts it may be difficult to choose a representative node to be the root. In that case, it may be useful to have a metric for evaluating the "strength" of connection between parts. This metric should measure the relative importance of an atomic part and its neighbors. In one embodiment, the metric that is used in this method is the relative length of the common edges between two parts and is defined as follows:

$common(P_i, P_j) = \Sigma \, || (v_a, v_b) ||$ s.t. (v_a, v_b) in t_k where $t_k \in P_i$ and (v_a, v_b) in t_l where $t_l \in P_j$

$boundary(P_i) = \Sigma \, || (v_a, v_b) ||$ s.t. (v_a, v_b) in t_k where $t_k \in P_i$ and not $t_l \in P_i$ s.t. (v_a, v_b) in t_l

$relative(P_i, P_j) = common(P_i, P_j) / \min(boundary(P_i), boundary(P_j))$

The method of construction is to keep removing edges based on a function of its metric. Such function may either be taking, in one embodiment, the largest, or, in another embodiment, the smallest metric. The algorithm for building H is as follows:

add $P_1, P_2, \ldots, P_m$ to H
while $|G'| > 1$
    get (P_i, P_j) from G' with f ($relative(P_i, P_j)$) where f could be max or min function
    create a new node M' in H to be the parent of P_i and P_j
    collapse P_i and P_j to new node M' (corresponding to M' in H) in G'

The above completes the description of the hierarchy building process. As mentioned in the summary and objects of the invention, there are various interactive applications that can benefit from the computed object hierarchy. In the following, we describe explicitly the use of the object hierarchy for one such application; other applications within the spirit and scope of the invention will become apparent to those skilled in the art from such a discussion.

APPLICATION TO COLLISION DETECTION

One of the most time consuming operation in collision detection is the object intersection test. This is also true for the ray intersection test in ray-tracing. It is generally accepted that a tight bounding volume is good for such tests since it will ensure that pruning is done as soon as possible. Recent developments in this area lead to efforts such as BOXTREE [BARE96], OBBTree [GOTT96] and *k*-DOP [KLOS98], to name a few. Most of them either take the top-down approach of partitioning or the bottom-up approach of merging, with no inherent interest in the shape of the model as a whole. Top-down partitioning is ineffective for models that are sparse, as the level of representation is too coarse at the top levels. Bottom-up methods suffer from efficiency problems since obtaining a globally optimal solution is prohibitively expensive.

As described earlier, UCOLLIDE [TAN99] uses a series of simplified models to extract parts of models, hence taking advantage of their shape. However, it is not clear how many simplified models to use or how to select them to get the best result.

This invention presents an automated way of obtaining the object hierarchy that is easily convertible into a bounding volume hierarchy suitable for collision detection and ray tracing. The atomic parts $P_1, P_2, \ldots, P_m$ which form the leaf nodes of H are in general simple shapes. Hence top-down partitioning methods can be applied effectively to reduce P_i to leaf nodes containing a single or small number of triangles.

For a disjoint connection graph G, the MST of each connected subgraph G_i forms a hierarchy H_i representing the object hierarchy of congregate part R_i, the

collection of polygons of the nodes in G_i. Let *bounding_box*(R_i) be the size of the bounding volume of R_i. The choice of bounding volumes is application dependent and transparent to the algorithm. The algorithm for merging the various H_i makes use of pair-wise merging of smallest enclosing bounding volume to obtain a locally optimal solution.

while | R | > 1

get R_i , R_j from R with the smallest *bounding_box*($R_i \cup R_j$)

create a new node M' in H to be the parent of H_i and H_j

collapse R_i , R_j to new node M' (corresponding to M' in H) in R

This manner of hierarchy building is similar in nature to existing bottom-up methods described in [BARE96] and [TAN99].

The resulting tree H may be reorganized by picking a new root to achieve a more balanced result. To control the branching factor, H may also be converted into a binary tree. Note that this is done on a per node basis, and does not involve alteration to the original model M.

It should be pointed out that this invention lends itself well as a plug-in replacement for component derivation in [TAN99]. The process of converting the component tree into a BV hierarchy is equally applicable to the object hierarchy.

OPERATION OF THE METHOD OF THE INVENTION

FIG. 5 is a flowchart that depicts the operation of the method of the invention in obtaining object hierarchies. First, the system loads or generates a database of objects appearing in a scene (M1). These objects may be elements of, for example, a CAD database or a virtual environment. For each of these objects, in step M2, skeleton, atomic parts, and object hierarchies are generated as detailed in **FIG. 3**. The input models and the skeletons, atomic parts, and object hierarchies are stored in the memory as shown in **FIG. 2**. At this point, the system is ready to begin an interactive display session with a user.

User can issue commands to, for example, display skeleton, atomic parts, or object hierarchies. Also, user can interactively do adjustment to modify skeletons, atomic parts or object hierarchies, and the system accesses the memory to obtain

the suitable data structures for modification and to pass them to the graphics subsystem for display of the data structures on the display screen. Additionally, in response to user requests M4, the system can use skeletons, atomic parts, and object hierarchies to compute other data structures such as the bounding volume hierarchies, as described earlier, for use in ray-tracing or virtual environment navigation to determine collisions M5.

The invention being thus described, it will be obvious that the same may be varied in many ways. Such variations are not to be regarded as departure from the spirit and scope of the invention, and all such modifications as would be obvious to one skilled in the art are intended to be included within the scope of the following claims.

REFERENCES

[BARE96] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell and A. Tal, "*BOXTREE: A Hierarchical Representation for Surfaces in 3D*", Proceedings Eurographics '96, Vol. 15(3), August 1996, pp. C-387-396, C-484.

[DEUS99] Deussen O., Hamsel J., Raab, A., Schlechtweg, S., Strothotte T., "*An Illustration Technique Using Hardware-based Intersections and Skeletons*", to appear in Proceedings Graphics Interface, June 1999.

[GOTT96] S. Gottschalk, M.C. Lin and D. Manocha, "OBBTree: A Hierarchical Structure for Rapid Interference Detection", Computer Graphics (SIGGRAPH '96 Proceedings), 1996, pp. 171-179.

[GUIB85] Guibas and Stolf, "*Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*", ACM Transaction of Graphics, vol. 4, (1985), 74—123.

[HOPP96] H. Hoppe, "*Progressive Meshes*", SIGGRAPH 96 Conference Proceedings, Annual Conference Series, Addison-Wesley, August 1996, pp. 99-108. See also European Patent Documents EP0789330A2, entitled *Selective Refinement of Meshes*.

[KLOS98] J. Klosowski, M. Held, J. Mitchell, H. Sowizral and K. Zikan, "*Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs*", IEEE Transactions on Visualization and Computer Graphics, vol. 4 (1), 1998, pp. 21—36.

[LOW97] Kok-Lim Low and Tiow-Seng Tan, "*Model Simplification Using Vertex-Clustering*", Proceedings on Symposium On Interactive 3D Graphics, 1997, pp. 75-81.

[TAN99] Tiow-Seng Tan, Ket-Fah Chong and Kok-Lim Low, "*Computing Bounding Volume Hierarchies Using Model Simplification*", Proceedings on Symposium On Interactive 3D Graphics, 1999, pp. 63-69.

CLAIMS

Having thus described our invention, what we claim as new, and desire to secure by Letters Patent are:

1. A method for execution by a data processor to process a model having a collection of triangles representing an object into atomic parts, comprising:

preprocessing the model to generate data structures for subsequent processing;

generating a skeletal structure of the model; and

generating atomic parts of the model.

2. The method as set forth in claim 1, said preprocessing step computing a vertex neighborhood graph VNB, an incident edge table INC, and a min heap MH of edges.

3. The method as set forth in claim 2, further comprising: maintaining edges in the min heap MH such that an associated weight of an edge is a length of the edge and an edge having the smallest weight is maintained on a top of the min heap MH.

4. The method as set forth in claim 2, further comprising: maintaining edges in the min heap MH such that a weight of an edge is some pre-calculated value of the edge's endpoints (vertices) and an edge having the smallest weight is maintained on a top of the min heap MH.

5. The method as set forth in claim 1 wherein said step of generating a skeletal structure of the model comprises:

letting edge (v0, v1) be a min weight edge from MH,

collapsing (v0, v1) into a new vertex vn;

appending vn at an end of a vertex table VT;

updating associated triangle lists ATL for all affected vertices and edges;

updating an incident edge table INC for all affected edges;

updating an vertex neighborhood graph VNB for all affected vertices;

updating a triangle list TL, the min heap MH and a skeleton S where necessary; and

repeating the above steps if necessary, until MH is empty.

6. The method as set forth in claim 5, further comprising creating a new vertex at either a midpoint of the edge being collapsed or snapping it to one of the two endpoints of the edge when the endpoint belongs to some existing edges in the skeleton computed so far.

7. The method as set forth in claim 5 wherein said step of updating skeleton S includes a step of determining that a skeletal branch is to be added when an edge collapse causes incident triangle list INC to become empty.

8. The method as set forth in claim 1 wherein the step of generating atomic parts of the model includes a step of collating triangles in connected skeletal branches.

9. The method as set forth in claim 1, further comprising:

computing a connection graph G of the atomic parts;

computing a minimal spanning tree (MST) G' for each connected subgraph of G; and

computing an object hierarchy H from each MST G'.

10. The method as set forth in claim 9 wherein the step of computing object hierarchy from each MST uses an unrooted folding method where immediate neighbors of leaf nodes are tagged and collapsed at every step of execution.

11. The method as set forth in claim 9 wherein the step of computing object hierarchy from each MST uses a rooted folding method where nodes are folded in a bottom-up manner.

12. The method as set forth in claim 11, further comprising: picking a representative node to be the root using the largest node or the node with the highest degree.

13. The method as set forth in claim 9 wherein the step of computing object

hierarchy from each MST uses a common edge method where nodes are collapsed based on a function, such as max or min, of their relative metric.

14. The method as set forth in claim 13, further comprising computing a metric for measuring a relative weight between adjacent atomic parts as the ratio of the total length of common edges to total length of the smaller boundary of the 2 adjacent atomic parts.

15. The method as set forth in claim 9 wherein the step of computing G' for each connected subgraph of G includes a step of using the cosine of the orientation of atomic parts as the cost function.

16. The method as set forth in claim 15, wherein the step of computing the cosine of the orientation of atomic parts includes the step of computing the orientation from the axis of least inertia.

17. The method as set forth in claim 15, wherein the step of computing the cosine of the orientation of atomic parts includes the step of computing the orientation from the largest eigenvector in the principal component analysis of the model.

18. The method as set forth in claim 9, further comprising:

subdividing the atomic parts in the object hierarchies until each node consists of a single triangle or a small number of triangles; and

combining the object hierarchies pairwise until a single bounding volume hierarchy is obtained,

wherein the single bounding volume hierarchy encompasses individual objects or assemblies thereof.

19. Graphics display apparatus for displaying and computing skeletons, atomic parts, object hierarchies, comprising:

memory means for storing a model of an object and for storing skeleton, atomic parts, object hierarchy;

means, responsive to a command from a user of the apparatus, for indicating adjustment to modify skeleton, atomic parts, or

object hierarchy; and

processing means, coupled to and responsive to the indicating means, for accessing the memory means to display either the model, skeleton, atomic parts, or object hierarchy.

20. Graphics display apparatus as set forth in claim 19 wherein said processing means includes means for processing the model of the object, as set forth in claim 1 and claim 9 so as to produce and store skeleton, atomic parts, and object hierarchy.

21. A graphics display apparatus for displaying and computing a bounding volume hierarchy, comprising:

memory means for storing a model of an object and for storing a skeleton, atomic parts, an object hierarchy, and a bounding volume hierarchy;

means, responsive to a command from a user of the apparatus, for indicating adjustment to modify the skeleton, the atomic parts, the object hierarchy, or the bounding volume hierarchy; and

processing means, coupled to and responsive to said indicating means, for accessing said memory means to display either the model, the skeleton, the atomic parts, the object hierarchy, or bounding volume hierarchy.

22. The graphics display apparatus as set forth in claim 21 wherein said processing means includes means for processing the model of the object into atomic parts including:

means for preprocessing the model to generate data structures for subsequent processing;

means for generating a skeletal structure of the model; and

means for generating atomic parts of the model

23. The graphics display apparatus as set forth in claim 22 wherein said processing means further includes:

means for computing a connection graph G of the atomic parts;

means for computing a minimal spanning tree (MST) G' for each

connected subgraph of G; and

means for computing an object hierarchy H from each MST G'.

24. The graphics display apparatus as set forth in claim 22 wherein said processing means further includes:

means for subdividing the atomic parts in the object hierarchies until each node consists of a single triangle or a small number of triangles; and

means for combining the object hierarchies pairwise until a single bounding volume hierarchy is obtained,

wherein the single bounding volume hierarchy encompasses individual objects or assemblies thereof.

FIG. 1

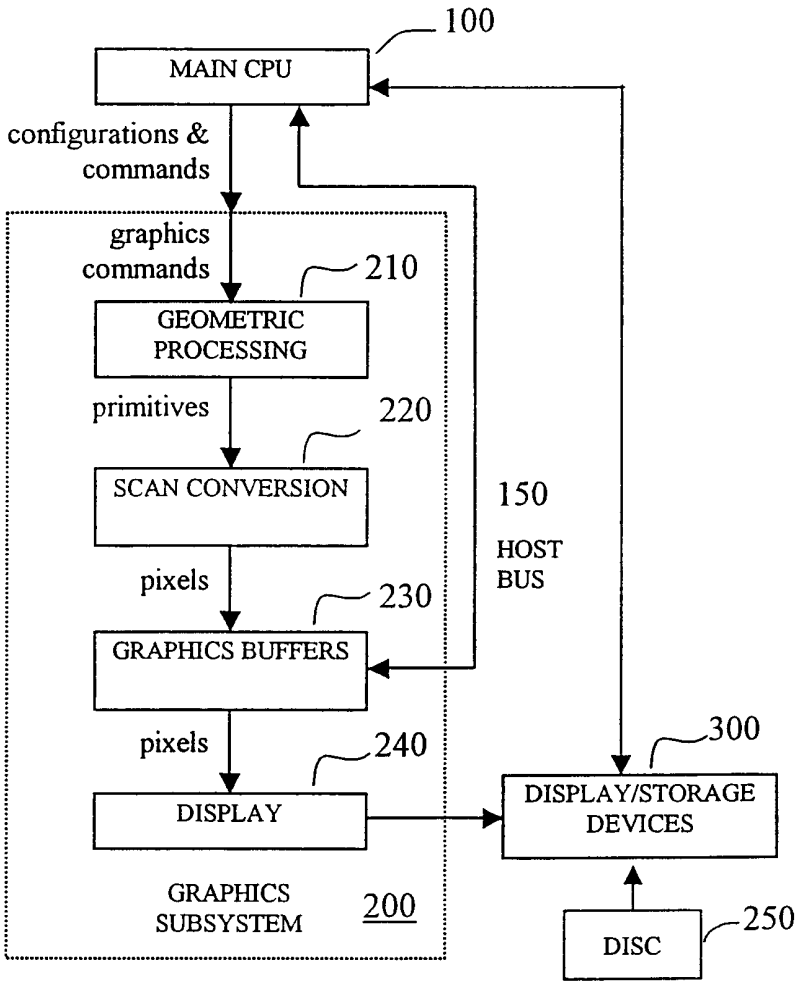

100
MAIN CPU
configurations & commands
graphics commands
210
GEOMETRIC PROCESSING
primitives
220
SCAN CONVERSION
150
HOST BUS
pixels
230
GRAPHICS BUFFERS
pixels
240
DISPLAY
300
DISPLAY/STORAGE DEVICES
GRAPHICS SUBSYSTEM
200
DISC
250

FIG. 2

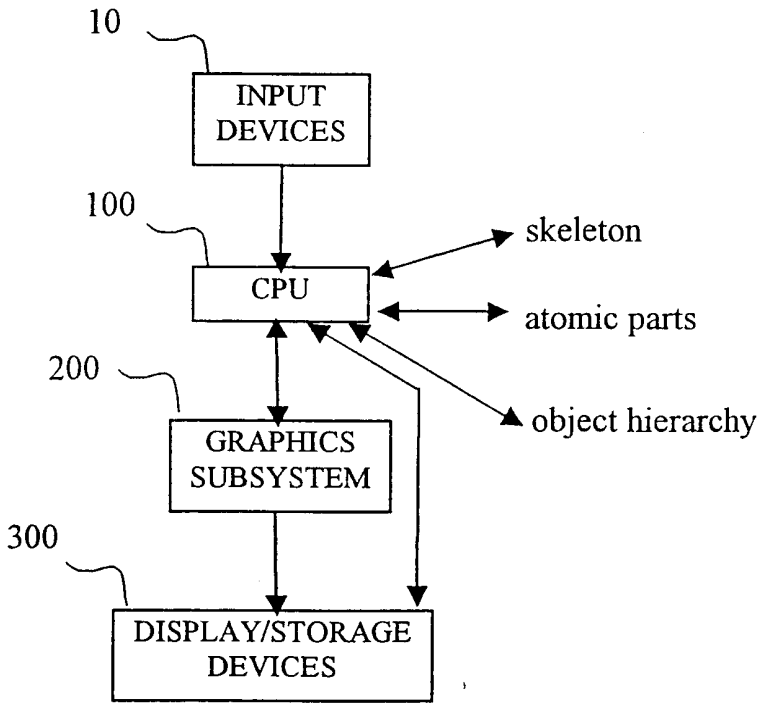

10
INPUT DEVICES
100
CPU
skeleton
atomic parts
object hierarchy
200
GRAPHICS SUBSYSTEM
300
DISPLAY/STORAGE DEVICES

FIG. 3

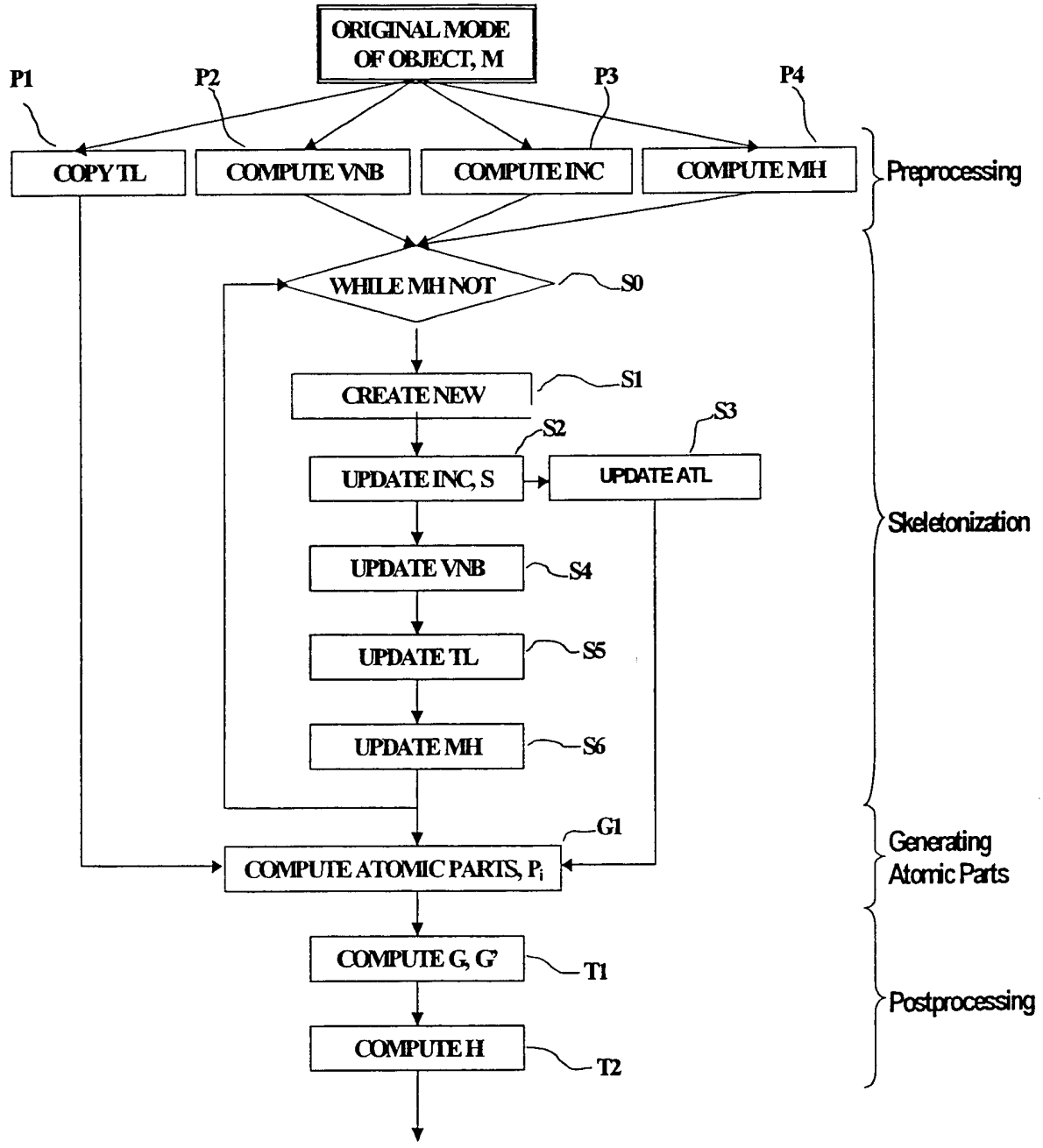
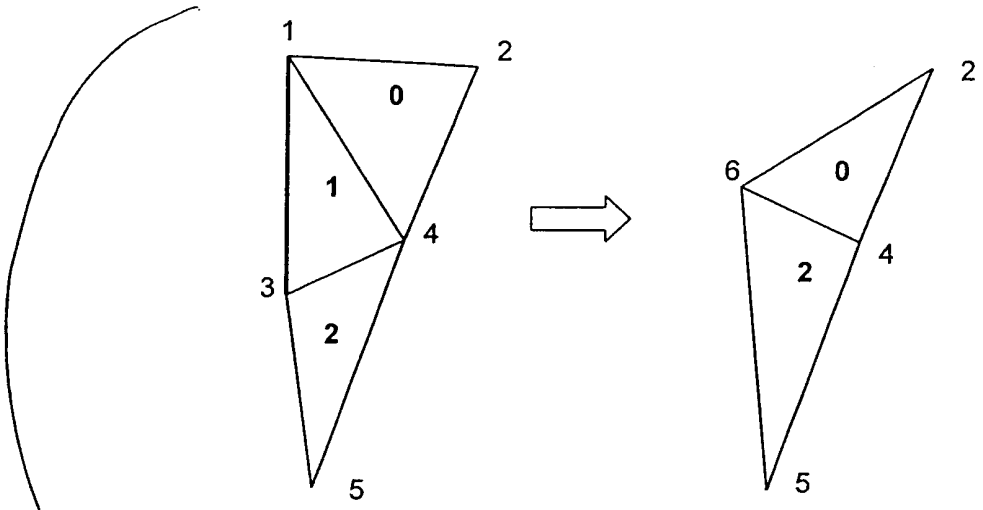

ORIGINAL MODE OF OBJECT, M
P1
P2
P3
P4
COPY TL
COMPUTE VNB
COMPUTE INC
COMPUTE MH
Preprocessing
WHILE MH NOT
S0
CREATE NEW
S1
S2
S3
UPDATE INC, S
UPDATE ATL
UPDATE VNB
S4
UPDATE TL
S5
UPDATE MH
S6
Skeletonization
G1
COMPUTE ATOMIC PARTS, Pi
Generating Atomic Parts
COMPUTE G, G'
T1
COMPUTE H
T2
Postprocessing

FIG. 4



v0 = 1, v1 = 3, vn = 6

Before:

INC(1,2) = {0}
INC(1,4) = {0, 1}
INC(1,3) = {1}
INC(2,4) = {0}
INC(3,4) = {1, 2}
INC(3,5) = {2}
INC(4,5) = {2}

VNB(1) = {2, 3, 4}
VNB(2) = {1, 4}
VNB(3) = {1, 4, 5}
VNB(4) = {1, 2, 3, 5}
VNB(5) = {3, 4}

TL(0) = {1, 4, 2}
TL(1) = {1, 3, 4}
TL(2) = {3, 5, 4}

After:

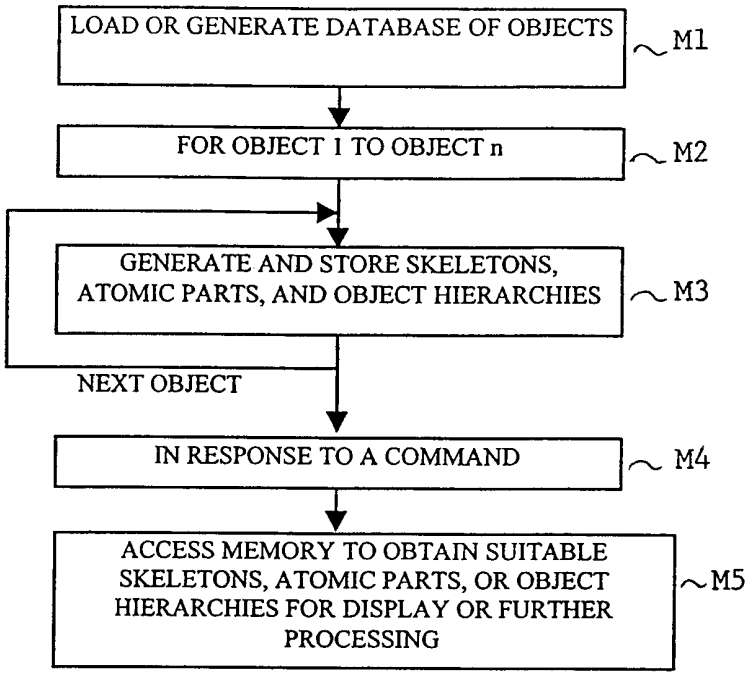
INC(2,4) = {0}
INC(2,6) = {0}
INC(4,6) = {0, 2}
INC(4,5) = {2}
INC(5,6) = {2}

VNB(2) = {4, 6}
VNB(4) = {2, 5, 6}
VNB(5) = {4, 6}
VNB(6) = {2, 4, 5}

TL(0) = {6, 4, 2}
TL(2) = {6, 5, 4}

ATL(4,6) = {1}

FIG 5


LOAD OR GENERATE DATABASE OF OBJECTS
M1
FOR OBJECT 1 TO OBJECT n
M2
GENERATE AND STORE SKELETONS, ATOMIC PARTS, AND OBJECT HIERARCHIES
M3
NEXT OBJECT
IN RESPONSE TO A COMMAND
M4
ACCESS MEMORY TO OBTAIN SUITABLE SKELETONS, ATOMIC PARTS, OR OBJECT HIERARCHIES FOR DISPLAY OR FURTHER PROCESSING
M5