Computer Graphics Research Laboratory
Quarterly Progress Report
No. 43*

Norman I. Badler
University of Pennsylvania
Department of Computer and Information Science
Philadelphia, PA 19104-6389

First Quarter, 1992

May 1992

* During this quarter the Computer Graphics Research Lab wishes to gratefully acknowledge support from MOCO Inc., NSF CISE Grant CDA88-22719, and ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Human Engineering Laboratory, Natick Laboratory, TACOM, and NASA Ames Research Center.

MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 92	Technical

4. TITLE AND SUBTITLE

Computer Graphics Research Laboratory Quarterly Progress Report

5. FUNDING NUMBERS

DAAL03-92-G-0221

6. AUTHOR(S)

Norman I. Badler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Univ. of Pennsylvania
Philadelphia, PA 19104-6389

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARO 30061.1-MA

11. SUPPLEMENTARY NOTES

The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report includes descriptions of various projects underway in the Computer Graphics Research Lab during January through March 1992. Included in this document are progress reports on new *Jack* features, the rule-based Spreadsheet Anthropometric Scaling System, contour bodies, locomotion, collision-avoidance reach, behavioral simulation, 3D motion analysis and reconstruction from monocular 2D views, task animation, radiosity rendering, ray tracing and filtering, and real-time sound interaction. Three Appendices include (A) an update on posture planning that will be presented in a poster session at the First AI Planning Conference, College Park, Maryland, in May 1992, (B) a PhD proposal on automated synthesis of simplified 3D models from detailed data, and (C) a survey paper on virtual building walkthrough systems.

14. SUBJECT TERMS

Computer Graphics, Spreadsheet Anthropometric Scaling System, Contour Bodies, Locomotion, Collision-Avoidance Reach, Behavioral Simulation, Motion Analysis, Task Animation

15. NUMBER OF PAGES

34

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

Computer Graphics Research Laboratory
Quarterly Progress Report
No. 43

Norman I. Badler
Department of Computer and Information Science
University of Pennsylvania
Philadlephia, PA 19104-6389
Fourth Quarter 1992

May 14, 1992

1 Introduction

This Quarterly Report includes descriptions of various projects underway in the Computer Graphics Research Lab during January through March 1992. Included in this document are progress reports on new *Jack* features, the rule-based Spreadsheet Anthropometric Scaling System, contour bodies, locomotion, collision-avoidance reach, behavioral simulation, 3D motion analysis and reconstruction from monocular 2D views, task animation, radiosity rendering, ray tracing and filtering, and real-time sound interaction. Three Appendices include (A) an update on posture planning that will be presented in a poster session at the First AI Planning Conference, College Park, Maryland, in May 1992, (B) a PhD proposal on automated synthesis of simplified 3D models from detailed data, and (C) a survey paper on virtual building walkthrough systems.

2 John Granieri

2.1 Recent Accomplishments

- The *Jack* 5.5 release was shipped in March.
- The Flock of Birds 6 DOF sensors from Ascension Technology were successfully integrated into *Jack* and were shown at NCGA in Anaheim. The interface library was written by Mike Hollick, and will be included in the next release of *Jack*. Several demos were also written that take advantage of the Birds to drive the human figure.
- I attended the 1992 Symposium on 3D Interactive Computer Graphics (sponsored by ACM SIGGRAPH) held in Cambridge. Cary Phillips and I presented the paper "Automatic View Control for 3D Direct Manipulation". This paper introduces a novel technique for automatically adjusting the view when manipulating objects in a 3D shaded environment. The technique is implemented in *Jack*, and will be available in the next release, via the command "unobstruct view" on the view menu. We use a hemi-cube projection and the hardware z-buffer to determine the visibility of the camera from the site of interest, then search the visibility map for an un-occluded position for the camera. The technique is a very useful adjunct to the direct-manipulation process.
- I also performed three days of training for Jacob Shiloah from Synergy Integration, a company which uses *Jack* in Israel to perform human factors analysis on a variety of design projects. They will also be distributing *Jack* in Israel.
- I attended the Silicon Graphics Developers Forum in Mountain View CA. The imminent release of the OpenGL will allow us to port *Jack* to other platforms which eventually support the OpenGL. There were also a variety of speed and visual feature enhancements which I learned about which will make their way into future releases of *Jack*. I also visited Ed Bellandi's group at FMC, and Barry Smith's group at NASA Ames. It was very enlightening to see what people are using *Jack* for.
- I've currently translated about 75% of *Jack* into fully ANSI-compliant C-code. This is necessary to make *Jack* compile and build correctly under Irix 4.0, and for the migration to C++.

2.2 *Jack* 5.5 Things

- **Anonymous FTP:** The anonymous ftp account (from **ftp.cis.upenn.edu** or 130.91.6.8), is where I place additional *Jack*-related programs and data files between formal *Jack* releases.

Currently, there are several useful files there. To log in, you would do something like the following:

```
% ftp 130.91.6.8
....user-id: anonymous
....password: your-login-name (e.g. I put granieri@graphics.cis.upenn.edu)
ftp> binary
ftp> cd pub/graphics
ftp> ls
ftp> get demojack.tar.Z
ftp> quit
%
```

If you are having problems connecting or do not have a direct Internet connection (i.e. some people can receive mail but can't do ftp transfers), send mail to jack@graphics.cis.upenn.edu, and I will mail you files uuencode-ed.

- **More Contour bodies:** In the anonymous ftp directory pub/graphics/bodies are 20 SASS-generated scaled human figures for use in *Jack* 5.5. We will use these until the new SASS is completed.
- **DXF Translator:** There is also an AutoCAD DXF-to-Psurf translator, which can convert simple DXF files into Psurf (pss) format. This is in the ftp directory pub/graphics.
- **Parsepea:** The *Jack* 5.5 distribution contained an incorrect version of parsepea (a program to translate environment files to ray tracer format). The correct version is in the ftp directory pub/graphics and can be downloaded any time.

2.3 Next 3-6 months

In the next quarter I plan to do the following:

- **Load carrying studies:** We are modifying *Jack* to meet the needs of users at US ARMY HEL to support the analysis of human figures carrying a variety of loads.
- ***Jack* on networks:** We will be spending a good part of the summer finishing the network versions of *Jack* for the Showcase exhibit at SIGGRAPH '92 (this was mentioned in the previous report).

- **Documentation:** I hope to have an updated User's Guide for the next release of *Jack*. Also, I will be creating a *Jack* Reference Manual which will formally list all *Jack* commands, both thier interactive format and their corresponding JCL formats. This will also document the network interfaces that will exist in *Jack*.
- **Geometry translators:** We will consolidate our geometry input programs for *Jack*, and be developing an IGES input/output geometry filter for *Jack*.

3 Francisco Azuola

3.1 Anthropometry Spreadsheet

Since the last report, two major achievements have been made. First, **SASS** it has been brought back to operation. It has been successfully used in the creation of a series of scaled human figures both for the polygon model as well as for the contour model, for 1st, 5th, 50th, 95th and 99th percentiles.

3.2 What to expect of SASS

As was originally planned, **SASS** has been redesigned and implemented keeping an object oriented philosophy in mind. In the implementation sense, this is not really visible to the user but it is of major importance for those who plan on upgrading **SASS** in the future (and all the code is now ANSI C compatible). In the design sense, what we mean with object oriented reflects in the fact that a hierarchy has been established to define the human figure components.

3.3 The Hierarchy of The Human Figure

The previous version of **SASS** handled segments by gathering them in a (simple) linked list structure. This was good enough to have any number of segments, but presented some inconveniences. In first place, the structure did not allow for defining relations among the segments in an easy way. Furthermore, the segments were defined as a triple of values, namely, an (x, y, z) tuple. These tuples were placed in the list in a fixed position, corresponding to the predefined location assigned to a particular segment. For example, (x1, y1, z1) corresponds to the head segment, (x2, y2, z2) corresponds to the neck segment, and so on.

In the new version, the structure used is a hierarchical one (tree). At the bottom of the tree, the leaves correspond to the segments. The internal nodes correspond to, what we call, the body parts or body objects (which should not be confused with the "objects" in the implementation). The root of the tree is reserved for storing the figure's information. The body parts are composed of body segments and the figure itself is composed of body parts. One can think of the figure as corresponding to the complete tree. There is another category, the joints, which has been appended to the root of the tree. The joints are the linkages among the segments and also among the body objects. A figure can be defined as a collection of body parts, joined together by joints. Each body part, in turn, can be defined as a collection of body segments put together by joints. Each segment has been specified with an access code, a segment type, and a list of sites. Joints are defined similarly. Body objects are specified by a type, an access code, and a list of components, namely a list of segments, joints, and sites related to that object. It is important to make some remarks here. In addition to this being a more powerful structure, it is also much more flexible because it is user definable. Indeed, the user can modify the specification of the objects, segments and joints and even the figure itself by just writing down a figure description file. In this way, it is possible to create different types of relations among the pieces.

3.4 The Rule System

SASS has also changed in a deeper sense. **SASS** not only works on relations but also is rule based. As an example, **SASS** currently defines a rule for computing the height of an individual as the sum of the segments' lengths in a path that goes from head to feet. For those segments in the path, the rule allows to varying their lengths if the stature changes and; vice versa, to change the stature if the length of any of the segments in the path changes. There is an alternate rule that keeps the stature fixed and adjusts the segments' lengths accordingly, if the length of one of them varies.

Another rule includes changing the mass according to the stature and, conversely, changing the stature according to a specific mass value; and rules for checking proper bounds in segment (object/figure) dimensions.

The rule system of **SASS** is a simple one, though. At the present moment, we do not have a definite set of rules. Even the rules that have been implemented are waiting to be refined in the future. Depending on the complexity of the rules, we will be required to upgrade the rule engine. This is not a trivial task since rules might change from one situation to the other. It is conceivable that the rules themselves could be specified by the user as opposed to what the current implementation provides, i.e., a fixed (hardcoded) set of rules.

3.5 Rules in SASS

The stature of the human figure is computed using two different rules. In one case, the stature is kept variable. If the stature varies, the segment lengths in the stature path vary accordingly. Similarly, if the length of any segments in that path varies, then the stature changes. The underlying criterion for doing these changes is a linear one. The segments in the stature path have been defined as: (head, neck, upper torso, center torso, lower torso, upper leg, lower leg, feet). The length of each of these segments, except for the feet, is computed as the girth value in the z coordinate. For the feet, the length is computed as the girth value in the y coordinate (since for the feet, the girth in the z coordinate is the longitudinal dimension). It should be noticed that the thickness and width of the segments are not affected by these changes, for there is no rule to decide the effects of stature changes in these parameters.

The updating process must be done carefully, for it might happen that modifying the length of a given segment violates the range of possible stature values, or conversely, if the stature is changed, this change might not be satisfiable by variations in the segment lengths.

The other case considers fixed stature. The idea is to adjust the segments' lengths along the stature path if the length of one of them varies, such that the global length (stature) remains constant. While this might appear easy to do at first, it is not a trivial matter. To understand why, we must study how the segments dimensions are obtained. Each segment's dimensions can be seen as a triple (x, y, z) of values. This triple of values, is obtained by interpolation from actual data provided by the user. This "real world" data corresponds, in fact, to the value of the girth in each of these coordinates for a set of different percentiles (e.g., 5th, 50th, 95th percentiles). SASS provides a given triple (x, y, z) for percentiles in the range 5-95 by means of interpolation (also, if the user specifies a triple, SASS provides a percentile value corresponding to that triple). Thus, a segment's dimensions are constrained by the "real world" value range.

Furthermore, the stature itself is restricted by a "real" set of values (for each of the percentiles). When the user specifies a particular change in the triple (z coordinate) of a given segment, the underlying rule attempts to satisfy the constraint of fixed stature, that is, it tries to keep the stature value constant. For example, assume the length of the head has decreased. To keep the stature fixed, the lengths of the other segments in the stature path must vary in the opposite way. Currently, the modification is done in a linear way since there are no rules to define this otherwise. But it might be the case that in the updating process one of the segment's dimensions (namely length) cannot be satisfied. In other words, the resulting dimension is out of the range established by the 5-95th percentile values. In this situation, the rule sets up the length to its closest limit value (5th percentile value or 95th percentile value), and tries to satisfy the requirement of fixed stature by modifying the remaining segments in the path. Notice that there is a possibility that the stature can not be kept constant. There is one more step involved in the updating process that

will be discussed later. In this mode (fixed stature), if the stature is globally varied by the user the segments change correspondingly (if possible).

3.6 Object Level

As discussed before, a figure is built up as a hierarchy: the segments in the lowest level, the body parts (objects) in the next level, and the figure itself as the root level. The body objects are defined (by the user) as sets of segments and joints. For instance, the object 'leg' can be defined as a set containing two segments ('upper leg' and 'lower leg'), two sites, and six associated joints.

For the matter of the following discussion, it is not relevant what the sites or the joints are but one can think of a simplified object involving only a set of segments. The object level is an abstraction of the idea of body parts. So we associate to each object a body part. It is important to keep in mind that the "real world" measurements are done on a segment basis. The objects (body parts) are defined to provide additional flexibility to the user. As the internal structure of each body part can be specified by the user, one can consider having as many parts as necessary (or as segments there are). By default, **SASS** has defined eight body parts, namely, head, torso, left arm, right arm, left leg, right leg, left foot, right foot. These objects encompass most of the (user defined) segments. Having objects allows the user to perform global modifications on a per body part basis, as opposed to doing localized changes on specific segments. Although it is possible to go and change values for a particular segment, it is *generally* desirable to be able to do modifications on a body part, as a whole. The idea of having body parts presents some difficulties though. When body parts are introduced, the rule system must consider performing the appropriate (coherent) updates on two different levels simultaneously. If the user changes values on the segment level, these changes are reflected also at the object level, and conversely, when changes are done in the object level, these changes affect the segment level values. Also, recall that changing the segment level values was governed by a set of rules. There is an equivalent (compatible) set of rules for the object level. For instance, changing stature is governed by rules in the object level (and in the segment level).

3.7 What Is An Object?

Objects are implemented as artificial structures. The "real world" data doesn't provide information for any body parts, only for body segments. In a sense, objects can be considered as clusters of segments and each time an object is accessed the access is redirected to the corresponding segments. Conversely, if a segment is accessed, all the objects containing that particular segment are accessed. There is more under the definition of a body part. Actually, the object's dimensions

are approximated by considering the bounding box around the segments of which it is composed. In this way, a body part comes to life as a tuple of (x, y, z) values. Why bother doing this? At first it might seem unnecessary since the components of an object, i.e., the segments, have some associated (x, y, z) tuples already. However, there two good reasons that justify our approach.

In first place, using a bounding box strategy, we can bound the dimensions of the segments (components) of a given object. Also, it allows us to have two sets of dimensions: the expected dimensions and the actual dimensions. The expected dimensions are those determined by the bounding box approach. The actual dimensions are the dimensions of the body part when we think of it as a cluster of segments. Thus, the actual dimensions reflects accurately (up to the accuracy of the segments' dimensions) the dimensions of the object.

Having these two dimension sets provides a way of constraining the growth of the body parts. The following rules apply. If a segment (member) in an object grows (or shrinks) then it should not grow beyond the limits of the object's expected dimensions for a given percentile, if we want to restrict an object's dimensions to be of a certain percentile. So we can, for example, try to adjust the dimensions of the other segments in the object's segment set so that we keep the object's percentile fixed.

It is important to understand the back and forward process that goes on between objects and segments. We can have the global dimensions of the body, for instance, those of a 50% (standardized) human being, but we know that the body parts need not to be 50% all of them. In fact, we do not have a rule yet to specify the percentile of the body parts (segment-wise) for a given global body percentile. So keeping that in mind, we must be able to change dimensions of the body objects (segments) to comply with all the possible compositions of a 50% body. We must be careful when specifying other rules for, say, stature. We need to make sure that a given change of stature does not break any other rules, that is, we must assure that the resulting body composition (i.e., the percentiles of the body parts (objects/segments)) are those valid for a 50% body. Also, we require to comply with the restrictions on the segments' (in this case on the stature path) dimensions, i.e. we cannot scale a segment beyond the limits established for that segment by the population data. Also, we need to assure that the stature modification rules are respected (i.e, those rules we mentioned before in which the segments' lengths are modified following a specific layout; currently modifications are done linearly). It seems that a possible solution, in this particular case, is along the following lines.

If the stature is modified, then a new global percentile is computed. For that new global percentile, we have a specific rule telling us what the possible compositions are. Thus, we use these compositions as our rules for doing the segment length modification, (instead of doing it linearly as it is done in the current version). Thus, there is no conflict. But that is only if there is a coherent definition of the possible compositions and the stature-path segments' length, i.e., the compositions must agree with the segments' length under the population data being used. In other words, the

compositions are not unique, they are dependent on the population data used.

To illustrate this, suppose we have the following (partial) composition set: feet 30%, legs 45%, torso 60%, head 40%,... for a 50% body. Then suppose we want to change the stature in such a way that the resulting body percentile is 60%, and the analogous (partial) composition set is (feet 40%, legs 56%, torso 50%, head 40%,...). Then we scale the objects in the stature path (which are those listed in the composition sets) to comply with this second composition set. But, we must be sure that there is no conflict in doing so, that is, for instance, the feet might be able only to grow from a 30% to a 40% under the population data being used. Thus, there is an inherent need for the compositions to be determined under a given population, i.e., different populations will have different compositions. Solving that problem, we must make sure that the scaling (of the segments) resulting out of this complies with the object's (body part) constraint, i.e., the bounding box limitations. This should be the case if we have composition sets that agree at both the segment level and the object level.

In the previous example, for instance the compositions were stated at the object level. There must be an equivalent composition at the segment level. Following this example, the segment version of the composition for the 50% figure is, for instance, (... upper leg 45%, lower leg 60%, upper torso 76%, center torso 57%, lower torso 45%, ...), assuming legs decompose in two pieces and torso in three pieces. But what if the compositions, even though being based on a particular population data, are not available for all the possible percentiles? (With good luck we hope to have one for a few of the percentiles.) We would have to interpolate compositions (if it is sound to do that) and make sure a given segment's length is not violated (according to its percentile range) when trying to go from a composition for the 50% figure to that of the 60% figure. If we had only one such composition to work with to account for all possible compositions on the percentiles range, then it would be necessary to make sure that this composition is not violating the range of values a given segment's length can have for the population under consideration.

This is basically what happens in the present version of SASS. Since we do not have a composition analysis available, we have assumed decompositions are unique for a given population (i.e., one composition for all the percentiles (not one for each)) and furthermore, this composition is linear, i.e., for a 50% figure (feet 50%, legs 50%, torso 50%,...) and similarly for the segment composition (... upper leg 50%, lower leg 50%, ...). This has sensibly increased the difficulty of the problem because such an assumption is far from being applicable to real world situations.

This has wound up in the need for additional rules in SASS to verify that there is an agreement among all parts. Recall, for instance, the stature problem. In that case, we are considering compositions to be linear. So we need to be especially careful not to end up with a segment's length violation. To avoid that we limit the growth of a segment to its 1% and its 99% (i.e., below and above limits). If we do not achieve the desired global growth, i.e, the local segment's growth was not sufficient, then we go and adjust the other segments in the stature path. This is done in

an iterative way. Also, observe that we have to keep track of two levels of abstraction, that is, the segments and the body parts. It is necessary to double check, once for the segments' lengths not to violate their limits and once for the objects' lengths not to violate their limits. This is necessary because the objects' composition has been assumed to be linear. A similar situation arises when considering the other way around, that is, modifying a segment's length implies a careful set of steps along the hierarchy to keep track of this modification's effects on the objects' lengths and then the effects of these on the global length (i.e. stature) so that the resulting stature has a value between its percentile limits. (If the stature is kept fixed then we do not go all the way up in the tree but we need to perform a readjustment of all the other segments (objects) to assure the stature is kept constant, whenever possible.) Thus linearity is not the right solution (and it is even a difficult to implement one), but currently it is the only solution.

From here we should conclude that there is a need for a sensible decomposition of the body at both the object and segment level (these two are not necessarily the same as we noted before) in order to be able to handle global and local growth. Incorporating these compositions will require of a more powerful rule system.

3.8 Figure Creation

SASS still cannot create a figure file due to the lack of a mapping between the population torso data and the model of the torso we use in *Jack*'s environment (i.e.,the 17 segment torso). A function to do so (at least approximately) is being developed.

However, to provide relief for the pain of SASS users, we have provided a function to produce a file containing the scaling of a figure, and then we rely on *Jack* to create the figure file. This might not make completely happy all of the SASS users, but we will explain in the following why we have decided to keep this as an option, even when we have available the figure file creation function.

The reason we consider using scaling files rather than figure files is simple. Consider the situation in which the user in *Jack* wants to determine the percentile (dimension) ranges of the human figure to comply with a given task, that is, the problem of finding the specific figure (%) that can fit in a particular working environment. One can attempt to read each of the possible figure files out of *Jack* libraries and try to keep the figure in the position we want. The other (more sensible) option is not to load different figure files, but instead, to load different scaling files. Then the (same) figure can be scaled using all these different files to find the one that best suits the given environment. This is not only faster but it is even more appealing to the user.

When a given scaling has been found best, if the user needs to do further adjustments, (for example if longer arms or longer legs required) a new scaling file can be created in *Jack* with the

required dimensions. There is still the problem of how to make sure the new scaling is in agreement with the population data. One possibility is to let SASS decide this.

3.9 Figure Scaling

As we stated before, SASS output is a figure scaling file. The scales in this file are obtained directly from the (x, y, z) tuples of the segments or objects. Thus, this scaling file represents the dimensions specified in the (population's) girth file, for a given figure percentile. On the other hand, *Jack* provides a front end in which a figure is displayed and scaled according to the information in the scaling file. Currently, two human figure models are used in *Jack*, the polybody and the contour body. The polybody is a (crude) approximation of a human body, containing a set of segments and joints. These segments do not correspond exactly to those segments defined in SASS. For instance, the polybody's torso is composed of 17 segments. SASS has three segments to account for the torso. The reason for this difference is that SASS' segments correspond to actual measurements of the human torso, while *Jack*'s segments were defined with the idea of simulating the human spine behavior. Thus, there is a mismatch between both definitions. More problems show up when we consider that a segment in SASS (and in *Jack*) is defined by a single tuple (x, y, z), that is, a uniform width and thickness is assumed in both cases. The final result is a human body model that has a (not so real) human-like appearance.

The major problem with the scaling is due to these mismatches. For instance, the upper leg, once scaled appears to be too thick and too wide, in comparison to the lower leg. The same problem occurs with the upper arm and the lower arm. Also, the scaling of the pelvis of the polybody presents problems; it seems to be too wide and thick. As another example, the torso appears to be narrow and short.

There are various solutions to these problems. The simplest one is to adapt the data to the model. In other words, modify the scaling factors in order to obtain a good (looking) figure scaling. There are no rules to do this though. The rule we use is to consider body lines as being (second order) continuous. There are no abrupt changes from one body part to the next one (assuming no deformations, like a hunched back, are present). Thus we approximate (in an arbitrary way) the scaling factors in order to achieve this continuity. The largest discrepancies are the ones mentioned above. Other minor ones are the scaled neck being too wide, hands being too narrow. In general, the scaling factors are not changed by more than 10%. Being the polybody is a linear model of the human figure (linear segments), this is possibly the only feasible solution. Attempting to change the actual model to adjust it to a particular data set does not seems like a very good idea, because it would be necessary to adjust the model for each such data set. In fact, there are even other difficulties to consider like for instance having the clavicles in the polybody as real external segments (no matching data available in SASS). The clavicles are part of the shoulder complex,

and they are used like an artificial supporting structure. On the other end, in the contour body model the clavicles are only present for completeness of the model.

We might then say, in conclusion, that the polybody model is a simple model of the human figure, and as any other model has advantages (e.g., simple, fairly accurate in behavior) and disadvantages (e.g., linear segments, gaps between segments). Thus, we should not expect a perfect match between this model and a real life human body data.

A third solution is to improve the model. That is exactly where the contour model of the body becomes handy. In fact, the scaling factors generated by SASS are mapped into the contour body with almost no modifications necessary (for the contour figure case, adjustments are done to the torso, legs, arms, and hands). These adjustments do not go over 5% of the actual values. Again, one must keep in mind that even though the contour model is a more accurate representation of the human body, it is not a perfect one. Moreover, we must recall that the SASS scaling factors file is created based on a generic (average) population and the figure resulting from that scaling might not completely match a real human being (for suppose that the population's average torso length is greater than the torso length of a given individual and the population's average leg length is smaller than the one of the same individual, then we end up with a not so real scaling for the contour model). Thus, even though we have assumed some adjustments are required, it is still necessary to prove if this is the right way to proceed. So far, the criterion that prevails is to display a good-looking (well proportioned) human figure.

3.10 Short Term Goals

The short term goal is to provide a figure definition function to be used as an alternative to the scaling definition function. Currently, this function is being designed (partially due to the lack of a real mapping between the "real" world torso measurements and the model torso segments).

3.11 Longer Term Goals

- (Shoulder Complex) We need to modify the definition of the shoulder complex. SASS can retain the same definition of (x, y, z) tuples for the upper and lower limit of the joints. However, the shoulder complex in *Jack* is defined with four degrees of freedom. The functions shoulder driver and clavicle driver in *Jack* do the conversion from the (x, y, z) format into a four degrees of freedom format.
- (Strength Sheet) An upgrade of this sheet is required to introduce new rules and/or improve the existing ones.

- (Database) As for the database, we think it is necessary to have our own database rather than using a Prolog shell. That is, having a database implemented in C can be faster and also we avoid the need of a Prolog shell.
- (Dynamics based model) It is necessary to introduce new center of mass and moments of inertia information for every segment in **SASS**. The figure file should be modified (and *Jack* should be able to recognize this changes) to incorporate this information.
- (Rule system) As new rules become available, the rule system must be upgraded accordingly.
- (Interaction *Jack*-**SASS**) A two way communication between **SASS** and *Jack* is necessary. The problem to solve is that of deciding whether changes in dimension done to the figure model, while working in *Jack* environment, are valid, under the population data used to (initially) create the figure.
- (Figure fitting) We need to create a function in *Jack* that performs a figure fitting. Given a set of 99 figure scalings (one for each percentile) find the one that fits best on a given environment.
- (Interface) There is a need for a new user interface, perhaps under X windows manager, to provide the user with more flexibility. We have not done much in this respect, even though we are aware of the problem. The current interface was designed in a very rigid way and it is difficult to change things around since everything is hardcoded.

4 Pei-Hwa Ho

4.1 Contour Body

The switching between contour body and stick figure used to just switch the segment psurfs with the default scale factors in the figure definition file. If the figure to be swapped has been scaled by any means (e.g. through **SASS** the switching will not carry over the new scale factors. A modified switching command will now do the switching correctly and thus will be compatible with future **SASS** output file.

5 Hyeongseok Ko

5.1 Locomotion

The most prominent problems in utilizing rotoscopy data for human walking animation are: Generalization and Constraint Satisfaction. We devised an algorithm to generalize a given rotoscopy data (*prototype*) to other walks under different body conditions and different step lengths (generalization). We do not assume any predetermined anthropometric ratio among the body segments. The kinematics of Cartesian points (markers) are considered instead of the joint angle data, which is considered to be essential to overcome the cumulative error along the links (constraint satisfaction).

For the generalization, we have a transformation algorithm that derives the walk of (S_2, sl_2) from the walk of (S_1, sl_1), where S_i and sl_i represent subject (body condition) and step length, respectively.

One of the desirable properties of our transformation is that it is *transitive*: Let ξ_{12} be the transformation from $w(S_1, sl_1)$ to $w(S_2, sl_2)$. Let ξ_{23} be the transformation from $w(S_2, sl_2)$ to $w(S_3, sl_3)$. Let $\bar{w}((S_3, sl_3)$ be the resulting walk profile by the real computation of the composite transformation $\xi_{23} \circ \xi_{12}$ applied to $w(S_1, sl_1)$. Note that w is just a tuple of a subject and a step length, whereas $\bar{w}$ is the profile that contains all the information to generate the walking animation. Let ξ_{13} be the transformation from $w(S_1, sl_1)$ to $w(S_3, sl_3)$. Let $\tilde{w}(S_1, sl_1)$ be the actual out come of it. Then

$$\bar{w}((S_3, sl_3) = \tilde{w}(S_1, sl_1) \tag{1}$$

holds.

The intuitive meaning of the above theorem is that the transformation preserves the original characteristics of the prototype. In other words, if a prototype is given, independently of the body condition and step length of the goal walk, our transformation algorithm tries to resemble the original characterisics. Therefore we can generate multiple styles of walking by having more than one prototype in our data base.

The above algorithm is in implemenation phase. After it is complete, we will extend it to handle the local stepping, and then uneven terrain locomotion will be studied.

6 Wallace Ching

6.1 Work done during the last quarter

I have kept making enhancements to the path planner system. Some of these are theoretical enhancements to the path planner process. Others are system enhancements that make the software more user friendly and usable by others.

Major enhancements include:

- The translation of the figure is now unified and handled in the same way as the joint movements.
- Starting and final configurations that collide with the environment are allowed. The system will find the nearest collision free configuration as the system start and goal node.
- The strength model interface is being made independent of the strength data source. It should be easy to modify it once the strength data is available.

6.2 Current and Future Work

The current work involves updating the system to handle the current human body with multi-segmented torso. The coupling of the shoulder joints need special attention. The multi-segmented body will be approximated with a bounding box in the collision detection phase. A planar path planner is being constructed from existing components that can handle the translation of the figure. This means that the system can now plan a collision free path for the whole figure within a cluttered environment.

Other work to be completed includes fine tuning of the searching process and better use of the strength data so that the resulting motion can be more natural.

Finally, the software is being made more user friendly so that it can be embedded into the next *Jack* release.

7 Tripp Becket

7.1 Walking and Behaviors

We can apply multiple avoid or attract behaviors to a human figure that:

- are sensitive to only one object, all objects of a cerain type (like all cylinders), or only the nearest or furthest k objects of a certain type.
- have distance thresholds.

Human figures, after figuring out which way they want to go, try to reduce the goal by walking. I have a heuristic algorithm for constructing the next step position that:

- won't turn more than 45 degrees in a single step (by turn I mean reduce the current and desired heading – it doesn't stop and turn, it always keeps moving...)
- makes step length a function of turning angle (if turning angle is 0 take maximum step, if 45 take minimum step).
- attempts to keep the feet 17cm apart laterally
- while turning, makes outside foot take shorter steps (inside foot reduces most of the turning...)
- decides on the next step only after the previous step is finished

8 Jianmin Zhao

8.1 Work in Progress

Having had my dissertation proposal passed, I am now implementing it. I have converted the ten single arm reach data from MOCO to *Jack* motion format. This data is very simple and easy to reconstruct by our primary system. After I finish coding to deal with critical configuration, I shall try two-arm general motion. In case I cannot get real-data, I have to design simulated motion through *Jack*.

9 Libby Levison

9.1 Previous Work

My research began when I used *Jack*, Yaps and KB to animate a set of instructions describing a simple aircraft maintenance task. This project has led me to further work on building a language which will allow a user to specify high-level description of tasks and actions into *Jack*.

Recently I re-animated the original "FCV removal" scene making use of Cary Phillips' animation behaviors. Although this generated a much more natural animation, each individual motion had to be scripted by hand. This means that each action must be decomposed into an explicit sequence of motions. Every time that an action is used, the same decompostion must be redone and restated. There is no current optimization nor method for defining and reusing sequences (possibly ones parameterized for various contexts). This strikes me as a logical continuation of Phillips' work: some of my current work begins to address this problem.

9.2 Current Work

My current research is in understanding instructions for the purpose of generating animations. As a member of the **Anim**ation and **N**atural **L**anguage project (AnimNL) I work between the Language, Information and Computation (LINC) Lab and the Graphics Lab, building animation definitions of those instructions which result in physical actions.

If the purpose of an interaction is cooperation on a task, the computer must understand the user's instructions and act appropriately. This is relevant at the level of the human-computer interface as well as in the domain of the AnimNL project, in which we want to instruct a graphics program to generate certain animations. In building a system which will interpret the user's instruction, I am specifically interested in verb-object relations; I have identified wide variations in intended action which occur when a verb appears with different objects. For example, the verb *open* is associated with two distinct physical actions in the instructions *open the door* and *open the soda can*. If we believe that each verb has a unique meaning then we must account for these variations in interpretation at the sentence or the instruction level. I would argue that each verb has a partial, core meaning; this meaning is completed in an utterance with information carried by the verb's object, as well as by understanding the *intention* of the given instruction. For example,the definition of *open* might be something like: *provide access to*. One fact that I know about the door to my apartment, either from living in my house or through visual perception, is the door's degrees of freedom. (I might also know that this is a heavy door and that it is hung to swing shut if not

propped open.) If part of the definition of *open* includes moving its object, then interpreting how to *open my door* entails checking how my door moves – what its degrees of freedom are – either translation or rotation. If the door is marked as allowing translations then I probably have a sliding-glass door; not only do rotations indicate a hinged door, but positive rotation implies pulling, while negative requires pushing. These observations suggest that building animation definitions based solely on the verb or exclusively on the object won't work. I advocate instead a hybrid system in which the core meaning of the verb makes use of (geometrical) information associated with the object.

Investigating these definitions requires an application that allows the user to give the system task instructions, as well as providing the user an easy way to check the computer's interpretation of those instructions – in other words, to verify that the correct action is performed. The animation of repair instructions satisfies this requirement: to generate an animation the computer must understand the instructions, and the resulting animation provides an easy way for an engineer to (visually) check the correctness of both the original instructions and the interpretation – the resulting simulation. (In addition, this application has real-world utility: rather than read an instruction manual, a technician or trainee can watch an animation of a simulated agent performing a repair or maintenance task.)

My research uses *Jack* to provide 3D-modeling capabilities as well as extensive human factors and anthropometric analysis tools. At the same time that I am examining linguistic issues in the instructional texts, I am investigating methodologies which will enable an engineer to produce simulations of task-level actions despite possibly limited knowledge of low-level animation techniques. I am using a minimal set of *action directives*:, animation instructions like `move left foot` or `bend torso`) to define higher-level actions such as **grasp**, **attach** or **open**. I call these composites **task-actions**. I hope to provide a richer set of task-action definitions as well as a utility for defining new task-actions. These action descriptions, from the viewpoint of animation, will allow an engineer with minimal knowledge of graphics to generate animations. The interpretation process will save the engineer from defining multiple animation procedures such as *open-door*, *open-book* and *open-jewelry-box*. I am trying for an economy of action definitions, relieving the engineer of the burden of specifying detail which the system might well be able to deduce.

In summary, then, I believe that I can classify both the verbs and their objects in instructional texts according to their lexical semantics: the verbs based on the underlying physical action, the objects dependent on geometrical information. I am building a high-level utility, within the *Jack* framework, which will determine, in a given instantiation, exactly how to apply the verb to its object by reasoning about such things as the geometry of the object. I will use *Jack* animation directives – primitives which describe high-level motor control – to build compositional definitions of the physical actions underlying the instructional verbs. These *task-actions* will describe the tasks to be performed at a high-level and not on a movement-by-movement basis.

I am currently investigating three areas:

1. Issues in Lexical semantics:
   Identify core meanings of verbs and build functional descriptions of nouns;

2. Developing a language to describe actions:
   Build a minimal primitive set to use in defining actions, as well as defining the syntax of the actions description language. This entails developing a way to incrementally chain action primitives together. For example, *push* might be the chain:
   `contact, constrain-to, translate, translate, translate...`,
   where the translate is repeated until the goal is acheived.

   One issue I am investigating is the usefulness of an underlying set of behaviors in terms of building the task-action definitions. For example, in instructing someone to *reach the cup* we would never specify *don't try to put your hand through the table.* While *Jack* provides collision detection: making use of it will make the resulting animation appear more realistic. However I am considering variations in the definition of *reach-action* when there are explicit background behaviors. For instance, if there was always a behavior `don't put hand through solid object`, it would never need to be explicitly stated, nor checked, in a task-action definition. *Jack* manages a large amount of the human motor control issues, and Phillips' behaviors allow us to step away from those details; a system of "world behaviors", describing how we (unconsciously) interact with the physical world, might make defining realistic set of task-actions a feasible task.

3. Implementing an action interpretation algorithm:
   Develop an algorithm which combines the information required by the action definitions with the knowledge stored in the object definition. For now, I simply intend to get the data sharing implemented. I am using C++ in order to take advantage of classes and inherited knowledge; subclasses will automatically inherit information from super-classes.

9.3 Future Work

- Provide a way of specifying a chain of associated behaviors in *Jack.*
  This is a first step in building composite actions. Actions should be linked to provide basic manipulation ability in the interface, for instance, the user ought to be able to visually verify that a *reach* and a *grasp* are linked together to form a *get*; if the user wants to change the start time of the chain it ought to be possible to modify the entire chain at once.

- Extend the JCL front end for the animation behaviors.
  Make it possible to read in animation behaviors that make use of symbolic names as the site of different movements. This is needed for testing and working the AnimNL code.

- Build an object knowledge base.
  For various objects in the instruction sets, define them both geometrically and functionally. Objects that the AnimNL group is concerned with are doors, thermoses, boxes, and fortune cookies.

10 Min-Zhi Shao

10.1 Radiosity

1. I have implemented the progressive radiosity method with adaptive meshing (patch to element) and analytical calculated form-factors (Baum el al, 1989). This should be the state of art progressive refinement radiosity method to date.
2. I am working on a two pass (global and local) radiosity shooting method based on the above. Since in our method, the shooting patch is either very strong (light source in global shooting) or very near to the elements which gathering the light energy (local shooting), the more accurate analytical form-factors and the adaptive elements subdivision techniques are critical in our implementation.

10.2 Future Goals

My next goal is try to extend radiosity to general environment with non-diffuse and also non-specular surfaces. The following are some of our major considerations so far:

1. Progressive form-factor refinement method as theoretical background (based on my 1988 paper)
2. Dense and even meshing for specular-like patches. But instead of keeping hemi-cube for every specular-like patch in the storage, we are going to store a list of patch numbers we found in the hemi-cube. We can also store the union of patch numbers of, say m by m neighboring patches. We expect the storage problem can be largely reduced with the trade-off of rebuilding hemi-cubes in each form-factor refinement iteration for specular-like patches. But z-buffer depth comparison can be largely reduced with the hemi-cube patch list in the storage. Therefore, the more complex of the environment, the more efficient (relatively) our method would be. And unlike the two-way (eye and light) ray tracing, the refinement procedure is much less dependent on the geometrical complexity of the environment. Furthermore, the solution is view-independent.

3. Adaptive meshing for diffuse-like patches based on the light energy distribution of light sources and specular-like patches.

4. Real-time viewing the environment. Ray tracing post processing technique is neither real-time nor efficient and necessary for non-purely-mirror patches (which we are not going to simulate, our environment is somewhat in between ideal diffuse [including] and ideal specular [not including] environments). But if the meshing of specular surfaces is as fine as the pixel size level, the pure mirror surface could then be included. We are going to use Gouraud interpolation for display directional radiosity. But storing all directional radiosities (with the resolution of hemi-cube) is neither necessary nor practical with the limitation of the machine memory. Therefore, we are considering to use surface fitting techniques, such as piecewise bicubic Bezier patch. This is a adaptive fitting method. So, we can largely reduce the final radiosity output storage by storing the necessary control points instead of radiosity in each direction.

Finally, a preprocessor for the radiosity input environment (*Jack* peabody file) is nice to be added. This seems to be well suit for a term project.

11 Jeffry S. Nimeroff

11.1 Work Done in the Past Quarter

During the first month of 1992, I completed a ray tracing implementation designed to allow me to easily test texture map antialiasing schemes. The first method that was completed and tested was a spatially invariant random weighted average method I designed based on solutions to the N-queens problem. A solution to the N-queens problem consists of the placement of N queens on an NxN chessboard so that none of the queens is threatened. The individual solutions make good discrete convolution masks for two dimensions and are treated as such when reconstructing a single texture value from an NxN grid of texture samples. Although these results produced images of slightly higher quality than the simpler area averaging schemes (box, Bartlett, etc.), the method was abandoned since the quality increase did not meet expectations.

Over the last two months, my research has consisted of reformulating the texture mapping process to be used in a prototypical version of the new *Jack* ray tracer. It was decided that a stochastic ray tracer would do a reasonable job of antialiasing the symbolic image function as long as no aliasing error was introduced by the texture mapping process. The texture mapping problem was sufficiently reduced to a reconstruction/low-pass filtering problem (relying on the ray tracer itself to deal with screen space antialiasing) Fitting a bicubic b-spline surface to the texture image

reconstructs a C^2 continuous version of the texture from its lattice samples. The C^2 continuity constraint also bandlimits the reconstructed texture image providing some feasible Nyquist limit for the ray tracer to perform its point sampling.

11.2 Current Work

Besides continuing my filtering research with the hopes of analyzing parameterized cubic interpolation schemes, my current work includes leading a group of graduate students in the design and implementation of a C++ prototype of the new *Jack* ray tracer. A C++ implementation allows for greater flexibility by providing a level of data encapsulation not found in standard C. This implementation should provide a platform which will allow others to continually update and enrich.

The texture antialiasing research that I am performing currently is going to be grafted into the new ray tracer during the coming quarter.

12 Ranjit Bhatnagar

12.1 Real-time Theremin

Our goal was a real-time animation of the *Jack* figure playing a Theremin, combined with real-time synthesis of appropriate sounds, under control of ascension technology 'Flock of Birds' space tracker.

The visual complexity of the *Jack* figure, full shaded animation is not possible at 'real-time' (approximately 12 or more frames per second) rates. Using wireframe animation and simplifying the *Jack* figure allows a rate of two to five frames per second, depending on the hardware. Further improvement may be possible.

I experimented with various ways of controlling the *Jack* figure, and settled on the use of *Jack's external figure control port facility*. I had to modify the port code slightly. The theremin control program sends joint-angle update commands to *Jack* through a socket. In the future it may, instead of updating the *Jack* figure's arms, it control the position of invisible objects to which the figure's hands will be attached, thus using *Jack's* reaching algorithms for more natural animation. This will further decrease the frame rate and increase the latency of the animation, however.

The Indigo has appropriate hardware for sound synthesis. Earlier plans to use an external MIDI synthesizer are on hold, but that might be a useful technique in the future. The theremin sound is generated as packets of samples, based on the user's input. Each packet is about one thirtieth of a second long, so the latency of sound output is just over 1/30th of a second, quite sufficient for real-time operation.

I have made a version of the theremin which plays recorded sounds rather than sine waves, and another which can have its pitch output quantized to any desired scale.

12.2 Future Goals

We are hoping to simplify the 3D model sufficiently that the animation can run at reasonable rates, even on the Indigo, which has very slow graphics. One current method of achieving higher frame rates is to run *Jack* on one of the very fast machines, and the theremin control and audio programs on the Indigo, communicating over the ethernet. (Because of the use of UNIX sockets for *Jack's* control ports, this is very simple.) The next project will be a virtual drumset.

Since the animation will always have a much higher latency than the sound, it may be possible to improve the interactivity of the animation by predicting the user's control movements up to half a second in advance. Even if the prediction algorithm is not very accurate, it may still improve the effect. Also, using a custom-written graphics program rather than the very flexible and slow *Jack* software could improve graphics speed significantly.

A Posture Planning for Agent Animation

Posture Planning For Agent Animation

Moon R. Jung
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia PA 19104-6389

Norman I. Badler
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia PA 19104-6389

Abstract

A human motion planning method called *posture planning* is described that addresses how an agent controls and coordinates body parts to achieve a given task in the Cartesian task space.

1 THE PROBLEM

Animating human bodies with respect to designed workspaces helps designers evaluate their design decisions. Motion planning is needed to generate motions to be animated. As an example, consider an agent who stands in front of a table (Figure 1) and is given a goal of *picking up* the block (Figure 2), which is under the table. Our goal is to find a motion plan by which the agent approaches the table, grasps the block, and lifts it up.

The fundamental problem of motor control is the problem of *degrees of freedom*, that is, how the body controls the massively redundant degrees of freedom of the body joints. The body postures can be uniquely represented in terms of joint angles directly. But there are 88 joint degrees of freedom in our body model (not counting fingers) and we do not know how the body actually controls massively redundant degrees of freedom.

The degrees of freedom problem is solved by sufficiently constraining the joint degrees of freedom by means of constraints imposed on the body. The body constraints are obtained from three sources: (i) the structural and physical properties of the body, (ii) the environment (e.g., obstacles), and (iii) the goals of the agent. Posture planning is a process to identify and solve these body constraints that are changing over time.

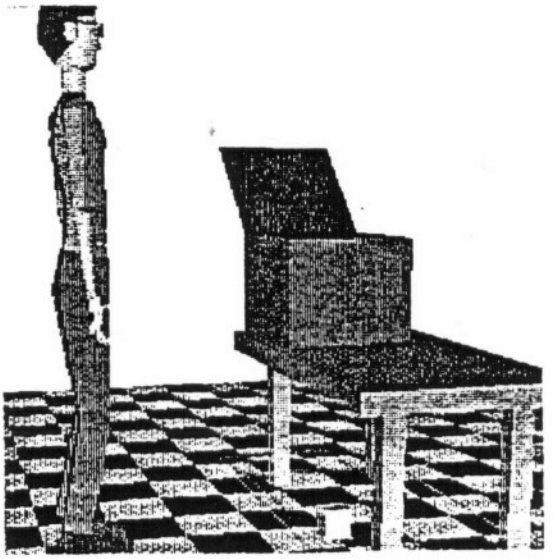


Figure 1: The Agent In Front Of The Table. A Small Block Is Under The Table.

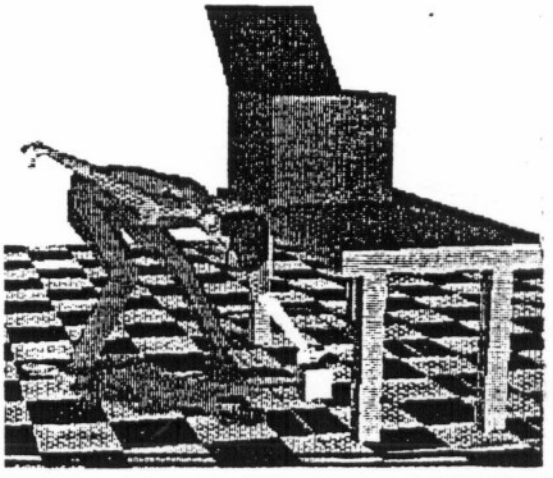


Figure 2: Reaching and Grasping The Block Under The Table.

2 TASK-SPACE CONTROL PARAMETERS

To control the redundant degrees of freedom of the joints, we represent body constraints in terms of higher-level control parameters called *task-space control parameters*. Three kinds of task-space control parameters are posited: *control points*, *control vectors*, and *pivot joints*. *Control points* are important points on the body, e.g., the feet, the pelvis, the head, and the hands. *Control vectors* are important vectors defined on the body to control orientation of the body. For example, the torso upward vector is a control vector for controlling the bending orientation of the upper body. The control vectors include *pelvis-forward-vector*, *righthand-palm-upvector*, *rightfoot-forward-vector*, and *head-view-vector*. *Pivot joints* are joints on the body relative to which control points/vectors are moved. At a given moment, only some of the task-space degrees of freedom are relevant, which are determined by a set of primitive motions selected to achieve given goals. The posture is viewed as a process that modifies *postural states* of the body using given motion strategies. Postural states of the body are defined by the values of the task-space control parameters identified above. Given values of the control parameters, a body posture that satisfies them is found by a robust inverse-kinematics algorithm (Zhao 1989) that formulates the body positioning problem as nonlinear optimization over the joint space of the body.

3 POSTURE PLANNING STRATEGIES

Motion strategies are obtained using gross-level structural properties of the body. Examples of them are as follows:

(1) A hand can be stretched to the ground by bending the upper body, while the pelvis is lowered.
(2) To reach an object, the agent tends to stretch his arm as much as possible while bending the pelvis as little as possible.
(3) When stretching a hand to reach the ground from the standing posture, the agent bends the upper body at the pelvis rather than lowering the pelvis (by bending the knee).
(4) When orienting the body along the vertical axis, stepping is triggered to avoid twisting of knee joints.

Using the motion strategies, the planner selects a partial sequence of primitive motions of control parts/vectors using a standard planning method (Chapman 1987). A motion of a control point or vector is primitive if it has a single moving or rotation direction, respectively. Selected primitive motions are mentally simulated to determine whether selected motions would satisfy the collision avoidance constraints. When a stepping motion is planned, the bounding box of the whole body is used to test collision between the body and obstacles. When other motions are considered, collision of the end effector (a hand), the head, and the torso is tested. That is, collision of the elbow is not considered at planning stage. The assumption is that a workspace for the end effector is designed so that it may provide enough free space for the elbow if it provides the free space for the end effector. Collision is determined by checking if the polyhedral sweeping volumes generated by the end effector, the head, the torso intersect obstacles. If a planned motion causes the end effector, the head, or the torso to collide with an object, the face of that object that is in the way of the sweeping volume is identified. Then the motion is modified so that the sweeping volume would pass by the boundary of that face. The majority of robot motion planning methods (Lozano-Perez 1987, Ching 1992) use the *joint-space* motion reasoning. That is, they assume that the goal configuration of the body is given in terms of a sequence of joint angles and constructs the free *joint-space* (the set of joint angles at which the body does not touch obstacles) to find a collision-free path of the body. The posture planner complements the robot motion planner by providing a feasible macro-level path and by finding a goal posture of the body using heuristic motion strategies.

Acknowledgements

This research is partially supported by Lockheed Engineering and Management Services (NASA Johnson Space Center), MOCO Inc., NSF CISE Grant CDA88-22719, and ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Human Engineering Laboratory, Natick Laboratory, TACOM, and NASA Ames Research Center.

References

Chapman, D. Planning for Conjunctive Goals. *Artificial Intelligence*, 1987, 32:333-377.

Ching, W. and N. Badler. Collision-free Path and Motion Planning for Anthropometric Figures. Submitted to the *SIGGRAPH*, 1992.

Lozano-Perez, T. A Simple Motion-Planning Algorithm for General Robot Manipulators. *IEEE Journal of Robotics and Automation*, Vol RA-3, No. 3, June 1987.

Zhao, J., & N. I. Badler. *Real Time Inverse Kinematics with Spatial Constraints and Joint Limits.* Technical Report MS-CIS-89-09, Computer and Information Science, University of Pennsylvania, PA, 1989.

B Automatic Synthesis of Simplified 3-D Models from Detailed Data

Automatic Synthesis of Simplified 3D Models from Detailed Data

A Proposal

Eunyoung Koh

March 30, 1992

Abstract

The goal of this research is to develop a system which enables automatic construction of the detail hierarchy for complex objects in order to provide progressive object details for displaying complex geometric environments. Based on the constructed detail hierarchy, our system is able to display objects in various detail levels depending on the viewing position. Various possible methods towards geometric detail reduction are surveyed as an effort to find an appropriate method to be used in our hierarchy construction module. Having decided to use superquadrics with global deformations and blobby models for our system, we present an automatic hierarchy construction scheme using these models. The mathematical formulations and recovery algorithms of these models are explained. In particular, we propose several improvements over Muraki's blobby model recovery algorithm which will result in significant gain in speed and efficiency. We deliver a detail hierarchy traversal algorithm which utilizes frame coherence. We discuss our selection of display metric and other issues related to display of the detail hierarchy.

Contents

Chapter 1

Introduction

1.1 Motivation

Conventional representations of a geometric environment describe the objects in full detail. The conventional representation scheme often involves many problems when the objects are displayed. Firstly, all the objects in the environment will be computed for clipping, hidden-surface elimination, and illumination. Even though an efficient clipping computation is applied to those objects which are not in the view range, the viewer can hardly resolve all the details which were displayed on the screen with heavy computational overhead. This unresolvability is a combined result of the viewer's eye movement, limited resolution of the display device and the relatively small area the object occupies on the screen.

Another problem occurs when too much detail falls into a small area of the display, hence generating aliasing effects [Amanatides 87]. Aliasing often makes the image unattractive and distracting to the viewer. Finally, we have to resolve the huge storage requirement for the large database in order to display a complex scene. Users occasionally have to divide the environment into subsets and composite the intermediate images in an ad-hoc way to cope with the storage requirement. This not only involves

unnecessary overhead but often gives an unsatisfactory final image resulting from some reflections and shadows being left out.

All these problems can be solved when we adopt the hierarchical representation of object details as proposed by Clark which will allow transition among different levels of detail for the objects [Clark 76]. The hierarchical database can be developed by designers before display time, either manually or semi-automatically [Rubin 80, Rubin 82, Feiner 85]. The hierarchy has been built through a laborious process and in an ad-hoc manner. It has been largely an unsolved problem to create a relation automatically between the data structures and the rendering process so that exactly the necessary level of detail is available [Badler 84].

A *modeling hierarchy* is obtained when an object is modeled. It records the way complex objects are built up out of simpler parts. When animating natural figures consisting of rigid limbs connected by joints it is usual to model a figure as a tree of dependent parts. Representing objects using such a modeling hierarchy is convenient for positioning objects and their components in space and for moving objects relative to one another. In addition, they offer considerable memory savings when objects and object components occur several times in a scene. In the modeling hierarchy, the leaf nodes typically contain nodes, edges, polygons, and possibly surface patches.

Clark extended this normal object hierarchy [Foley 90] to include sub-hierarchies which contain objects modeled in greater and greater detail [Clark 76]. He gave a recursive descent algorithm for searches and traversals which proceed only down to the smallest resolvable level of detail.

Clark's hierarchical representation, which will be called *detail hierarchy* in the rest of this article, specifies the entire environment as a tree where the root is interpreted as the whole environment. Each node in the tree is either a collection of objects or an object at a certain level of detail. Every arc in the tree contains information about a transformation which prescribes the relative position of the child from its parent node. There are two types of arcs in the tree: those that represent pointers to child objects whose orientation

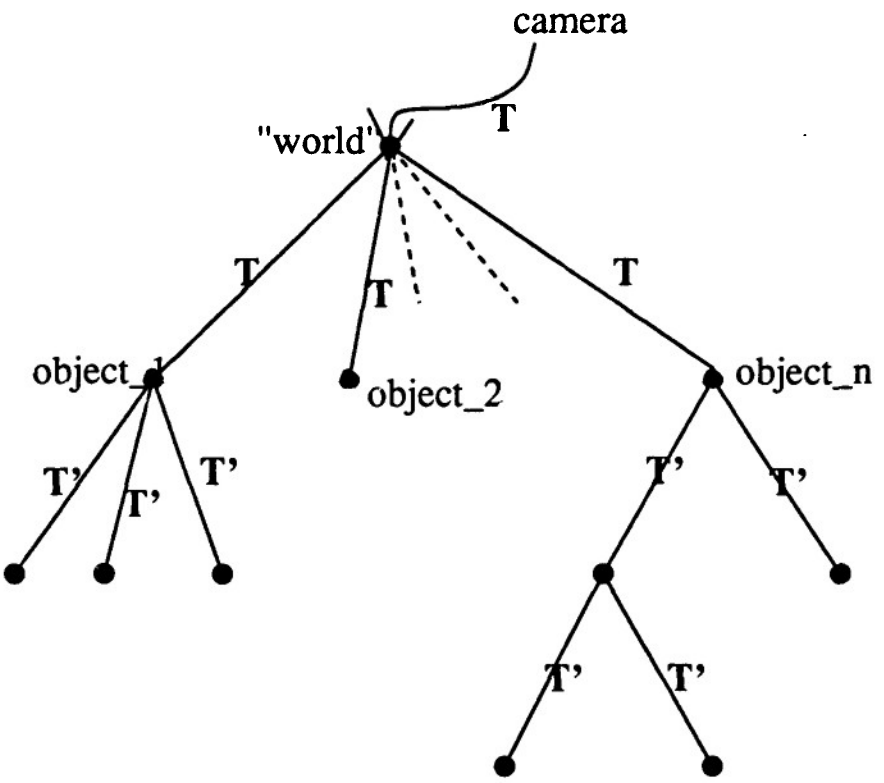


Figure 1.1: A detail hierarchy of an environment

and placement are defined relative to its parent and those that account for pointers to more detailed structure which collectively define a more detailed specification of the object.

Each nonterminal node in this detail hierarchy represents a sufficient description of the object if it covers no more than some pre-specified area on the display. If the description formed at a level is inadequate because it covers a larger area on the screen, the object description is obtained from their child nodes at lower levels which supply more details. An example of using the hierarchy to describe an environment is shown in figure 1.1. In the figure, T' is a transformation to more detailed description of an object and T is an object's transformation relative to its parent node.

As we mentioned previously, the conventional object representation almost always conveys certain hierarchy information. However, the detail hierarchy is different from the modeling hierarchy in the sense that the detail hierarchy does provide geometric

approximations to the objects which provide intermediate details. On the other hand, in the modeling hierarchy only the leaf nodes contain the geometry information of the objects, providing only one level of detail of the objects.

For example, a good geometric representation of a human body figure typically requires over 6,000 data points. The computer resources required to display and manipulate such large amounts of data are significant and often cause the figures to lose the real-time motion capability. It is desirable to have the human figure rendered as a couple of blocks or a single rectangular polyhedron when the area it occupies is relatively small.

The following table in figure 1.2 shows the polygon processing capability of current graphics hardware technology. In this table, we list series of personal IRISes [SG 91] for the hardware. With the higher end models, we can have approximately 30K polygons processed per second. Thus approximately only 30K / 60 hz = 0.5K polygons (or 1K polygons when employing the interlacing technique) can be processed in real-time. This is the number of polygons in the geometric environment which users can manipulate in an interactive manner without noticing time delay in display. Therefore, it is imperative to develop a system which can provide multiple levels of detail for objects.

A typical application of this detail hierarchy can be found in the area of computer controlled simulators such as flight simulator which need to render complex images. The flight simulator displays a dynamic, three-dimensional, out-of-the-window view of the scene in real time while responding to operator input from the command and control system. One of the major modules in a flight simulator is a scene manager. In the scene manager, the objects are usually represented in several levels of detail, with the detailed version shown only when the viewer is sufficiently close to the objects. This module is responsible for retrieving from mass storage the database objects within the panorama of the current pilot position and providing the appropriate levels of detail of these objects for further processing. The scene manager ought to be provided with the ability of ansition from one level of detail to the next one in a smooth manner [Yan 85].

	Base Sys. 8bit color no Z buffer	G 24bit color 24bit Z	TG 24bit color 24bit Z
4D/35	6K Polygons 16K Triangles 92K Vectors	6K Polygons 16K Triangles 92K Vectors	29K Polygons 40K Triangles 219k Vectors
4D/30	6K Polygons 16K Triangles 92K Vectors	6K Polygons 16K Triangles 92K Vectors	29K Polygons 40K Triangles 219K Vectors
4D/25	6K Polygons 16K Triangles 92K Vectors	6K Polygons 16K Triangles 92K Vectors	28K Polygons 36K Triangles 204K Vectors

Polygons/sec =
10 x 10 (100 pixel).
full 24-bit color,
unlighted, Gouraud-shaded,
Z-buffered,
arbitrary orientation.

Triangles/sec =
10x10 (50 pixel) mesh,
full 24-bit color,
unlighted, flat shaded,
Z-buffered,
arbitrary orientation.

Vectors/sec =
10 pixel, conncted,
full 24-bit color, 3D,
arbitrary orientation.

Figure 1.2: Graphics configuration of personal IRISes

Another arising area of application of the hierarchical representation of object detail is in virtual reality. Due to drastic cut-down in the price of computer hardware, virtual reality equipments have become affordable to a wider class of users. Virtual realities or virtual worlds help researchers visualize and manipulate complex data and help architects show complex building ideas to clients. The headgear mounted on the viewer includes a sensor to detect head orientation. If the computer is powerful enough or the scene itself is simple enough, the image can be updated with about 30 frames a second. This is fast enough to give viewers the impression that the scene is changing smoothly. However, the complexity of scene which can be managed by current or near future computer graphics hardware technology is limited. This complexity problem can be compensated by using the hierarchical representation of object details.

1.2 Previous Work on Representing Objects with Detail Hierarchy

As mentioned in the above, Clark introduced the object representation with detail hierarchy [Clark 76]. Though Clark mentions that the hierarchy structure can be constructed by using a bottom-up approach from the most detailed description of objects, there was no proposal for automating this process.

Rubin and Whitted propose a scheme whereby the object space is represented only by bounding boxes [Rubin 80, Rubin 82]. The leaf nodes are rectangular parallelepipeds which are oriented to minimize their size and formed into an approximation of an object component. The creation of the bounding boxes is a rather tedious process that requires a human operator. Their approach using rectangular parallellepipeds with affine transformations contains a total generality of describing an arbitrary object but they are far from being efficient in representing arbitrary objects.

In order to reduce time spent on ray object intersections in ray tracing techniques,

many acceleration techniques used structure information which the scene bears [Goldsmith 87, Snyder 87, Weghorst 84, Kay 86]. Others subdivided space into an octree to speed up the ray-object computation [Glassner 84, Glassner 88]. Building tree hierarchy for ray-tracing, however, was done by merging objects hierarchically into larger groups by some bottom-up pruning of existing definitions of the objects. They did not attempt to provide simplified description of each object.

Recently, Blake presented a viewer centered *metric* for computing adaptive detail as a methodical way of using adaptive detail [Blake 90]. He was successful in providing some theoretical foundations underlying the practice of adaptive detail display of complex scenes but did not furnish any formula which can be used as a generic tool for rendering adaptive detail. Also, construction of the detail hierarchy for complex geometric environments was not addressed in his work.

1.3 Statement of the Problem

The goal of this research is to develop a system which enables automatic construction of the detail hierarchy for complex objects in order to provide progressive object details. Based on the constructed detail hierarchy, our system is able to display objects in various detail levels depending on the viewing position. Viewers can also specify which objects are more important thus need to be displayed in more detail by setting high values to the priority metric of the objects. The overall system configuration is given in figure 1.3.

Given the input of a complex geometric environment, the hierarchy construction module performs as an off-line process and generates an extended geometric environment with multi-level detail. Our automatic hierarchy construction scheme uses shape recovery algorithms to build simplified models from a detailed object data. It also defines a metric threshold value for each level of detail so it can be used by the display manager.

The display module manages the environment with detail hierarchy and determines

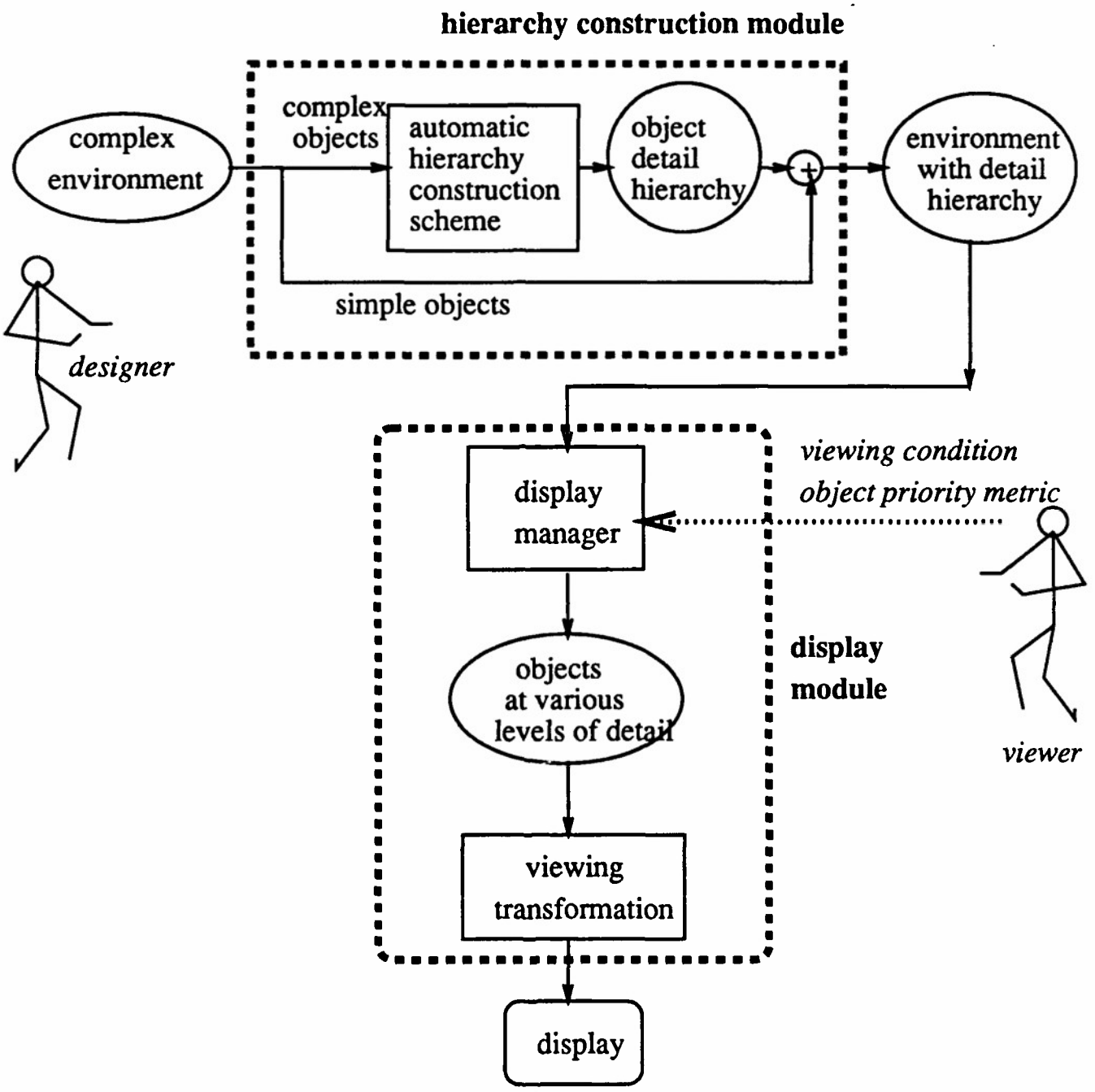


Figure 1.3: A system configuration

the adequate levels of detail according to the viewing condition and the object priority metric values specified by the viewer. Display manager computes a metric value for each level of detail and compare it with the metric threshold values to decide whether the object needs more or less detail.

In our current research, we only consider static objects and do not attempt to apply our algorithm to deformable or articulated objects. Attributes such as surface color apparently play a very important role in users' perception of the object. Handling attributes requires research on determining the importance of each attribute in human perception of objects. For example, a red light, though small, may be clearly visible at detail levels which would normally subordinate its geometry. We do not attempt to study the effects of attributes; instead we will only concentrate on the importance of shape given by the geometric information at this time.

1.4 Organization

In chapter 2, we review various possible methods towards the geometric detail reduction. This survey was conducted in an effort to find an appropriate method to be used in our hierarchy construction module. Each method holds its own object representation scheme. Thus we analyze each method and its object representation scheme to judge its suitability in our application. The methods which will be discussed include implicitly defined surfaces (both iso-surfaces and generalized superquadrics surfaces), spatial subdivision methods, parametric surface patches, etc.

Having decided to use superquadrics with global deformation and blobby models for the geometric detail reduction, we present more details on the automatic hierarchy construction scheme using these models in chapter 3 in a step by step fashion. The mathematical formulations and recovery algorithms of these models are explained in chapter 4. In particular, we propose several improvements over Muraki's blobby model recovery algorithm which will result in significant gain in speed and efficiency. We also

discuss the rendering of blobby model and explain a polygonization algorithm.

We discuss the display module in chapter 5. We deliver a detail hierarchy traversal algorithm which utilizes the frame coherence. This algorithm tries to minimize the number of node visits when the image needs to be updated because of moving objects or slight change in viewing condition. We also discuss the selection of metric and other issues related to display of the detail hierarchy.

Finally, we give a summary of our research problem and explain work in progress. We conclude with a section detailing the resulting contributions from this research.

Chapter 2

Methods for Geometric Detail Reduction

2.1 Overview

One of the most widely used representation scheme for geometric objects is the boundary representation in which nodes, edges and faces are explicitly stored. Considering the goal of our work which is to generate intermediate geometric approximations to objects, an obvious and intuitive approach is to directly manipulate these boundary representation and smooth out some of the details. However, direct manipulation of boundary representation to generate simplified models from detailed input data poses a more complex problem than it appears and so far no satisfactory algorithms have been reported. In this work, we would like to approach the problem of reducing the geometric details of an object by identifying the outstanding features of the object and representing them in certain simplified formulations so that the minute surface details will be smoothed out.

There are many surface or volume modeling schemes which have been used in computer graphics and machine vision community. Moreover, work in model recovery has generated many algorithms to recover volumetric models from input data which

were formulated as optimization problems [Solina 87, Muraki 91]. An alternative approach to the approximation of surfaces was taken by surface fitting using parametric Bezier surfaces [Schmitt 86, Cheng 89]. An effort to incorporate the global shape parameter with the local degrees of freedom has produced deformable superquadrics [Terzopoulos 91, Metaxas 91]. Implicit surfaces were generalized by using modal deformations and displacement maps to provide local and fine surface detail by offsetting the surface of the solid along its surface normals [Sclaroff 91, Pentland 91]. The research on model based shape recovery suggests a solution to our problem because it provides a method of distinguishing features and averaging out surface detail at the same time.

Requicha [Requicha 80] provides a list of the properties desirable in a solid representation scheme. Although our interest does not exactly fall into the solid representation scheme, the list can serve as criteria for comparing and analyzing many available object representation methods and finally help to select one scheme for our purpose.

Some of the items on this list are:

- The *domain* of representation must be large enough to allow a useful set of physical objects to be represented.

- A representation should be *compact* to save space, which in turn may save communication time in a distributed system.

- A representation should allow the use of *efficient* algorithms for computing desired physical properties, and most important for us, for creating images.

The criteria which are particularly important in our modeling were not enumerated in Requicha's list. For example, our object representation should be able to provide adequate intermediate levels of detail which can be used to incrementally approximate the object. The representation should also provide an easy and efficient way to compute these intermediate levels.

In this chapter, we review the available object representation schemes and select the best scheme based on the given criteria.

2.2 Voxel Representation

Under this scheme, objects are decomposed into identical cells named *voxels* (volume elements) which are arranged in a fixed and regular grid. Representing an object is easy under this scheme since all required is to decide which cells are occupied and which are not [Foley 90]. Editing with voxels is not intuitive to users. Thus, voxel representation is usually recovered from other commonly used representation such as boundary representation or a set of data points obtained through an image processor. It is often used in biomedical applications to allow volume visualization where the data are usually obtained from sources such as computerized axial tomography (CAT) scans.

Under this scheme, operations such as deciding whether a cell is inside or outside of the solid and determining whether two objects are adjacent are simple to carry out. They have thus been used in hardware-based solid modeling systems intended for applications in which the gain in speed of Boolean set operations outweighs the coarseness of the resulting images.

However, voxels do not allow partial occupancy though fractional *density* values may be stored in non-binary voxel spaces, thus many solids can be only approximated. Storage of the voxels require enormous space since up to n^3 occupied cells are needed to represent an object at a resolution of n voxels in each of the three dimensions.

2.3 Octrees

Octree representation is a hierarchical variant of the voxel representation. It is designed to remedy the issue of demanding storage requirements of voxel representation

[Samet 88a, Samet 88b, Carlbom 85]. In this representation, the entire object space is divided repeatedly into cubes or rectangular parallelepipeds resulting in a tree structure. The leaf nodes do not contain primitives such as edges and polygons but are rather homogeneous cells which give the occupancy information. Octrees approximate the object components by repeated subdivision into cubes or parallelepipieds to some degree of precision.

The spatial decomposition provides a representation applicable to a wide class of objects and allows geometrical properties to be computed rapidly. Octrees have been generalized to represent polyhedral objects and named as polytree[Carlbom 85] or extended octree[Brunet 90]. Octrees do allow the detail in the projected image to be varied by changing the depth to which the octree is processed. This can be done adaptively depending on the resolution of the display [Sandor 85].

Octrees do not provide the memory savings offered by object decomposition, nor does it provide any structure for managing or interacting with components of a complex object. The storage requirement is still severe as compared to other representation schemes. Besides, octrees can only represent bounded solids. Moreover, the rectilinearity is definitely not an advantage in object appearance.

2.4 Superquadrics

Superquaric surfaces were introduced to computer graphics to allow complex solids and surfaces to be constructed and altered easily from a few interactive parameters [Barr 81]. Barr developed solid modeling operations which simulate twisting, bending, tapering or similar transformations of geometric objects [Barr 84]. The deformations extend the shape of solid primitives and allow modeling to be done by intuitive and easily visualized operations.

Solina formulated an optimization problem which can recover the superquadrics with global deformations such as bending and tapering from input data [Solina 87]. Based

on Solina's recovery procedure, Gupta developed a volumetric segmentation algorithm which can recover the structured part hierarchy of an object [Gupta 91]. We will explain superquadric models in more detail in a later chapter.

2.5 Dynamic Superquadrics with Local and Global Deformations

Terzopoulos and Metaxas presented a physically-based approach to fitting complex 3D shapes using dynamic models [Terzopoulos 91, Metaxas 91]. They formulated *deformable superquadrics* which incorporate the global shape parameters of a conventional superellipsoid with the local degrees of freedom of a spline. The local deformation parameters help to reconstruct the details of complex shapes which the global abstraction misses. They formulated the model fitting to visual data by transforming the data into forces and simulating the equations of motion through time to adjust the translational, rotational, and deformational degrees of freedom of the models.

They argue that geometry is often insufficient for analyzing the motions and interactions of complex objects. A model based on computational physics is suggested to remedy its shortcomings. Thus in addition to geometry, their models includes simulated forces, masses, strain energies, and other physical quantities.

This complex formulation may generate well-fitted superellipsoid at the expense of heavy computation overhead. To generate the intermediate geometric approximations in our system, one does not require the full dynamics property that this fitting procedure adopts. Also their models have shown to work only on pre-segmented data. Thus, we opt for a different fitting procedure which is simpler and more efficient.

2.6 Modal Deformations with Displacement Maps

Pentland and Sclaroff used modal deformations to describe the overall shape of an object where displacement maps are used to provide local and fine surface details by offsetting the surface of the solid along its surface normals [Sclaroff 91, Pentland 91]. The advantage of this approach as a modeling scheme is that collision detection and dynamic simulation become simple and inexpensive even for complex shapes. They also provided a method for fitting such models to three dimensional point data by determining bother the deformation parameters and a displacement map.

The distance maps were designed to describe shape details. They are not adequate to represent a whole object which consists of many parts. Moreover, in their ThingWorld system, the displacement maps are represented by a regularly spaced grid in the surface's parametric space. Therefore, the detail which their models can capture depends on the grid resolution rather than the inherent shape detail of the object. The blobby model which we will discuss later will remedy both of these problems.

2.7 Spline Surfaces

Schmitt *et al* proposed a top down method for the problem of surface fitting from sampled data [Schmitt 86]. This method is based on an adaptive subdivision approach. It begins with a rough approximating surface and progressively refines it in successive steps to adjust the regions where the data is poorly approximated. Their method constructs a parametric piecewise polynomial surface representation. The surface fitting is effected through a use of subdivision techniques. The subdivision creates new vertices which are then positioned so as to achieve a closer approximation of the underlying data.

The method has been implemented using a parametric piecewise bicubic Bernstein-Bezier surface possessing G^1 continuity. An example of a surface fitting to a human head data was shown in the original paper.

One advantage of this approach is that the refinement is essentially local, reducing the computational requirements which permits the processing of large databases. Furthermore, the subdivision method provides a hierarchical representation of the surface as a quadtree-like structure which may be utilized in our system.

However, for surfaces with minute details or undulating surfaces, this algorithm may actually try to fit to those details and hence generate a large number of surfaces in the process. This is an undesirable feature considering the goal of our system design which is to generate intermediate geometric approximations typically consisting of fewer polygons. Therefore we choose to adopt a different modeling scheme.

2.8 Potential Surfaces

Potential surfaces are iso-surfaces of potential energy emerging from a number of origins. They have the property that the iso-surfaces are smoothly continuous even though the origins are separated. They are simple to edit since all required to do is to create, move and delete the corresponding origins, and change a few parameters in the potential field formulation [Wyvill 86, Blinn 82].

The *blobby model* introduced by Blinn is one form of potential surface [Blinn 82]. Muraki has developed a recovery procedure for this blobby model from input data[Muraki 91]. By adjusting the number of blobs used in the fitting process, we can obtain a hierarchy of intermediate representations. Wyvill *et al*'s *soft object* is different from the blobby model in the formulation of the field function. That is, the soft object adopted a polynomial field function so that the function do not influence any point beyond a certain distance away.

2.9 Distance Surfaces and Convolution Surfaces

Distance surfaces are a generalized form of potential surfaces which allow *polygonal skeletons* instead of *point skeletons* [Requicha 83, Payne 92]. This is done by computing the potential from only the nearest point of the polygon. The distance surface gives the union of the volumes generated by all the individual points of the collective skeleton. Thus a skeleton consisting of two line segments may generate bulges at the joint.

Convolution surfaces have almost the same shape as distance surfaces except that they generate smooth blending by convolving the skeleton with a three-dimensional Gaussian filter kernel [Bloomenthal 91].

So far no recovery procedure has been developed for this class of modeling schemes. It is conceivable that the recovery of the distance surfaces or convolution surfaces may be difficult due to their complex formulations.

2.10 Medial Axis Transform

Blum has introduced a transformation, known as the symmetric axis transform or medial axis transform that decomposes a figure in 2-D into simpler figures [Blum 78]. In an attempt to extend the medial axis transform, Nackman generalized the mathematical tools used by Blum to three dimensions for further study of Blum's transform in three dimensions [Nackman 82]. However, there was no algorithm provided for shape description and we found that application of his work is not yet feasible.

O'Rourke and Badler presented an algorithm which decomposes a three-dimensional object, specified by a set of surface points, into a collection of overlapping spheres [ORourke 79]. This spherical decomposition permits the computation of points on the symmetric surface of an object in 3-D. All the sphere centers resulting from their sphere decomposition algorithm lie on the medial surface of the object. However, the sphere centers do not necessarily cover the symmetric surface. Thus, for a very complex

object, the sphere centers represent a sampling from its complicated symmetric surface. Recovering symmetric surfaces from the scattered surface points has not been addressed.

2.11 Summary

Our goal here is to find a suitable modeling scheme which fits into our application, namely, building a detail hierarchy for displaying complex environments. To justify our selection, we compared the aforementioned modeling schemes based on the given criteria (figure 2.1).

There was no known recovery method for the convolution surface and distance surface. The sphere decomposition algorithm by O'Rourke and Badler allows the computation of symmetric surface points. However, an effective sphere search algorithm which can minimize the number of recovered spheres was not provided. Therefore, in the worst case, the number of spheres can match the number of surface points.

One of the basic problems of the spatial subdivision methods such as voxels or octrees was their huge storage requirement. In addition, each cell is a box with an intensity value. The cell cannot convey the surface tangent information which is very important in rendering the shape with visual fidelity to the modeled object. Although we are considering only static objects in this research, we would like to see a scheme which can be easily extended to articulated figures. The space subdivision methods do not have this desirable property since they cannot deliver any structure information.

Superquadrics with global deformations were adopted in our scheme to represent coarse object descriptions because of its simplicity and descriptive capability of shapes. Deformable superquadrics with global and local deformations and generalized implicit surfaces, on the other hand, were considered too complex to use for coarse descriptions. As mentioned before, they work only on pre-segmented parts so they can not be used to describe complex objects which may contain structured part hierarchy.

Criteria / Models	Recovery Procedure	Storage for Recovered Model	Model Representation Scheme	Other Comments	Suitabillity for creating a hierarchy
Distance Surface / Convolution Surface	No known Method Exists				
Medial Axis Transform	O'Rourke and Badler	Sphere Sets	Solid (Union of Spheres)	More spheres than original surfaces	Creates Too Many Primitives
Spatial Subdivision (Voxel,Octree)	(inherent structure)	Every voxel or octant -> huge	Filled Solid	no structural information	Huge Storage Requirement
Superquadrics with Global Deformation	Solina	Sq. parameters, Deformation parameters	Solid (Superquadrics)	Relatively Simple for Part Fitting	Useful for Coarse Approximation
Deformable Superquadrics	Terzopoulos and Metaxas	Sq. parameters, Deformation Information	Solid (Generalized superquadrics)	Hard to fit on an object with many parts	Too Complex for Parts
Modal Deformation	Pentland and Sclaroff	Sq. & Modal Deform. param. Distance Map	Solid (Generalized superquadrics)	Hard to fit on an object with many parts	Too Complex for Parts
Surface Patches	Barsky et al.	Control Points	Surface	Undulating Surfaces -> More Polygons	Fitting to Smooth Surface
Implicit Surfaces	Muraki	Parameters for Blobby Model	Solid (Implicit Surface)	Performs low-pass filtering	Useful for Good Approximation

Figure 2.1: A table comparing the merits of different representation schemes

An adaptive fitting algorithm for surface patches given by Schmitt *et al* provides a possible hierarchy construction scheme for smoothly defined objects. However, the smoothness assumption is not satisfied by many geometric objects in graphics application. Also they require the input data to be regularly spaced in rectangular grids.

We have chosen to use the blobby model to build intermediate geometric approximations for complex objects. The blobby model can capture the part structure of the input object and give an effect of low pass filtering of the input data. The recovery algorithm for this model provided by Muraki reveals some shortcomings at the current stage. We propose a number of enhancements to Muraki's work such as using superquadrics as primitives. Especially we try to speed up the recovery process of multiple primitives by performing residual analysis and proposing other improvements. More details will be discussed in chapter 4.

Chapter 3

Synthesis

3.1 Overview

As we have discussed in the previous chapter, superquadrics with global deformation will be used to construct the intermediate geometric approximation to the input object. Due to their inherent limitation in the object shape vocabulary, however, superquadrics are not adequate to approximate an object with intricate detail to a high level of fidelity. Thus, we will also use the blobby model for the intermediate geometric approximation to obtain a better approximation to the original object than the superquadric surfaces. The formulation of the blobby model facilitates this better approximation: it is an implicitly defined surface with a summation of potential functions.

In this chapter, we are going to explain how superquadrics and blobby model are used in our detail hierarchy construction scheme. As we noted in the previous chapter, recovery algorithms from the input or range data points are available for these models. We use the shape recovery algorithm to fit these models to the given input object. The recovered models are used as the intermediate geometric approximations to the original object. We fit the models to the input object in various levels of accuracy and construct the detail hierarchy from the recovered models.

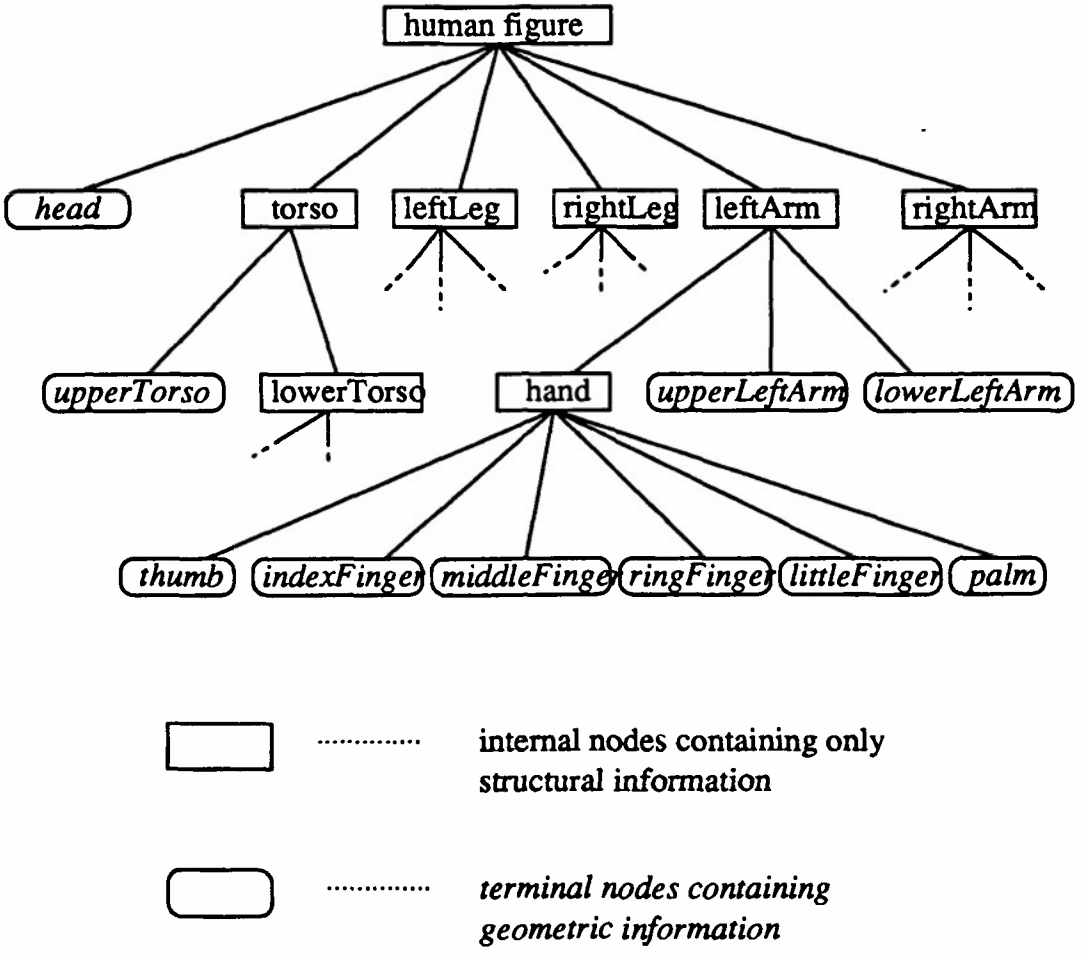


Figure 3.1: An example object with user defined hierarchy: a human figure

The types of figures given as input to our system can be categorized into two classes. Objects in the first class have a user-defined object hierarchy. An example of such a hierarchy for a human figure is shown in figure 3.1. Since we handle only static objects, we assume that this human figure remains in the given posture. Objects created with the editing facility in conventional computer aided design (CAD) systems usually belong to this class. Objects in the second class contain only geometric information with no explicit hierarchy definition. Examples are objects obtained through a range sensing device, a 3-D digitizer, or from a conversion program for CAD systems. From now on, we will call the first class objects as DOH (Defined Object Hierarchy) objects and the second class objects as NOH (No Object Hierarchy) objects.

In the description of the hierarchy construction, we use the term *object* to mean any part of the object for which we are trying to construct coarser levels of detail. Thus an *object* may be a whole figure, a set of segments (or links), or a segment (or link).

The hierarchy construction process can be divided into five steps. At the first step, fitting of a superquadric model with global deformation is conducted at each node of a DOH object or to the input object in the case of a NOH object. By storing the recovered superquadric models at the nodes, we provide coarser description of the object which corresponds to that level of the hierarchy.

By analyzing the squareness parameters of the recovered superquadric model, we can categorize it into a cylinder, round object, cuboid, or ellipsoid. Thus, the next step is to obtain a more simplied model from the recovered superquadric model through the categorization. That is, the recovered superquadric model can be further approximated as a simple cuboid, cylinder, etc. This level of description will be useful when the object is very far from the viewer so it occupies very little area on the screen.

At the third step, we propose to apply the volumetric segmentation in an effort to recover the inherent structured part hierarchy of the object. However, this volumetric segmentation problem has been considered difficult to solve for very complex objects. It is still a subject that is being pursued in computer vision community. This step remains optional at this time.

The next step is applied only to complex objects with more pronounced features which the superquadric model can not capture. In other words, we try to recover more elaborate detail by using blobby models and store them between a superquadric approximation and the given input object. This process is done at the current terminal nodes and increases the depth of the subtree by adding intermediate nodes before the processed terminal node.

The last step is not directly related to the synthesis of simplied models. We further process the hierarchy constructed so far to be used by a display manager by defining the threshold values of the display metric in each internal node. These threshold values

are used in determining the levels of detail to be displayed.

3.2 Hierarchy Construction Process

The hierarchy construction process is depicted in figure 3.2. Each step is explained in the following subsections. Thd new detail hierarchy which is expanded by this process from the given human figure (figure 3.1) is shown in figure 3.3.

3.2.1 Fitting Superquadric Models

The superquadric fitting is done is a recursive descent manner. The algorithm is as follows:

1. Transform *all* the segments in the figure into a global coordinate system.
2. Set $NODE$ = root of the tree.
3. Run a superquadric fitting module on the geometric data that belongs to the whole subtree whose root is $NODE$. Then we store the fitted superquadrics into the data structure of $NODE$.
4. Set $NODE$ to each child of $NODE$ and recursively execute the previous step until the whole tree is traversed.

3.2.2 Classifying Superquadric Models

Though the superquadrics reduce the detail dramatically for complex objects, the resulting polyhedral approximation to the superquadric model can still consist of too many polygons. The actual polygonization of the superquadrics is done by an adaptive method which starts with evenly spaced angles in two dimensions and subdivides the angles if

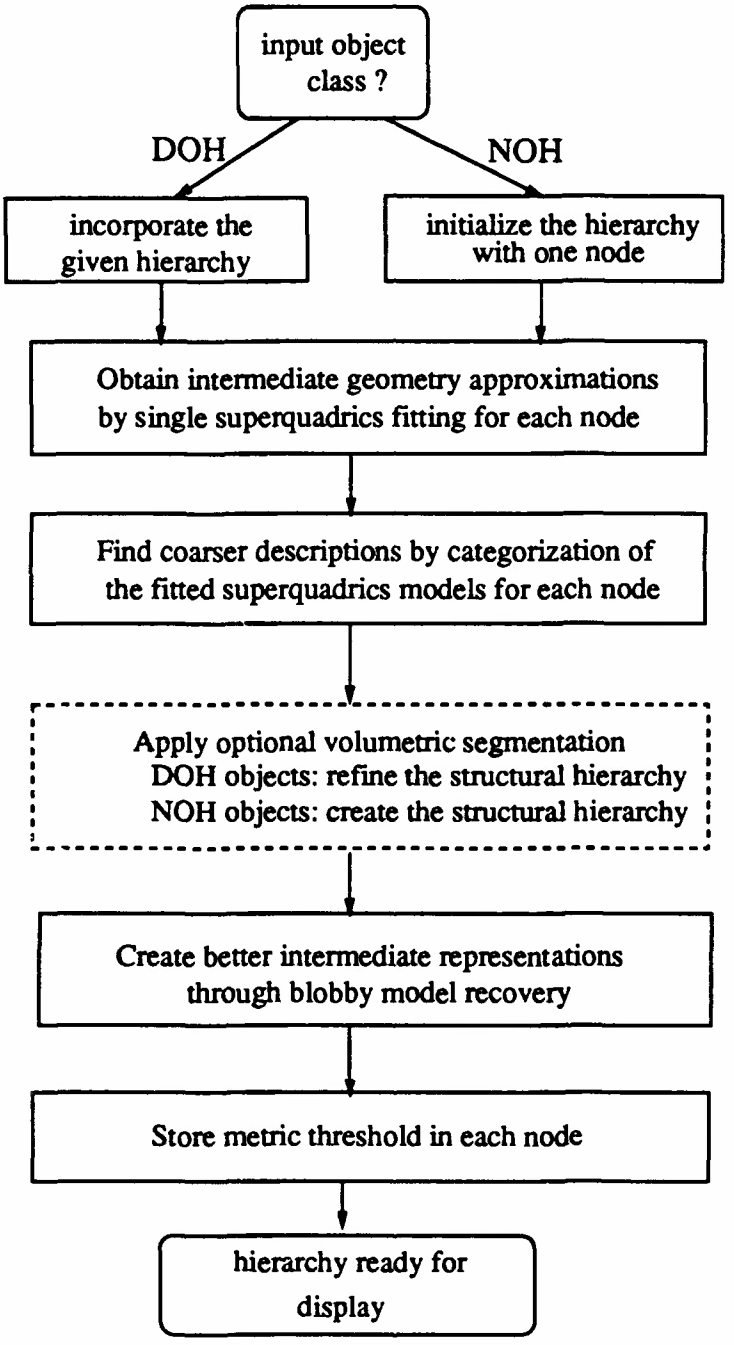
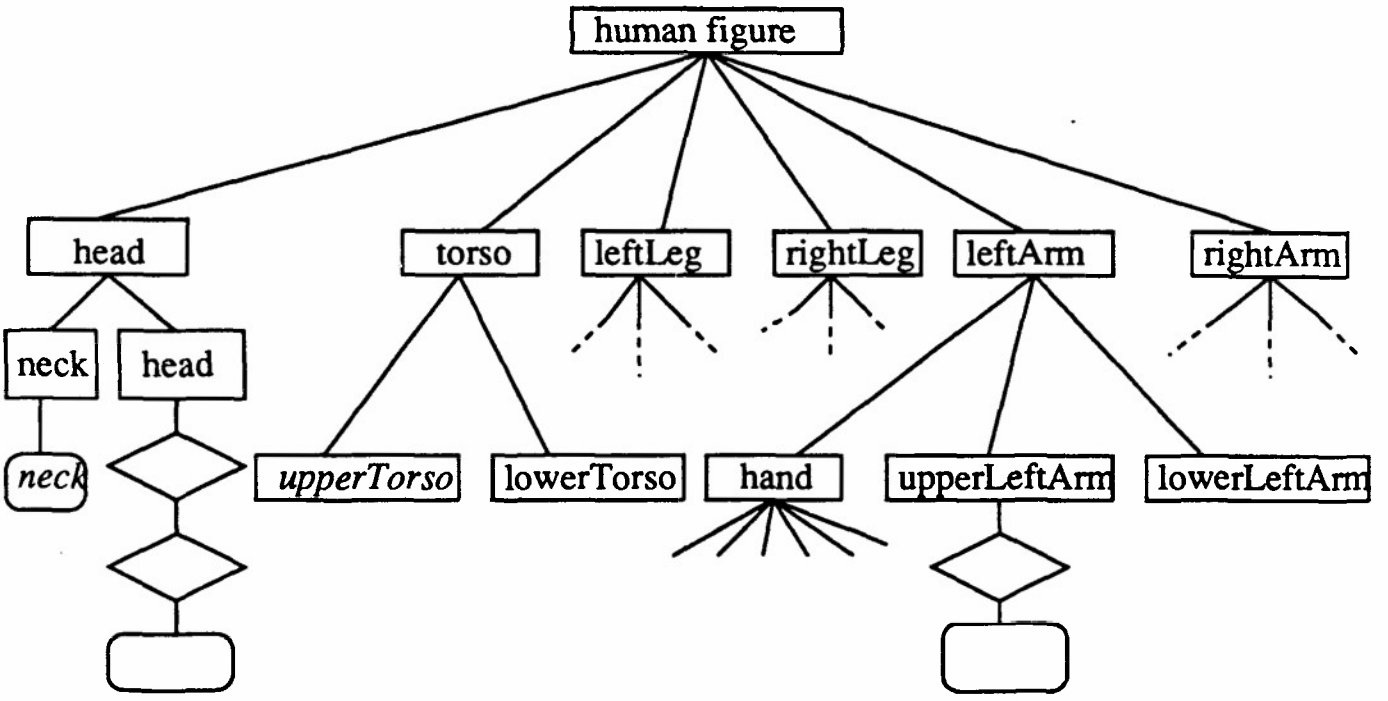
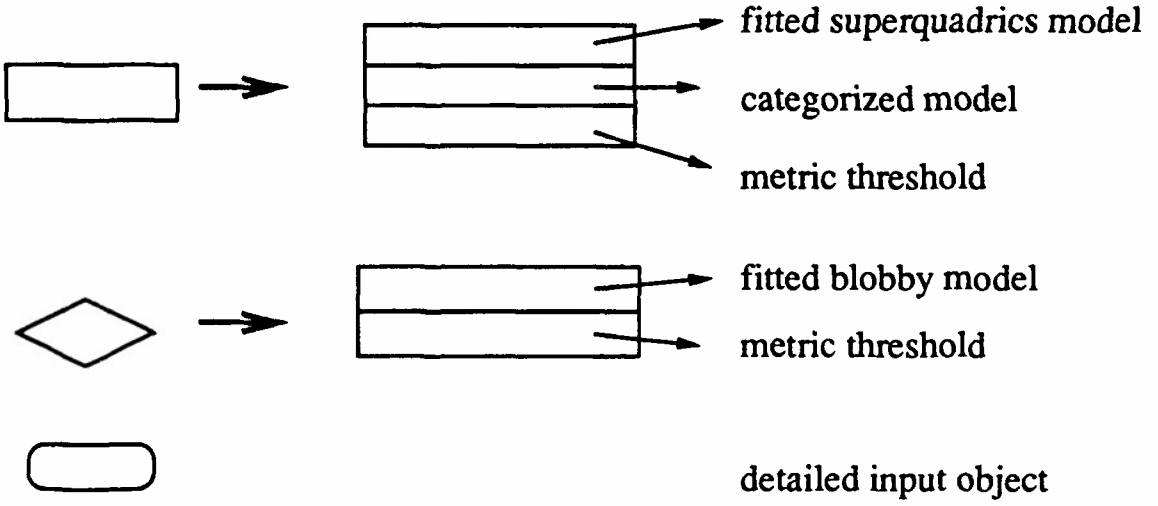


Figure 3.2: Abstract view of hierarchy construction



Node Explanations:



DOH Objects

Figure 3.3: The same hierarchy for the human figure after the intermediate representations are computed and stored in the nodes

the tangents at the two adjacent angles position show a big discrepancy. This will generate more polygons at places with sharp curvature. Therefore, if the object occupies only a few pixels on display (i.e. very far from the view point), it is a waste of computation to display all the polygons. Hence, we propose to add another level of coarser description.

Based on the values of the superquadric model parameters, we can categorize the fitted model into several classes: box, round, edged ellipsoid, and cylinder [Bajcsy 90] (figure 3.4). For example, cylinders can be either circular or elliptical based on the ratio of their two smaller, superquadric size parameters. Then for each class, we can define a simplified polyhedron depending on the size and the parameter values of the fitted model. This simplified polyhedron consists of fewer polygons and hence is more efficient for small display area.

We can extend the notion to have this added level not only at the root of the tree but also at every internal node. The effectiveness of this extra level remains to be verified from an experimentation.

3.2.3 Applying Volumetric Segmentation

The next step is an optional step. We can attempt to apply the volumetric segmentation process to the object. This is to recover the inherent part hierarchy in the object. This is more useful for NOH objects since they do not come with a structural hierarchy. After this volumetric segmentation process, the NOH objects will have a hierarchy defined and hence can be treated similarly as DOH objects.

For DOH objects, since they already come with a hierarchy defined, the segmentation process may serve to *refine* the hierarchy. For NOH objects, since they do not come with a hierarchy, the objective of the segmentation process will then be to *create* a structural hierarchy for them.

Note that the intermediate geometric approximations will be obtained automatically

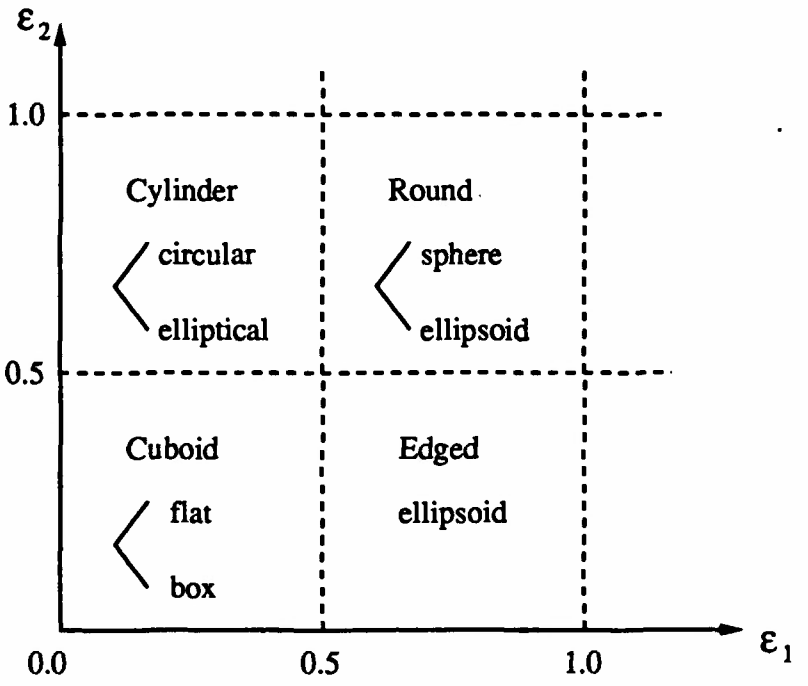


Figure 3.4: illustration of the categorization process

for each node in the subtree created in this segmentation process. This process in effect creates new leaves and increases the depth of the hierarchy tree.

3.2.4 Fitting Blobby Models

For objects that have more pronounced features, the superquadrics fitting process may fail to capture these features into the superquadrics model. For example, let us consider the head of a human figure. The whole head segment will be represented by one superquadric model which can not describe all the features such as the nose and the ears clearly. This may be true even if the volumetric segmentation process is applied.

For these types of leaves that the superquadrics fitting process does not generate a good fit for the original geometric data, we can generate a better intermediate representation using a different fitting process, the blobby model fitting process. This class of model has the capability of representing more detail features. Blobby model will be described in more detail in chapter 4. The model generated will be stored into the data

structure of the node as before.

3.2.5 Determining Metric Thresholds

The threshold values of the metric that we store in the nodes of the hierarchy determine the levels of detail to be displayed under a viewing condition. The threshold values are to be measured empirically. Experiments are conducted so that different types of objects can be displayed in various viewing conditions. Users try to determine at which point coarser levels of description can be turned on as the objects are translated away from the viewer. The system will then compute the metric values at these thresholds. A collection of these data can be generalized and served as metric thresholds.

The generic metric thresholds defined through the above process may perform poorly for some objects. For example, an object with many features may need a more strict threshold. On the other hand, a geometrically simple object can apply a loose threshold without much loss of the quality of the image. We will attempt to compensate these poorly defined metric thresholds by allowing users to modify the thresholds for some particular objects. This can be implemented as a simple editing facility. Thus specified values will then serve as the metric thresholds for those objects.

In our current research, we do not tackle the problem of finding the best metric or the best method of determining the metric thresholds. More details about computing the metric values will be explained in chapter 5 on display mechanism.

3.3 Segmentation Process

This is a process to divide up the input object into meaningful parts. Gupta performed residual analysis to create clusters and initiated the fitting procedure for parts using superquadrics models [Gupta 91]. The intermediate models from this fitting process may be used to expand the structural hierarchy for the object. The volumetric segmentation

is an active research area in computer vision.

3.4 Polygonization of the Superquadric and Blobby Models

We render the superquadric or blobby model in the intermediate geometric approximation by polygonizing it and displaying the approximated polyheron. A superquadric model can be formulated as a parametric surface of two angular parameters. Thus an adaptive polygonization can be devised by starting at evenly spaced parameter values and recursively refining the parameters when the tangents at the two adjacent parameter values differ significantly. We can not apply this simple method to a blobby model because it can not be formulated this way and its shape often exhibits concavity. Note that concavity in the superquadric model occurs only when the global deformation is applied which is accounted for by the parametric formulation. Thus, for blobby models we are going to use a polygonization algorithm developed for implicitly defined surfaces. The polygonization algorithm and other rendering options are further discussed in chapter 4 when we give more specific explanation on each model.

One interesting observation is that we can link the polygonization process with the metric threshold value. That is, we know that which range of the resolution the object will need on the screen. We can compute the inter-pixel distance on the object and use it when sampling the surface points.

Chapter 4

Intermediate Models

In this chapter, we describe the models and their recovery procedures we have chosen for intermediate geometric approximations in more detail. In section 1, we introduce superquadrics. Section 2 describes a recovery procedure of superquadrics with global deformations for a single-part object as given by Solina [Solina 87]. Section 3 explains the blobby model defined by Blinn [Blinn 82] and section 4 explains a blobby model recovery procedure given by Muraki [Muraki 91]. In section 5, we propose several improvements over Muraki's algorithm. Finally, we discuss rendering of the blobby models in section 6.

4.1 Superquadrics

Barr provided an explanation of geometrical meaning of the superquadric surfaces by using a spherical product of two two-dimensional curves [Barr 81]. Given two two-dimensional curves,

$$\underline{h}(\omega) = \begin{bmatrix} h_1(\omega) \\ h_2(\omega) \end{bmatrix} \omega_0 \leq \omega \leq \omega_1$$

and

$$\underline{m}(\eta) = \begin{bmatrix} m_1(\eta) \\ m_2(\eta) \end{bmatrix}, \eta_0 \leq \eta \leq \eta_1,$$

the spherical product $\underline{x} = \underline{m} \otimes \underline{h}$ of the two curves is a surface defined by

$$\underline{x}(\eta,\omega) = \begin{bmatrix} m_1(\eta)h_1(\omega) \\ m_1(\eta)h_2(\omega) \\ m_2(\eta) \end{bmatrix}, \begin{matrix} \omega_0 \leq \omega \leq \omega_1 \\ \eta_0 \leq \eta \leq \eta_1 \end{matrix}$$

Geometrically, $\underline{h}(\omega)$ is a horizontal curve vertically modulated by $\underline{m}(\eta)$. η is a north-south parameter, like latitude, whereas ω is an east-west parameter, like longitude. For example, if $\underline{m}(\eta)$ is a half circle and $\underline{h}(\omega)$ is a hull circle, then the spherical product of these two produces a unit sphere:

$$\underline{m} \otimes \underline{h} = \begin{bmatrix} cos\eta \\ sin\eta \end{bmatrix} \otimes \begin{bmatrix} cos\omega \\ sin\omega \end{bmatrix} = \begin{bmatrix} cos\eta cos\omega \\ cos\eta sin\omega \\ sin\eta \end{bmatrix}, \begin{matrix} -\pi/2 \leq \eta \leq \pi/2 \\ -\pi \leq \omega \leq \pi \end{matrix}$$

In two dimensions the sine-cosine curve

$$\begin{matrix} x = acos^{\epsilon}\theta \\ y = bsin^{\epsilon}\theta \end{matrix} \quad -\pi \leq \theta < \pi$$

or

$$(\frac{x}{a})^{\frac{2}{\epsilon}} + (\frac{y}{b})^{\frac{2}{\epsilon}} = 1$$

is called a superellipse.

The spherical product of pairs of these types of curves produces a representation for the superquadrics. The two exponents are squareness parameters. They are used to pinch, round, and square off portions of the solid shapes and to soften the sharpness of square (figure 4.1). The shape progresses square, round, bevel, and pinch as the exponents increase from 0 to 3:

$\epsilon < 1$: shape is somewhat square.

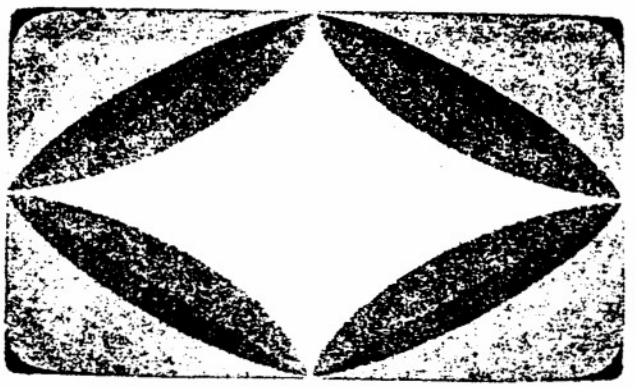


Figure 4.1: The two-dimensional basis functions of superellipses

$\epsilon \approx 1$: shape is round.

$\epsilon \approx 2$ shape has a flat bevel.

$\epsilon > 2$ shape is pinched.

Position vector of superellipsoid is given:

$$\underline{x}(\eta,\omega) = \begin{bmatrix} cos^{\epsilon_1}(\eta) \\ a_3 sin^{\epsilon_1}(\eta) \end{bmatrix} \otimes \begin{bmatrix} a_1 cos^{\epsilon_2}(\omega) \\ a_2 sin^{\epsilon_2}(\omega) \end{bmatrix}$$

$$= \begin{bmatrix} a_1 cos^{\epsilon_1}(\eta) cos^{\epsilon_2}(\omega) \\ a_2 cos^{\epsilon_1}(\eta) sin^{\epsilon_2}(\omega) \\ a_3 sin^{\epsilon_1}(\eta) \end{bmatrix}, \quad \begin{matrix} -\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2} \\ -\pi \leq \omega \leq \pi \end{matrix} . \tag{4.1}$$

Parameter a_1, a_2, and a_3 define the superquadric size in x, y, and z direction respectively in object centered coordinate system. ϵ_1 is the squareness parameter in the north-south direction; ϵ_2 is the squareness parameter in the east-west direction (figure 4.2). Cuboids are produced when both ϵ_1 and ϵ_2 are < 1. Cylindroids are produced when $\epsilon_1 \approx 1$ and $\epsilon_2 < 1$. Pinched shapes are produced when either ϵ_1 or $\epsilon_2 > 2$.

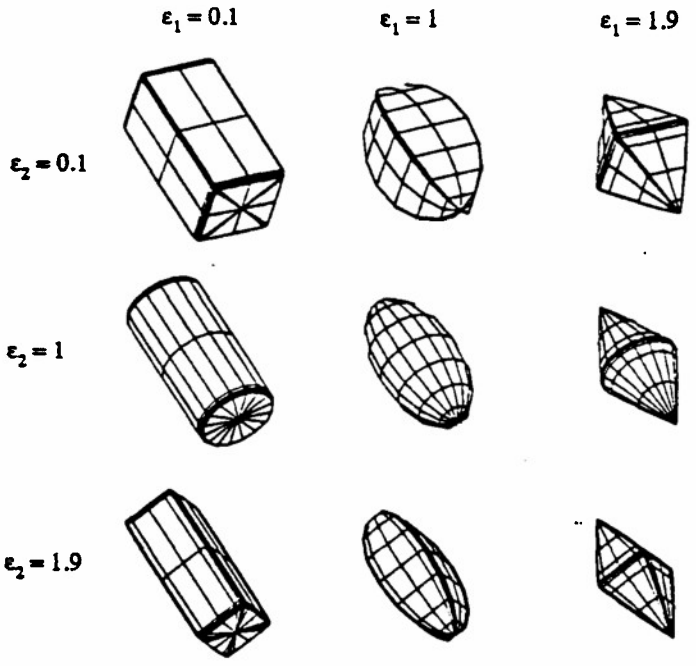


Figure 4.2: Superquadric shapes: $0.1 \leq \epsilon_1, \epsilon_2 \leq 1.0$

Flat-beveled shaped are produced when either ϵ_1 or $\epsilon_2 = 2$. If both ϵ_1 and ϵ_2 are 1, the surface defines an ellipsoid.

Superquadric implicit equation is derived from equation 4.1 by eliminating η and ϵ [Solina 87]:

$$\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}} = 1. \tag{4.2}$$

4.2 Single-part Recovery

The inside-outside function for superquadric model is modified to be used for the recovery procedure by adding the exponent ϵ_1 in the equation 4.2 [Solina 87]:

$$F(x,y,z) = \left[\left[\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right]^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}}\right]^{\epsilon_1}.$$

The outmost exponent ϵ_1 was added by Solina to cancel out the effect of low values

of ϵ_1 during the model recovery process. Similar modification was also done in the modal deformations [Sclaroff 91]. Gupta showed that the modified inside-outside function F for a point (x, y, z) is the square of the factor by which the superquadric has to be scaled to make it pass through the point (x, y, z) [Gupta 91]. That is, this factor is the amount a superquadric has to be expanded or contracted to make it pass through an arbitrary point.

Solina has formulated the optimization function (goodness of fit) in general position and orientation for deformed superquadric model:

$$GOF = \sqrt{a_1 a_2 a_3}\left(\frac{1}{N}\sum_{i=1}^{N}(F(x_i, y_i, z_i) - 1)^2\right)$$

The condition that a point should satisfy the inside-outside functions provides the constraint for a point to lie on the superquadric model. The $\sqrt{a_1 a_2 a_3}$ factor provides for the smallest volume satisfying the surface constraint.

The global deformations such as tapering and bending [Barr 84] are also accounted for into this optimization formula. These deformations are only performed relative to the z-axis. Thus if the z-axis is determined incorrectly, the deformation can not be recovered. We can improve this situation by determining the z-axis as the axis which keeps the symmetry of the object for the tapering and bending deformations.

The optimization is done by an iterative non-linear minimization method. Levenberg-Marquardt method was used in the recovery procedure. This method is based on Newton method and incorporates a gradient method [Reklaitis 83]. By combining the two methods, Levenberg-Marquardt method tries to improve the convergence rate even if the given initial values are not close enough to the solutions.

4.3 Blobby Model

Blobby model expresses a 3D object in terms of the isosurface of a scalar field which is generated from many primitives. Blobby model was introduced by Blinn for displaying

molecular models [Blinn 82]. The field value $V_i(x, y, z)$ at any point (x, y, z) generated by a primitive at a point (x_i, y_i, z_i) is given as follows:

$$V_i(x, y, z) = b_i exp\{-a_i f_i(x, y, z)\}. \tag{4.3}$$

The function $f_i(x, y, z)$ defines the shape of the scalar field. In the case of a spherical field, f_i is given as

$$f_i(x, y, z) = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2,$$

and in the case of a superquadric shaped field, it is given as

$$f_i(x, y, z) = \{(x - x_i)^{\frac{2}{\epsilon_{1_i}}} + (y - y_i)^{\frac{2}{\epsilon_{1_i}}}\}^{\frac{\epsilon_{1_i}}{\epsilon_{2_i}}} + (z - z_i)^{\frac{2}{\epsilon_{2_i}}}.$$

In the case of spherical shape, the exponential term in equation 4.3 brings a Gaussian bump centered at (x_i, y_i, z_i) with height b_i and variance $\frac{1}{a_i}$. When there are several primitives at once, the scalar field from each primitive is added and the resulting isosurface can show a very complicated shape. The field from N primitives at any point (x, y, z) is expressed as

$$V(x, y, z) = \sum_{i=1}^{N} b_i exp\{-a_i f_i(x, y, z)\}. \tag{4.4}$$

The isosurface of value $T(> 0)$ is defined as an implicit function:

$$V(x, y, z) = T.$$

4.4 Blobby Model Fitting

E_{value} constitutes the surface constraint and is given as:

$$E_{value} = \sum_{j=1}^{M} \{V(x_j, y_j, z_j) - T\}^2. \tag{4.5}$$

In order to ensure that the blobby model is fitted into the correct inside or outside direction, the surface normals of the range data are considered. Thus E_{normal} is defined as:

$$E_{normal} = \sum_{j=1}^{M} \left| \mathbf{n_j} - \frac{\mathbf{N(x_j, y_j, z_j)}}{\left|\mathbf{N(x_j, y_j, z_j)}\right|} \right|^2 . \tag{4.6}$$

Here $N(x, y, z)$ is the normal vector of the blobby model and is defined so that it coincides with the negative direction of the gradient of the scalar field which is represented as:

$$N(x, y, z) = -\nabla V(x, y, z),$$

where ∇ is the vector operator,

$$\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}).$$

The unit surface normal $\mathbf{n_j}$ of a data point j is calculated by averaging out the face normals adjacent to the data point.

In the case of a flat surface, any primitive with the values of $a_i = 0$ and $b_i = T$ satisfies both equations 4.5 and 4.6. However, there is no constraint on the shape forming in the area where there is no range data. Therefore, there is a possibility that the primitive which fits to the range data makes strange shapes away from the vicinity of the range data points. To avoid these problems, a new constraint is added which minimizes the influence of the field of each primitive. The equation of the field of a primitive over 3D space is expressed as:

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} V_i(x, y, z) dx dy dz = \left(\frac{\pi}{a_i}\right)^{\frac{3}{2}} b_i.$$

Considering the case where b_i is negative, the shrinking constraint is defined as follows:

$$E_{shrink} = (\sum_{i=1}^{N} a_i^{-\frac{3}{2}} |b_i|)^2. \tag{4.7}$$

By combining equations 4.5, 4.6, and 4.7, we obtain a minimization function for fitting a blobby model to range data as follows [Muraki 91]:

$$E = \frac{1}{M}(E_{value} + \alpha E_{normal}) + \beta E_{shrink}.$$

Here α and β are weighing parameters which control the strength of the surface normal constraint and the shrink constraint.

Muraki's algorithm makes an initial fit between a primitive and the range data, and then divide the primitive into two primitives so as to increase the goodness of fit. Continuing this division for all primitives, the detailed surface of the object can be expressed by the isosurface, which is generated by the primitives. To find a primitive which will strongly influences the result of the minimization, they examine all of the primitives and determine how much the energy value is reduced by the division of the primitive, and then adopt the division which reduces the energy value the most. The selected primitive is divided into two primitives while holding fixed the parameters of the other primitives in the list.

Blobby model recovery can be applied to hierarchical analysis of an object by considering the way a set of primitives branch out from a parent primitive. It is our observation that blobby model recovery can be used as a tool for volumetric segmentation by analyzing the proximity and sizes of the recovered blobs. Object reconstruction can be achieved up to minute detail by fitting a blobby model to range data although it was shown to be a very computationally expensive process.

4.5 Improvements over Muraki's blobby model recovery

The procedure for blobby model fitting given in [Muraki 91] can be improved substantially by adopting the following enhancements:

- We can introduce residual analysis to incorporate the fitting result from the previous step in the selection process of the dividing primitive.
- We can create multiple primitives rather than two primitives originating from the selected primitive. The initialization of the new multiple primitives can be done by analyzing *clusters*.

- Muraki's method recovers only the parameters of two newly bifurcated primitives with other primitive parameters fixed. Instead, we can recover all the parameters of all the primitives involved.

- We can extend the shape of blobby model to superquadrics. The improved blobby model will be able to represent flat surfaces efficiently as well.

In the following subsections, we describe each item in more detail.

4.5.1 Selecting A Splitting Primitive

Muraki uses an exhaustive search to select a blob which will be divided into two blobs. That is, the algorithm examines all the primitives in the list and computes how much the energy value is reduced by the division of each primitive. Then the primitive which reduces the energy value the most is adopted for the division at the next step. With large numbers of primitives, this results in a great deal of waste in computation because we will choose only one primitive for division and all the computation done on other primitives will not be utilized at all.

The selection process can be expedited by incorporating the fitting result from the previous stage. In order to pursue this, we pick the primitive which affects many numbers of poorly fitted data points from the previous fitting step for the next splitting primitive.

In our approach, each primitive maintains a list of data points which are affected by the primitive the most among all the recovered primitives. Also, each data point keeps the residual value which bears the information on how well the data point is fitted to the recovered blobby model.

Let's define the *energy value* for a data point (x, y, z) from a primitive P_i as in equation 4.3: $V_i(x, y, z) = b_i exp\{-a_i f_i(x, y, z)\}$. Each primitive P keeps a list of data points whose energy value from P is the biggest. That is, the data points belonging to

the list of a primitive are those who have the biggest energy value from the primitive among all the primitives recovered. A *residual value* for a data point (x, y, z) is defined as:

$$R(x, y, z) = |V(x, y, z) - T|$$

where $V(x, y, z)$ is given as equation 4.4: $V(x, y, z) = \sum_{i=1}^{N} b_i exp\{-a_i f_i(x, y, z)\}$.

The *total residual value* for a primitive is defined as the sum of the residual values of the data points in the list of the primitive. Then, we select the primitive with the greatest total residual value for the next dividing primitive. Here, note that we compare the energy values of a data point to find out the primitive which influences the data point the most. For the indicator of the fitting for each data point, however, we use the optimization function value which is called a residual value.

4.5.2 Splitting A Primitive Into Multiple Primitives

We can further speed up the recovery process by dividing the selected primitive into multiple primitives rather than only two primitives. The number of new primitives and the initial values of their parameters are determined by using some heuristics which are obtained by performing residual analysis on the result from the previous fitting.

We mentioned in the above section that each primitive maintains a list of data points whose energy values are the largest from that primitive. We can create clusters from the data points in the list of the selected primitive. This process is similar to *clustering* in [Gupta 91]. That is, each data point in the list can be classified into three classes: overestimated, exact, and underestimated. Each maximal set of overestimated data points which are connected forms a cluster. Also, we create another cluster by collecting all the exact and underestimated data points. Each cluster with data points more than a given threshold is considered non-trivial.

Then the number of the new non-trivial primitives originating from the divided primitive is decided as the number of new primitives. The initial values of the parameters

for each primitive are obtained from the center, radius, and variance of each cluster.

4.5.3 Simultaneous Fitting

Another problem with Muraki's algorithm is in recovering the parameters of only the two new primitives from the division while holding the parameters of other primitives fixed. Instead we can recover the parameters of all the primitives at the same time.

For example, let's consider an object which consists of three spheres. After recovery of two primitives, one of them is chosen to be divided. If we use Muraki's bifurcation method for the fitting, the result of fitting will be still unsatisfactory because the parameters of one primitive are fixed through the fitting process (figure 4.3). Therefore, the fitting will continue until the number of primitives grows huge in order to obtain sufficiently good fit. On the other hand, with the simultaneous fitting, only three primitives are needed to get a very good fit to the object. Therefore, the simultaneous fitting enables efficient fitting to some objects which may involve great overhead in the bifurcation method.

However, recovering all the parameters may create a large linear equations system when the number of the primitives is large. When the order of matrix is large, the commonly used Gaussian method tends to show the round off errors. Therefore, we need to adopt an iterative method for solving the large linear equations system in the optimization process.

4.5.4 Using Superquadric Shaped Blobby Model

The formulation of the optimization problem for blobby model fitting can be extended to superquadric shaped blobby models. Superquadric blobby model will increase the shape vocabulary by far. Especially, flat surfaces can be represented by the blobby model [Bloomenthal 91].

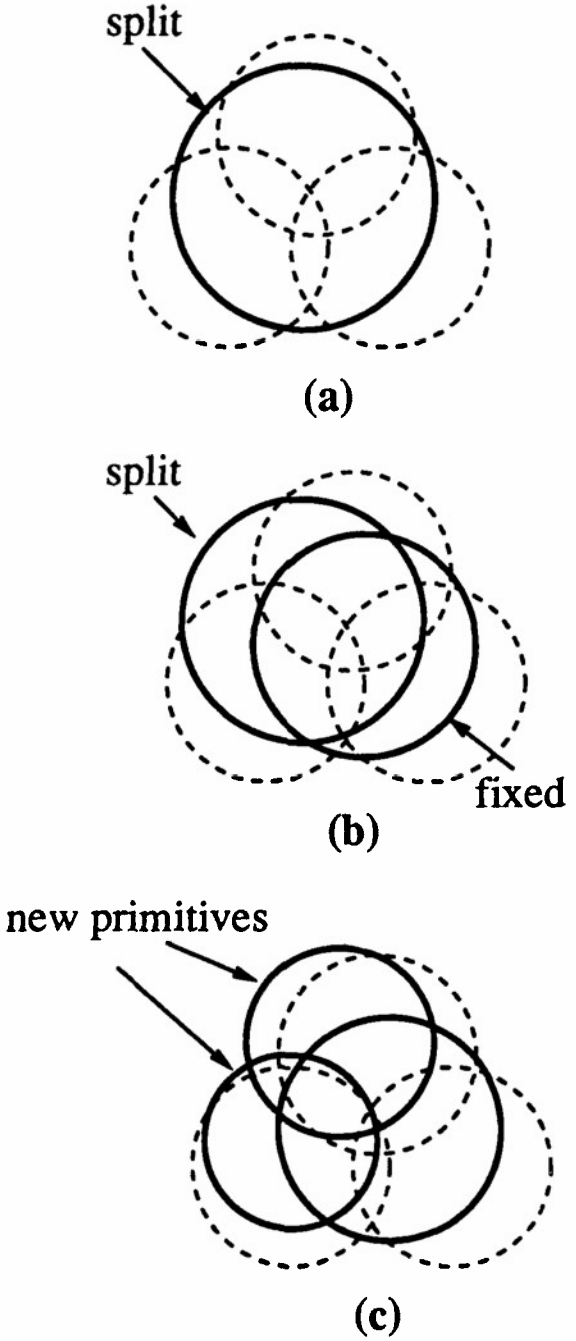


Figure 4.3: An example which shows the limitation of the bifurcation method (a) initial fitting with one primitive (b) one primitive splits into two (c) one primitive splits again while the other stays fixed, resulting in poor overall fitting even with three primitives

4.5.5 A new fitting algorithm

A blobby model fitting algorithm incorporating all the suggested improvements is given as follows:

1. Fit a blob to the range data by solving the minimization problem.
2. Repeat the following steps until the residual is small enough.
3. Select a primitive P_i which has the greatest total residual value.
4. Perform clustering of the data points listed in P_i.
5. Split P_i into multiple primitives. Delete P_i from primitive list and add all new primitives into the list.
6. Fit all the primitives in the list to the range data by solving the minimization problem.
7. Insert each data point in the divided primitive into the list of one of the newly created primitives by comparing the energy values. Then compute the total residual values for each primitive.

Note that this last step can be done with little computation because the residual values and the energy values of all the data points were already computed during the fitting procedure. Also we assume that each primitive can still retain the data points in its list even if its shape changed during the new fitting process. Likewise, for the data points from the selected primitive for division, we only compare the energy values from the newly divided primitives to find the primitive with the biggest influence on the data points. This will avoid excessive comparison among all the primitives for each data point in the list of the divided primitive.

4.6 Rendering Blobby Models

There are three common approaches in rendering implicit functions. One method is to polygonize the implicit function [Bloomenthal 88, Hall 90, Wyvill 86] and use the conventional rendering of polyhedrons. Another rendering technique is to use a method based on ray tracing [Blinn 82, Kalra 89, Hanrahan 83]. The third method is a scan line method for shaded display of polynomial functions [Sederberg 89].

The ray tracing and polygonization method are capable of displaying arbitrary implicit surface, but do not operate in real or near real time on today's workstations [Bloomenthal 90]. Since our aim is to provide a geometric environment representation for efficient real-time rendering and manipulation, we can not rely on ray-tracing based mechanisms.

For our interactive applications, we need to polygonize the blobby models and store them as polyhedrons. Besides, modern hardware provides built-in procedures for displaying polygons rapidly. On the other hand, storing the intermediate geometry approximations using the parameters is a compact and efficient storage method for ray tracing if need for ray tracing arises for better picture quality.

4.6.1 Survey of Polygonization Techniques for Implicit Surfaces

Hall and Warren presented a method for finding an adaptive polygonal approximation of piecewise implicit functions [Hall 90]. The polygonalization can then be rendered using standard shaded polygon drawing techniques. The class of implicit functions they tackled was the contours of trivariate polynomial functions represented in Bernstein/Bezier form. Their approach ensures to yield an approximation guaranteed accurate to within some user-specified tolerance of the actual surface for the algebraic surfaces.

The method of Wyvill, McPheeters, and Wyvill involves creating an array of cubes and evaluating the function at each corner point. For each cube that exhibits a sign

change between vertices, the method constructs a linear approximation to the surface [Wyvill 86]. One major drawback of their method is that their algorithm may sample heavily in areas in which the function is nearly linear and therefore relatively uninteresting. That is, their algorithm developed for soft object data structure does not propose an adaptive algorithm.

Bloomenthal extended this method to do adaptive subdivision according to surface curvature by using an octree space partitioning technique [Bloomenthal 88]. The implicit function is adaptively sampled as it is surrounded by a spatial partitioning. The partitioning is represented by an octree and a peicewise polygonal representation is derived from the octree. Criteria for subdivision of an octree node were based on object characteristics such as tangency and curvature.

4.6.2 Our Polygonization Algorithm

The number of polygons generated in the polygonization is critical in our problem. This cause us to choose an adaptive algorithm.

We use Barr's straddling box method to generate spatial partitions which divides space into convex polyhedral cells, whose union surrounds the surface. This guarantees that no part of the surface is missing from the spatial partitioning. Note that Wyvill's or Bloomenthal's methods may result in holes by ignoring some fine features. We perform adaptive subdivision by subdividing a cell when it generates more than one polygon or when the tangents at the corner points differ significantly.

A crack may occur along the shared face of two cells if one cell is subdivided and the other is not as pointed out in [Bloomenthal 88]. We solve this problem by choosing the edges generated from the subdivided cell for the polygons from both of the cells. We can also try to combine any two adjacent faces if their tangents are close enough by merging their enclosing boxes and computing a new polygon. The surface tangent at the center of the polygon is used.

Chapter 5

Display Mechanism

5.1 Overview

In this chapter, we are going to review the display mechanism that determines the level of details to be displayed for the objects in the environment under the current viewing condition. This involves traversing the hierarchy and selecting the appropriate level to display based on a metric. We will first discuss the metric to be used and then the algorithms for traversing the hierarchy.

5.2 Selecting a Metric for Adaptive Details

The level of details to be displayed for an object is determined based on a metric. Generally speaking, the metric is a function of the area which the object occupies on the screen in terms of the number of pixels and the distance of the object from the viewer.

The term *perspective effects* has long been used by artists to mean the way the appearances of things change with increasing distance from the viewer. It meant both the loss of spatial detail seen in objects, and the color changes over longer distances. The metric is primarily dependent on distance and it measures perspective effects in the

broad sense defined above. The primary cause of loss of detail is the *inverse square law* which describes the relative diminution of areas with distance. A secondary cause of loss of detail is *atmospheric perspective* which results because light traveling through the air is scattered and absorbed. Under normal conditions the geometric perspective effect predominates in the medium range of distances. The longer range is always dominated by atmospheric effects.

The ideal metric should measure the importance of the object to the viewer and not just its size on the image. This metric will then be dependent on the frequency and color of the object as well as its size and location on the screen.

In our system, we use a relatively simple metric which basically measures the number of pixels the object occupies on the screen under the *current viewing condition*. This is actually computed by using the bounding volume of the object. The bounding volume of an object can be easily obtained by a simple bottom up method which goes through the object hierarchy and records the maximum and minimum values in x, y and z. A bounding cuboid can then be represented by these values. These values are pre-processed and stored in the nodes of hierarchy during the hierarchy construction process as described in Chapter 3.

During the real-time display, the current viewing transformation is applied to the bounding volume of the object and the screen area of the bounding volume (hence the metric value) can then be computed. Recall that we have stored threshold value of the metric in the nodes of the hierarchy during the hierarchy construction process (Chapter 3). The current metric value is then compared to these threshold values and determine the level of details to be displayed. This procedure is explained in more detail in the following sections.

Other types of bounding volumes were suggested for efficient ray object intersection test. Of particular interests are bounding volumes developed by Weghorst, Hooper, and Greenberg [Weghorst 84]. In order to determine the optimal bounding volumes, they examined the cost of intersecting the volume, the item complexity, and the project void

areas. They determine a bounding volume from three candidates, namely, a sphere, a rectangular parallelepiped, and a cylinder. Considering the project void area is especially interesting to us. Kay and Kajiya also come up with a new type of bounding volume which can be made to fit the convex hull of an object arbitrarily tightly in exchange for the complexity of representation [Kay 86].

Another possible component to be considered in the metric is the goodness-of-fit of the intermediate geometric representation to the original data. The rationale behind this is that if the object occupies a large screen area and the goodness-of-fit for the current node is poor, it may suggest us to go down another level to use a better representation. However, the effectiveness of this metric is not fully obvious yet and awaits further experimentation.

The display system may also allow viewers to specify the importance of an object to them. This user specified priority metric of an object will overwrite the system defined default metric threshold. The system will display an object with high priority metric in more detail and an object with low priority metric in less detail. This kind of facility will be useful when a user is manipulating a very complex object (or environment) but is interested in only a relatively small part of the object (or environment). The priority metric will shorten the response time of the system and provide a better interactive feeling and control to the viewer.

The metric we have adopted can be categorized as a *spatial metric* [Blake 90]. There is another type of metric called *dynamic metric* which concerns with moving objects and measures the speed with which the projected image moves. Since our work focuses mainly on static objects, this type of metric is not considered here.

5.3 Intermediate Geometric Approximations in the Hierarchy

The intermediate geometric approximations stored in the nodes of the hierarchy are essentially parametric equations of either superquadrics or blobby models. We can save these intermediate geometric approximations by storing only the parameter values and leave the polygonization for the real time display process. Or we can precompute the polygonization and store the polyhedrals into the data file explicitly.

The former method will result in considerable reduction in disk storage but require running the polygonization module to retrieve the surface shape in real time display. On the other hand, storing the polygons explicitly will require more disk space but it will be faster in display. The decision between these two schemes can be done after careful consideration of this tradeoff including the speed of the polygonization algorithm and the storage efficiency of the resulting polyherons. For better real time interactive performance, we opt for the latter option to store all the polygons into the data file, which can be compressed to minimize the storage requirement.

5.4 Traversal of the Hierarchy for Real Time Display

5.4.1 Initial Display of the Environment

The procedure for displaying a geometric environment is as follows:

1. Set $NODE$ to the root of hierarchy.

2. Apply the viewing transformation to the segment corresponding to $NODE$. Compute the area it occupies on the screen. We use the bounding cuboid of the object in the computation since computing with the original segment data is expensive.

3. Recall that each internal node contains the metric value in terms of threshold area or distance. Compare the size of the area with the threshold metric value stored in the node. If the area is smaller than the threshold area, this means that the current level of detail is good enough for the object under the current viewing condition.

4. Otherwise, set $NODE$ to *each* child of $NODE$. Apply the previous procedure to each new $NODE$. In effect, we are traversing the hierarchy in a recursive descent way and utilizing more complex intermediate geometric approximation of the object. Clipping can also be done during the tree traversal.

An example showing how the algorithm works is shown in figure 5.1.

5.4.2 Changing Viewing Conditions

Whenever the viewpoint is changed, e.g. zooming, or any object translates or rotates in the environment, the viewing condition is changed and we need to recompute the level of details to be displayed for the objects involved in the operation. Based on the *frame coherence* property which means that the objects in the scene are not going to change drastically in display between two consecutive frames, we do not need to restart the computation from the root of the hierarchy. Instead, we can start at the current level of detail and move up or down the hierarchy as necessary.

At any time in display, we maintain a list of nodes of the hierarchy which constitute the current levels of detail for all the objects in the environment. Let's call the nodes used for display under the current viewing condition as *appearance nodes* and name the list of appearance nodes as *appearance list*. The appearance list is actually implemented as an array of lists where each array member corresponds to each level of the hierarchy tree and each list enumerates all the appearance nodes at that level.

The algorithm for deciding the levels of detail for the next frame maintains other data structures as well. The *active list* is initialized to appearance list and keeps track

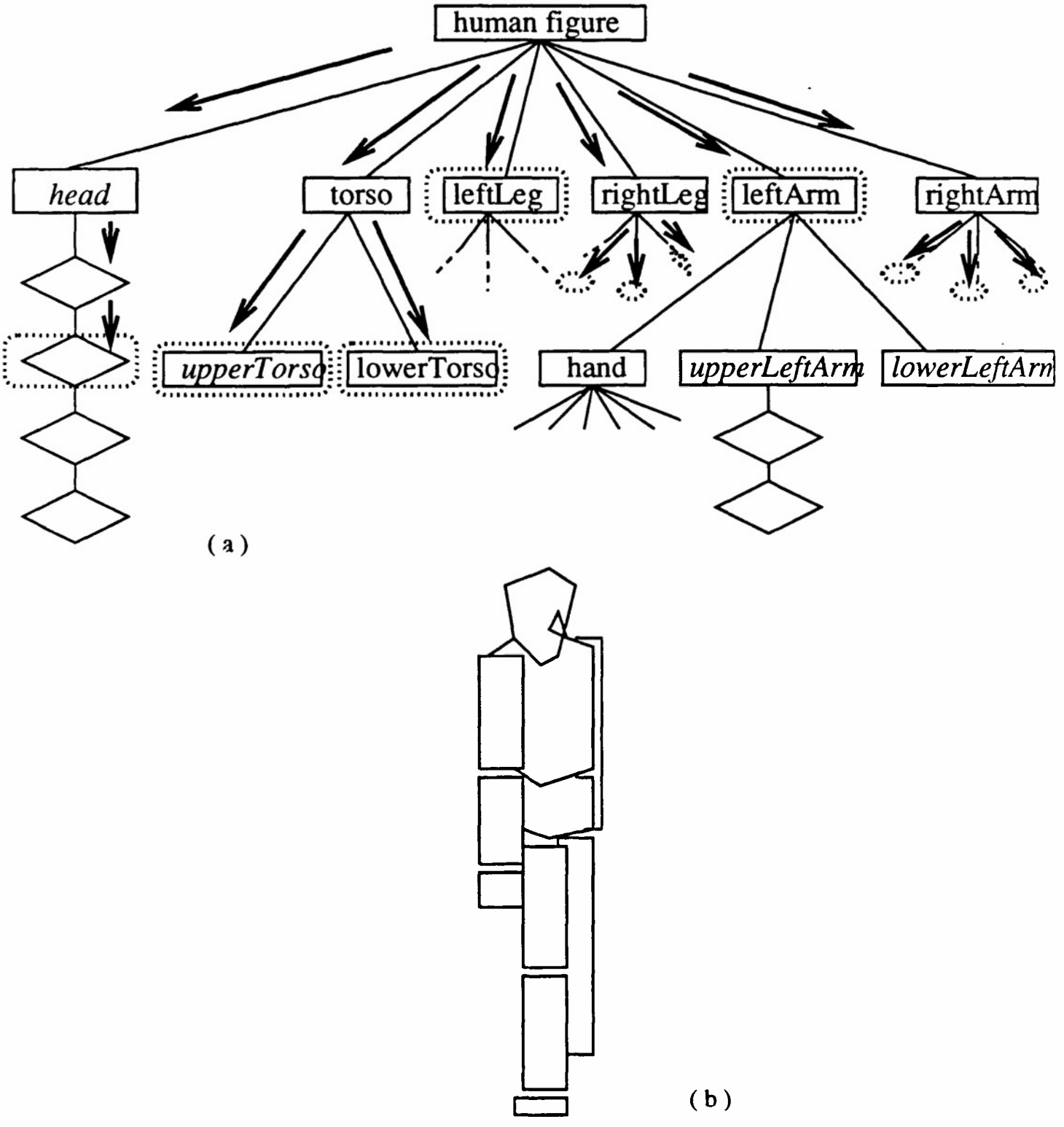


Figure 5.1: an example demonstrating how an object such as a human figure is displayed: (a) the traversal of the hierarchy (b) the human figure displayed with intermediate details

of all the nodes which need further processing. The active list is also implemented as an array of lists like appearance list which contains each list for each depth level.

We start the computation from the lowest level of the hierarchy tree. Thus, we compute the metric for nodes which are in the lowest level of the tree from the active list. If the metric area is larger than threshold metric value then we have to descend to its children. So we add all its children to a new list, *display candidate list* or *candidate list*. The display candidate list maintains all nodes which needs computation to decide whether it is in the appropriate level of detail or it will call for further descending to their children. But any node in the display candidate list assures that it needs at least that level of detail, thus it will not ascend the tree.

If the metric value is smaller than the threshold value, then we have to decide whether we can display its parent instead of the current node. This process is not simple because one of the siblings may need further detail. Therefore we employ an *request/acknowledge* scheme. We add the *request* from the current node to its parent. Then the first step for each node in the active list is to test whether it has *requests* from all its children. If not, it simply adds all the children who sent their *requests* to the *display candidate* list. If it has all the *requests*, then it computes its own metric and decides which level is appropriate for display.

An algorithm which achieves this procedure is given in the following:

1. Initialize *Active* list to *Appearance* list. Set N to the biggest depth of nodes in *Active* list.

2. For each node in the *Active* list, send *virtual request* from all its children. This is done as part of the initialization.

3. Repeat the following until N is 0 or the *active* list is empty

   (a) Set $NODE$ to an element of the *active* list in depth N and delete $NODE$ from the list.

(b) Test whether $NODE$ has received *requests* from all its children. If not, we ought to display its children instead of $NODE$. Add the children of $NODE$ who sent *request* to *appearance* list.

We insert them directly to appearance list because all the necessary tests have been done already in the previous iterations and we know that they sent request to $NODE$ because they wanted coarser detail rather than more detail. We are done the processing of $NODE$ in case that $NODE$ does not have all requests and we can proceed to next iteration with another $NODE$ or a new value of N.

(c) Compute the metric value for $NODE$ using its bounding cuboid.

(d) Compare the size of the area with the threshold metric value store in the $NODE$. If the area is larger then the threshold area, this means that the display area is too big for the object to be display with this level of detail and we have to use its children to display. At this stage, we know that all its children sent a request. There are two types of requests: *virtual request* or *request*. Note that virtual requests are raised only as part of initialization and in this case all the children of $NODE$ have virtual requests.

i. *virtual request:* Add each child of $NODE$ into *candidate* list if the child is an internal node or *appearance* list if a leaf node.

ii. *request:* Add all the children of $NODE$ who sent request to $NODE$ to *appearance* list. Here again, the child nodes who sent request are those who do not need more detail. So we add them directly into appearance list.

(e) Otherwise, we ought to display $NODE$ or its parent. Add the parent node of $NODE$ to the *active* list and issue *request* from $NODE$ to the newly added parent node. This newly added node will be taken care of at the next iteration of this process with a decreased value of depth N.

(f) If the *active* list at depth N is empty, set $N = N - 1$.

4. Process each element in the *candidate* list to get an appropriate level of detail by descending the hierarchy as follows:

   (a) Set $NODE$ to an element in the *candidate* list and delete it from the *candidate* list.

   (b) Compute the metric value for $NODE$ using its bounding cuboid.

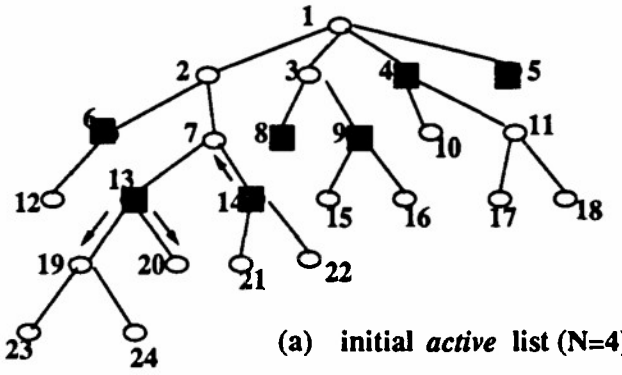
   (c) If the metric area is smaller then the threshold area, then add $NODE$ to the *appearance* list. That is, $NODE$ is the level of detail which will be rendered on display. this means that the display area is too big for the object to be display with this level of detail and we have to use its children to display. Add the children of $NODE$ who sent *request* to *candidate* list.

   (d) Otherwise, add all the children of $NODE$ into *candidate* list.

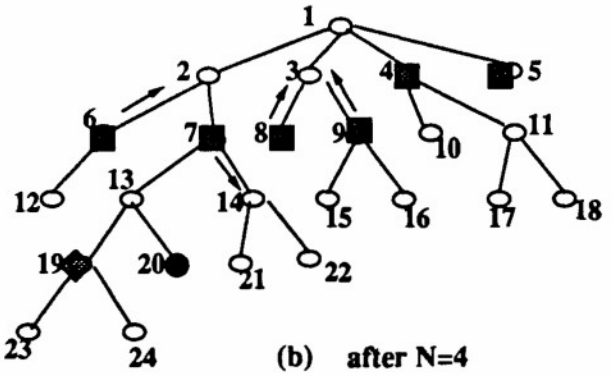
   (e) Repeat this process until the *candidate* list is empty.

Let's take an example of a tree whose initial tree configuration is shown in figure 5.2 (a). All the nodes in the *active* list are colored grey. Recall that these nodes are also the *appearance* nodes before the viewing condition changes. Since the maximum depth from the active list is 4, we start processing the active nodes with the depth value of N = 4. Note that the arrow symbols from the active nodes at the level of N indicate either ascending or descending from the node according to the result from the metric comparison.

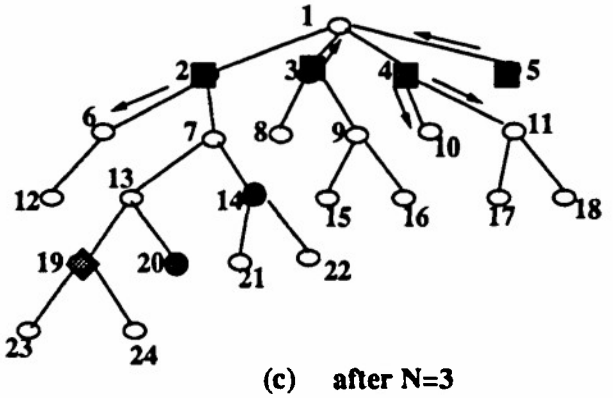
The result after the first iteration of the algorithm is shown in figure 5.2 (b). After processing node 13, node 20 is inserted into appearance list since it is a leaf node while node 19 is inserted into candidate list. Also node 7 becomes active because of the request from node 14. Likewise, the results after processing for different level depth values are presented in figure 5.2 (c), 5.3 (d), and 5.3 (e). In figure 5.2 (c), node 14 becomes an appearance node even if it is an internal node because it raised request to its parent for coarser detail. Figure 5.3 (f) shows all the appearance nodes after processing candidate list.



(a) initial *active* list (N=4)



(b) after N=4



(c) after N=3

Figure 5.2: an example demonstrating the view changing algorithm (a)-(c)

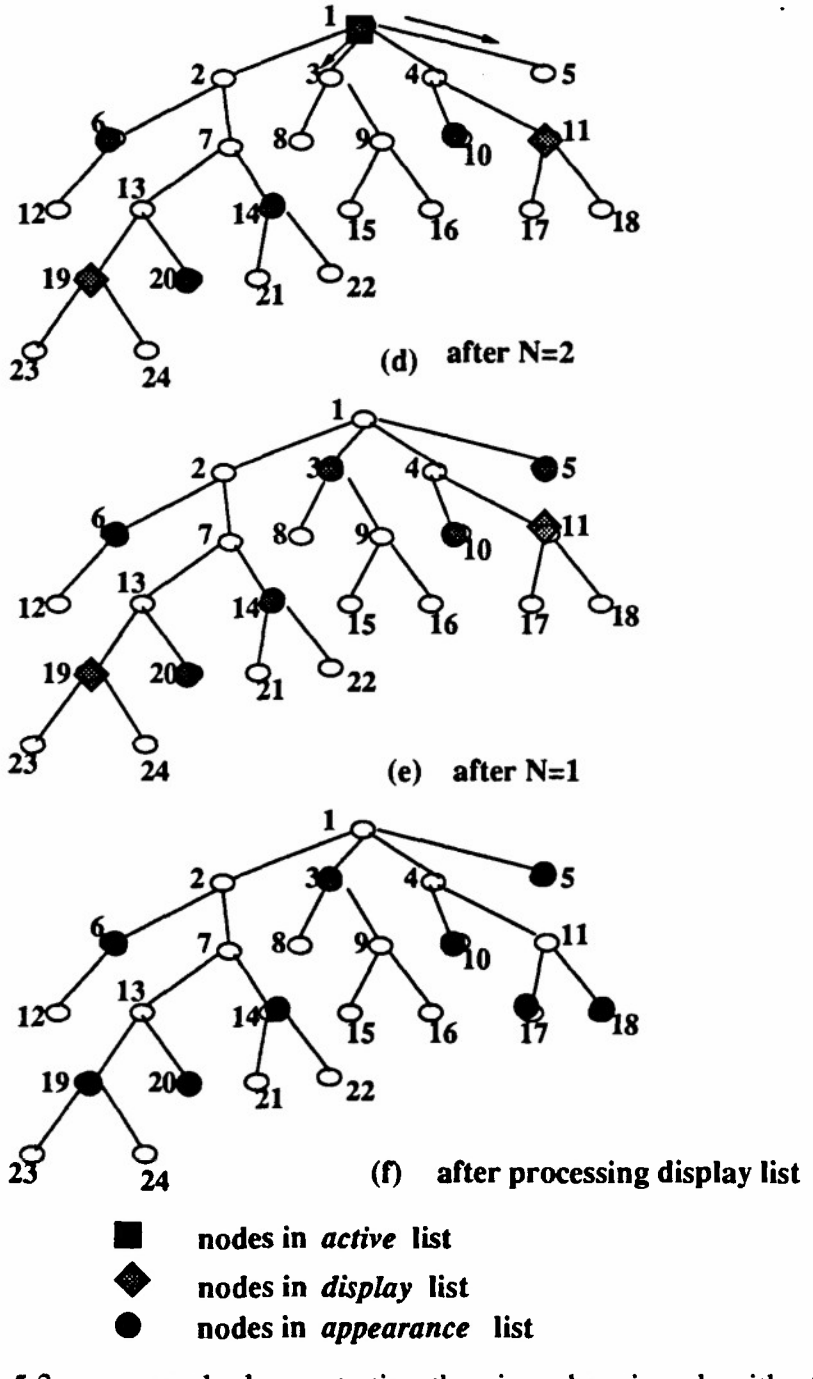


Figure 5.3: an example demonstrating the view changing algorithm(d)-(f)

5.4.3 Discussion on efficiency

The algorithm given in the above is complex and may raise a question to readers whether this will be really more efficient than starting all over from the root. The answer can be yes or no depending on the given hierarchy and viewing condition. Certainly there may be situations where this new algorithm is more complex than the one given in the previous section. However, if the given geometric environment is complex and if the frame coherence is satisfied, then this new algorithm is more efficient because it requires descending or ascending from the current levels of detail only a few steps. Therefore, in general, this new algorithm is more efficient.

5.5 Other Issues

5.5.1 Display Technique

In order to avoid flickering in transition between two levels of detail, we perform fade in / fade out technique by making the old description transparent while the new description is taking its shape. This is in effect interpolating the two descriptions which can be either linear or nonlinear. This ought to reduce the abruptness of change of detail in the object.

5.5.2 Handling Attributes

Color of an object, for instance, can be non-uniform and requires special treatment when the object is transformed to a coarser description. For each face in the coarser description, the color of that face can be assigned to the dominant color of the underlying faces. The dominant color is determined as the one which occupies the most area of the underlying faces.

A more effective solution to the attribute problem is to use the *texture mapping*

technique, which the hardware has managed to handle in real time in advanced graphics machines during recent years. By using the *shrinkwrap* technique [Bier 86] which allows texture mapped to multiple faces from a 2-D rectangular texture pattern, we can impose the texture of the details on the surface of the lower detail description.

5.6 Handling Articulated Figures

As mentioned before, in our current research we only consider static objects and do not attempt to apply our algorithm to deformable or articulated objects. However, our approach lends itself to a feasible extension to handle articulated objects. This process can be helped by recovering the part hierarchy of an object through *volumetric segmentation*. A rough idea for manipulating articulation in the simplified model is to divide the coarser model into several parts which correspond to the *segments* or *links* in the articulated object. Whenever a joint is rotated or translated, we can compute a transformation matrix which will perform the movement. Then we can apply the movement to the corresponding parts by multiplying the transformation matrix to the parts. This matrix multiplication can be done efficiently by hardware.

5.7 Performance Analysis

We plan to do performance analysis on several objects. The analysis will be done on objects with the defined object hierarchy (DOH) such as a human figure model or a house. Objects with no given hierarchy (NOH) but with fairly complex geometry and multiple components such as a tank will also be tested. For each object, we will compare the rendering time of the object under view changes and some static environments in various levels of details.

Chapter 6

Conclusion

6.1 Work in Progress

We have implemented Muraki's blobby model fitting algorithm by using the surface and shrink (volume) constraints and have tested on simple objects. It still needs to include the surface normal constraint to recover complex objects correctly and we are currently working on it. An adaptive polygonization is also being implemented.

We will then work on the enhancement to Muraki's algorithm as described in a previous chapter. We will implement the control structure of the system including the hierarchy construction module. Subsequently we will work on the display traversal routine. The hierarchy tree traversal will be implemented on the top of JACK so that the geometric environment can be displayed in JACK. JACK also provides a convenient tool to build up the user interface for allowing users to specify the priority metric of the objects. Finally, we will run our system on a number of sample objects and analyze the performance gain in different cases.

6.2 Contribution

The principal contributions of this work are as follows:

- We propose a novel scheme to construct the intermediate geometric approximations for complex objects. Different levels of details are obtained automatically in the process and can be used for effective real-time display of complex environment.
- The system will be useful in areas like flight simulation systems or virtual reality environments in which complex geometric environments have to be displayed in real time. Our system automates the hierarchy construction process which will reduce the modeling effort significantly.
- We survey the existing object representation schemes and analyze their merits for our geometric detail reduction application.
- We extend Muraki's blobby model fitting algorithm to superquadrics shaped blobby model. We improved the algorithm by using residual analysis technique and incorporating heuristics in the blob splitting process.
- Based on the existing polygonization methods of implicit surfaces, we develop a new adaptive polygonization algorithm for implicitly defined surfaces.
- We devise a display algorithm utilizing the intermediate geometric approximations. The algorithm makes use of a metric in determining the level of detail to be displayed. In particular, the algorithm explores the frame coherence property and keeps traversal of the hierarchy minimal.

Bibliography

[Amanatides 87] Amanatides, John. Realism in Computer Graphics: A Survey. *IEEE Computer Graphics and Applications* 7(1):44–56, January 1987.

[Badler 84] Badler, Norman I. and I. Carlbom. The Computer Graphics Scene in U.S. *Eurographics '84* 185–200, 1984.

[Bajcsy 90] Bajcsy, Ruzena, Kwangyeon Wohn, Franc Solina, Alok Gupta, Gareth Funka-Lea, Celina Imielinska, Pramath Sinha, and Constantine Tsikos. *Final Report On Advanced Research In Range Image Interpretation For Automated Mail Handling.* Technical Report MS-CIS-90-14, University of Pennsylvania, 1990.

[Barr 81] Barr, Alan H. Superquadrics and Angle-Preserving Transformations. *IEEE Computer Graphics and Applications* 11–23, January 1981.

[Barr 84] Barr, Alan H. Global and Local Deformations of Solid Primitives. *Computer Graphics (Proc. SIGGRAPH)* 18(3):21–30, July 1984.

[Bier 86] Bier, Eric A. and Jr. Kenneth R. Sloan. Two-Part Texture Mappings. *IEEE Computer Graphics and Applications* 6(9):40–53, 1986.

[Blake 90] Blake, Edwin Haupt. *Complexity in Natural Scenes: A Viewer Centered Metric for Computing Adaptive Detail.* PhD thesis, London University, Department of Computer Science, Queen Mary College, Mile End Road, London, E1 4NS, 1990.

[Blinn 82] Blinn, James F. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics* 1(3):235–256, July 1982.

[Bloomenthal 88] Bloomenthal, Jules. Polygonization of implicit surfaces. *Computer Aided Geometric Design* 5:341–355, 1988.

[Bloomenthal 90] Bloomenthal, Jules and Brian Wyvill. Interacitve Techniques for Implicit Modeling. *Computer Graphics (Special issue on 1990 Symposium on Interacitve 3D Graphics)* 24(2):109–116, March 1990.

[Bloomenthal 91] Bloomenthal, Jules and Ken Shoemake. Convolution Surfaces. *Computer Graphics (Proc. SIGGRAPH)* 25(4):251–256, July 1991.

[Blum 78] Blum, H. and R.N. Nagel. Shape Description using weighted symmetric axis features. *Pattern Recognition* 10:167–180, 1978.

[Brunet 90] Brunet, Pere and Isabel Navazo. Solid Representation and Operation Using Extended Octrees. *ACM Transactions on Graphics* 9(2):170–197, April 1990.

[Carlbom 85] Carlbom, Ingrid, Indranil Chakravarty, and David Vanderschel. A Hierarchical Data Structure for Representing the Spatial Decomposition of 3D Objects. *IEEE Computer Graphics and Applications* 24–31, April 1985.

[Cheng 89] Cheng, Fuhua and Ardeshir Goshtasby. A Parallel B-Spline Surface Fitting Algorithm. *ACM Transactions on Graphics* 8(1):41–50, 1989.

[Clark 76] Clark, James H. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM* 19(10):547–554, October 1976.

[Feiner 85] Feiner, Steven. APEX:An Experiment in the Automated Creation of Pictorial Explanations. *IEEE Computer Graphics and Applications* 5(8):29–37, November 1985.

[Foley 90] Foley, James D., Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: principles and practice.* Addison Wesley, 2nd edition, 1990.

[Glassner 84] Glassner, Andrew S. Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics and Applications* 15–22, October 1984.

[Glassner 88] Glassner, Andrew S. Spacetime Ray Tracing for Animation. *IEEE Computer Graphics and Applications* 60–70, March 1988.

[Goldsmith 87] Goldsmith, Jeffrey and John Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications* 14–20, May 1987.

[Gupta 91] Gupta, Alok. *Surface and Volumetric Segmentation of Complex 3-D Objects Using Parametric Shape Models.* PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1991.

[Hall 90] Hall, Mark and Joe Warren. Adaptive Polygonalization of Implicitly Defined Surfaces. *IEEE Computer Graphics and Applications* 10(6):33–42, November 1990.

[Hanrahan 83] Hanrahan, Pat. Ray Tracing Algebraic Surfaces. *Computer Graphics (Proc. SIGGRAPH)* 17(3):83–90, July 1983.

[Kalra 89] Kalra, Devendra and Alan H. Barr. Guaranteed Ray Intersections with Implicit Surfaces. *Computer Graphics (Proc. SIGGRAPH)* 23:297–306, 1989.

[Kay 86] Kay, Timothy L. and James T. Kajiya. Ray Tracing Complex Scenes. *Computer Graphics (Proc. SIGGRAPH)* 269–278, 1986.

[Metaxas 91] Metaxas, Dimitri and Demetri Terzopoulos. Constrained Deformable Superquadrics and Nonrigid Motion Tracking. *CVPR-91, Hawaii* , 1991.

[Muraki 91] Muraki, Shigeru. Volumetric Shape Description of Range Data using "Blobby Model". *Computer Graphics (Proc. SIGGRAPH)* 25:227–235, 1991.

[Nackman 82] Nackman, Lee R. Curvature Relations in Three-Dimensional Symmetric Axes. *Computer Graphics and Image Processing* 20:43–57, 1982.

[ORourke 79] O'Rourke, Joseph and Norman Badler. Decomposition of Three-Dimensional Objects into Spheres. *IEEE Trans. Pattern Analysis and Machine Intelligence* 1(3):295–305, July 1979.

[Payne 92] Payne, Bradley A. and Arthur W. Toga. Distance Field Manipulation of Surface Models. *IEEE Computer Graphics and Applications* 12(1):65–71, 1992.

[Pentland 91] Pentland, Alex and Stan Sclaroff. Closed-Form Solutions for Physically-Based Shape Modeling and Recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence* 13(7):715–729, July 1991.

[Reklaitis 83] Reklaitis, G. V., A. Ravindran, and K. M. Ragsdell. *Engineering Optimization: Methods and Applications.* John Wiley and Sons, 1983.

[Requicha 80] Requicha, A. A. G. Representations for Rigid Solids: Theory, Methods, and Systems. *ACM Computing Surveys* 12(4):437–464, 1980.

[Requicha 83] Requicha, A. A. G. Toward a Theory of Geometric Tolerancing. *International Journal of Robotics Research* 2(4):45–49, Winter 1983.

[Rubin 80] Rubin, Steven M. and Turner Whitted. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. *Computer Graphics (Proc. SIGGRAPH)* 14(3):110–116, August 1980.

[Rubin 82] Rubin, Steven M. The Representation and Display of Scenes with a Wide Range of Detail. *Computer Graphics and Image Processing* 19:291–298, 1982.

[Samet 88a] Samet, Hanan and Robert E. Webber. Hierarchical Data Structures and Algorithms for Computer Graphics, Part I: Fundamentals. *IEEE Computer Graphics and Applications* 48–68, May 1988.

[Samet 88b] Samet, Hanan and Robert E. Webber. Hierarchical Data Structures and Algorithms for Computer Graphics, Part I: Applications. *IEEE Computer Graphics and Applications* 59–75, July 1988.

[Sandor 85] Sandor, John. Octree Data Structures and Perspective Imagery. *Computers & Graphics* 9(4):393–405, July 1985.

[Schmitt 86] Schmitt, Francis J. M., Brian A. Barsky, and Wen-Hui Du. An Adaptive Method for Surface-Fitting from Sampled Data. *Computer Graphics (Proc. SIGGRAPH)* 179–188, 1986.

[Sclaroff 91] Sclaroff, Stan and Alex Pentland. Generalized Implicit Functions For Computer Graphics. *Computer Graphics (Proc. SIGGRAPH)* 25:247–250, 1991.

[Sederberg 89] Sederberg, T.W. and A.K. Zunderl. Scan Line Display of Algebraic Surfaces. *Computer Graphics (Proc. SIGGRAPH)* 23(3):147–156, July 1989.

[SG 91] SG. *Periodic Table of the IRISes.* Silicon Graphics Computer Systems, November 1991.

[Snyder 87] Snyder, John M. and Alan H. Barr. Ray Tracing Complex Models Containing Surface Tessellations. *Computer Graphics (Proc. SIGGRAPH)* 21(4):119–128, July 1987.

[Solina 87] Solina, Franc. *Shape Recovery and Segmentation with Deformable Part Models.* PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1987.

[Terzopoulos 91] Terzopoulos, Demetri and Dimitri Metaxas. Dynamic 3D Models with Local and Global Deformations: Deformable Superequadrics. *IEEE Trans. Pattern Analysis and Machine Intelligence* 13(7):703–714, July 1991.

[Weghorst 84] Weghorst, Hank, Gary Hooper, and Donald P. Greenberg. Improved Computational Methods for Ray Tracing. *ACM Transactions on Graphics* 3(1):52–69, January 1984.

[Wyvill 86] Wyvill, Geoff, Craig McPheeters, and Brian Wyvill. Data Structure for soft objects. *The Visual Computer* 2:227–234, 1986.

[Yan 85] Yan, Johnson K. Advances in computer–generated imagery for flight simulation. *IEEE Computer Graphics and Applications* 5(8):37–51, August 1985.

C Virtual Building Walkthrough Systems

Virtual Building Walkthrough Systems

Paul J. Diefenbach

April 26, 1992

Contents

List of Figures

List of Tables

1 Abstract

This paper outlines and examines the use of pre-computation and adaptive refinement methods in interactive virtual building walkthrough applications. In particular, the paper focuses on contrasting potentially visible set attainment and traversal methods. These methods reduce the polygon display space by rendering only a potentially visible subset of the model space for a particular viewpoint. This method requires model space subdivision pre-computation before display, but results in greatly increased display rates. In addition, adaptive refinement during display techniques are examined. These techniques use hierarchical model representations as well as a sieve approach to image refinement for increased scene realism.

2 Introduction

While the advance of modern hardware technology has allowed today's graphic superworkstations to reach rendering rates of hundreds of thousands of polygons per second, this is still too slow to support acceptable user interaction in complex static scenes. However, acceptable levels of interaction can occur by focusing on the static nature of most environments. In particular, the use of pre-computation and progressive refinement can be used to greatly increase the display performance in these environments. One such environment, the virtual building, is particularly well suited for such methods due to its inherent static and axial attributes. This environment also faces the additional demands of high-cost illumination models and real-time user interaction when used to produce realistic graphical "walkthroughs" of the building model. In this application of the virtual building, the goal is to produce the most realistic interior image possible at a framerate where a virtual-world illusion takes hold. Thus, the *virtual building walkthrough* (*VBW*) system faces two diametrically opposed goals; increased realism and detail (more polygons and high-cost

illumination models) vs. higher framerate (less polygons and low-cost illumination models). To satisfy both goals, a VBW system must use knowledge of its environment as well as of human visual perception and interaction to strike a satisfactory balance. Use of knowledge of the model environment and of the human user come in the form of precomputation and adaptive refinement, respectively.

In precomputation, *a priori* knowledge of the building model can relieve much of the burden of on-the-fly computations during user interaction. This approach can be used to examine all viewing areas to determine the visible set of polygons at any given instant and thereby reduce the visibility-space for that view. In addition, precomputation can effectively solve much of the lighting model calculations, making complexly illuminated scenes possible.

While precomputation is independent of the interactive walkthrough, adaptive refinement uses its knowledge of perceptual rates to make display-time adjustments *during* the walkthrough. Scene complexity is reduced or increased depending on the movement of the user. During stationary or slowly moving periods, more time is available for calculations and the image complexity is increased.

This paper will examine the use of the two aforementioned methods in virtual building environments to produce continuous user interaction at perceptually acceptable rates. Three systems are examined, each of which takes a different approach to using precomputation in the visibility-space reduction problem.

The first method analyzed, BSP trees, was originally developed to compute priority orderings of coherent viewing areas. Schumacker developed the BSP tree to partition space into these areas; Fuchs modified this approach to create a total visibility ordering of the space. Schumacker's notion of model space partitioning was later adapted to fit the virtual building environment. The latter two methods examined are of this genre. Airey's method is examined first. This method uses BSP trees to partition the model space, and visible polygon sets are computed for each "room". The last

method examined is Teller's visible set reduction solution. This method too divides the space using BSP trees; however only the set of visible "rooms" for each room are computed, not an exhaustive list of polygons.

Following the examination of the model-space reduction methods, the use of precomputation in the lighting calculations is detailed. In addition, Airey's system's use of progressive refinement under stationary camera positioning is detailed.

3 Background

Before presenting the specific methodologies to be analyzed, some background and terminology must be introduced. In order to understand the problems in achieving interactive display rates, this rate itself must be defined and the display process must be understood.

3.1 Motion Frame Rates

Visual perception is naturally attuned to the rotoscopic process. It is this fact that allows the movie illusion to be perceptually fluid. However, this illusion falters when the frame rate decreases under some threshold. This threshold is not absolute; instead it depends on the relative velocity of the viewer and scene among other factors. Video and film rates are respectively 30 and 24 frames per second (fps), and in most instances this is adequate for creating smooth motion. However when some object in a scene moves too much from one frame to another, temporal aliasing occurs. This results in at best a perception of jerkiness in the animation, and at worst a complete loss of motion sensation.

In an interactive building walkthrough, the environment is static and camera motion is relatively slow. This allows the motion continuity perception to start at rates as low as 6 fps [ARB90]. The visual illusion and corresponding interactivity increases as the frame rate increases. At rates greater

than 20 fps, the perceptual gains lessen.

3.2 Display Process

Most graphical system rely on polygonal geometry as the basis of its display primitives. This is especially true for the display of architectural structures due to the inherent polygonal nature of most buildings. For this reason, this paper will not discuss other solid model representations.

The display of polygonal objects, or segments, is typically accomplished in a graphical pipeline. This process consists of traversing the display model and obtaining primitives for the next pipeline state. These primitives, in this case convex polygonal faces, are then passed to a geometry stage which transforms and clips this object-space data (the polygons' vertices) into screen coordinates. The screen coordinates are then used to rasterize the primitives thereby "filling in" the polygons in screen coordinates. It is in this stage that individual pixel values are calculated, and correct depth cuing is generally performed. This screen data (pixel data) is then displayed by a display subsystem.

As was noted above, the rasterization subsystem performs depth cuing. This process displays only polygons which are "closer" to the viewer. This operation is done in most systems using a variant of the *Z-buffer* technique. This technique uses the Z-coordinate or depth of the particular point of the viewpoint-transformed polygon it is displaying. If and only if this point's depth or Z value is less that that of the existing pixels Z value, it overwrites that pixel. Depth or Z values are therefore maintained for every pixel on the screen, thereby resulting in a "Z-buffer".

The above display process is the foundation of most graphical superworkstations, with variations in the parallelization which occurs within the pipeline. In each case, however, hardware limits the processing rate of the primitives to current rates in the order of 10^6 primitives per second. In a comlex building environment, having 10^6 convex polygons has become commonplace, with many

environments containing orders of magnitude more. In addition, in radiosity modeled environments each segment's polygonal faces are divided into smaller *patches* to facilitate more complex lighting models. This further increases the display list of polygonal primitives. Furthermore, current hardware provides frame buffer bandwidths of approximately 40 million pixels per second. This is already severely taxed due to the raster overwrite rendering methods which potentially render each polygon fully [MF89]. Speedup of this rendering pipeline is limited to only such user-controllable matters as the lighting model used or whether or not Z buffering is enabled.

Whereby these two display parameters are controllable, they in themselves do not contribute greatly to display speedup. The overwhelming factor determining display frame rates remains the shear number of primitives passed through the pipeline. For each frame image, every polygon in the model is sent through the pipeline. This *display list*'s length is directly proportional to the time required to display the image. While this might seem uncontrollable for a particular environment, reduction can be accomplished if one notes that at any instance in a scene, many polygons are not visible at all. This fact is magnified in a building environment, where walls act as delimiters of the environment. Assuming opacity of the walls, they obscure the neighboring rooms. Doors and windows, together defined as *portals*, are the exception to this observation.

Therefore, any system which through pre-computation can determine the visible set of polygons and thereby reduce the number passed through the display pipeline can increase the system frame rate. This is the commonality of the described methods in this paper, known as the *potentially visible set* or PVS problem [ARB90], and is this paper's major focus. Other methods for speedup such as described above are also later discussed. The following section presents the basis of many of the current PVS methods employed today.

4 Fuchs' BSP Trees

One of the earliest effective preprocessing methods for performing visibility calçulations is the binary space-partitioning (BSP) tree algorithm. This method has many applications in solving the PVS problem, with several outlined below.

The BSP tree method was developed by Henry Fuchs, Zvi Kedem, and Bruce Naylor [FKN80]. The approach is based on a method of space subdivision developed by Schumacker [SBGS69] and explored by Sutherland [SSS74]. Schumacker's method used the observation that if a plane can be found which wholly separates one group of faces from another, then faces on the same side of the plane as the viewing position cannot be obscured by any of the faces on the other side of the plane. These groups of faces Schumacker termed *clusters*. By manually introducing invisible dividing planes into the scene, areas could be built in which the obscuring priority of the cluster is known through generation of what Fuchs later termed a BSP tree. Fuchs' use of BSP trees uses the scene polygons themselves as the dividing planes to create the tree. Before detailing the creation and exploitation of the BSP tree in calculating the PVS, an introduction to the BSP structure is necessary.

4.1 The BSP (k-d) Tree

The BSP tree is a particular instance of the k-d tree, formally developed by Bentley [Ben75]. The k-d tree is a data structure for the storage of k-space dimensional data for the efficient retrieval through associative searches. Bentley introduced various operations for the k-d tree. He showed that the cost of an initial sort is $O(n \log n)$, demonstrated that neighbor queries are linear time ordered, and provided a method for general intersection queries. The importance of these contributions will be seen shortly.

The k-d tree is a data structure which stores k-tuple data records in a multidimensional binary

search tree. In this tree, each record represents a node and each node may have none, one, or two children. In addition, each node has associated with it a discriminator field used in creation and searches. These features are described in detail in [Ben75].

As formerly noted, BSP trees are specific invocations of k-d trees, where k is the dimension of the graphical scene space (in VBW systems, $k = 3$). In every BSP representation, internal nodes represent the $k-1$ dividing plane which divide a space into two subspace regions. In a 2-dimensional space, these nodes represent a dividing line and each subspace is itself a half-plane. In 3-dimensional space, the nodes represent a 2-dimensional plane splitting a volume into two half-space volumes.

4.2 Schumacker's Use

The development of the BSP tree can be traced back to Schumacker's 1969 [SBGS69] introduction of a informalized hidden-surface algorithm. This method was further explored by Sutherland, Sproull, and Schumacker's Characterization of Ten Hidden-Surface Algorithms [SSS74], where notions of its coherence properties were first noted.

Schumacker's use of the then unamed structure makes use of a tree structure to calculate the *cluster priority* depending on the viewpoint location. In this application, a cluster is taken to be a collection of faces that can be assigned a fixed set of priority numbers which provide correct view-independent visibility ordering after backface culling (see Figure 1). This ordering is for polygons within each cluster; inter-cluster priority follows directly from the BSP tree. This method does however rely on human selection of the clusters and the subsequent introduction of *invisible*[1] partitioning planes into the environment before creating the tree.

Once a tree is created, its internal nodes represent the partitioning planes and its leaves are regions in space. Associated with each leaf is a priority ordering (see Figure 2). This ordering

[1] Invisible here means not part of any display list and only used in calculations

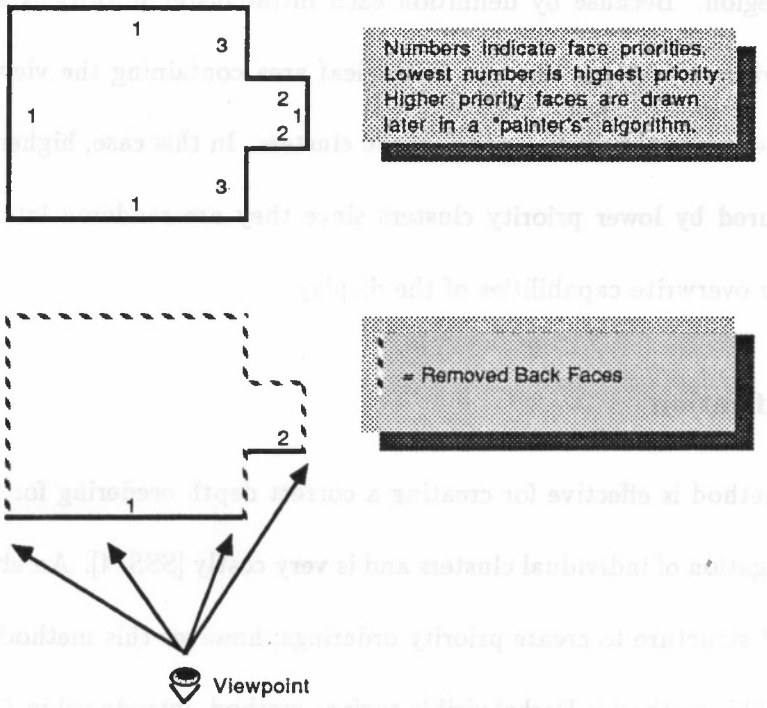


Figure 1: Intra-cluster Polygon Priorities

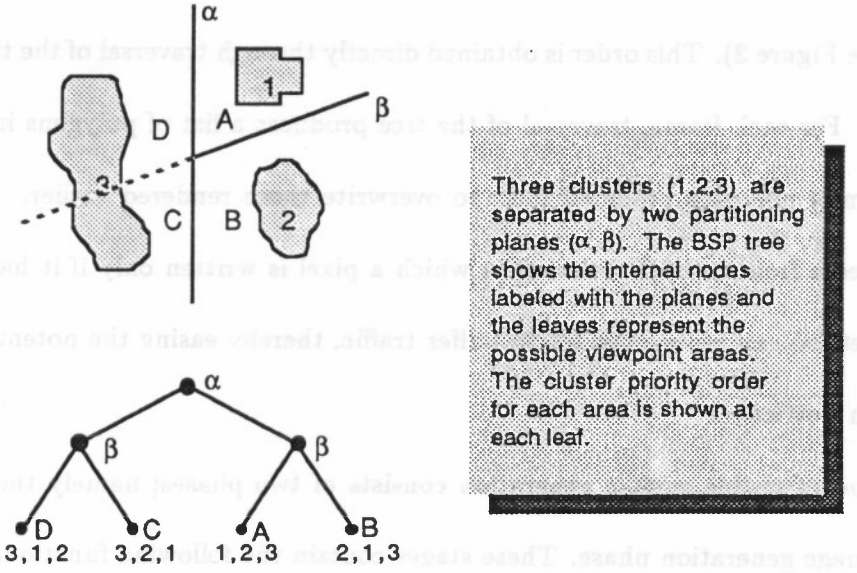


Figure 2: Inter-cluster BSP Tree

provides a unique determination of which clusters can obscure one another if the viewpoint is located within that region. Because by definition each intra-cluster priority is known, a correct image can be rendered by traversing the tree to the leaf area containing the viewpoint and using its cluster priority ordering as the display order of the clusters. In this case, higher priority (closer) clusters are not obscured by lower priority clusters since they are rendered later in the process, thereby relying on the overwrite capabilities of the display.

4.3 Fuchs' Modification

While Shumacker's method is effective for creating a correct depth oredering for any view, it does rely on operator segregation of individual clusters and is very costly [SSS74]. An alternative method also relies on the BSP structure to create priority orderings; however this method does not require cluster discernment. This method is Fuchs' visible surface method, introduced in *On Visible Surface Generation by A Priori Tree Structures* [FKN80] using a formalized and named BSP tree.

Fuchs' BSP algorithm uses the environment's polygons themselves as the partitioning planes of the scene space. This produces not a cluster priority order; rather the BSP tree creates a polygon priority order (see Figure 3). This order is obtained directly through traversal of the tree in modified in-order method. For each frame, traversal of the tree produces a list of polygons in back-to-front order. This permits polygons rendered later to overwrite those rendered earlier. An alternative traversal produces a front-to-back ordering in which a pixel is written only if it has not yet been written. This method can reduce the frame buffer traffic, thereby easing the potential for frame a buffer bandwidth bottleneck.

Fuchs' method of visible surface generation consists of two phases; namely the preprocessing phase and the image generation phase. These stages contain the following functionality:

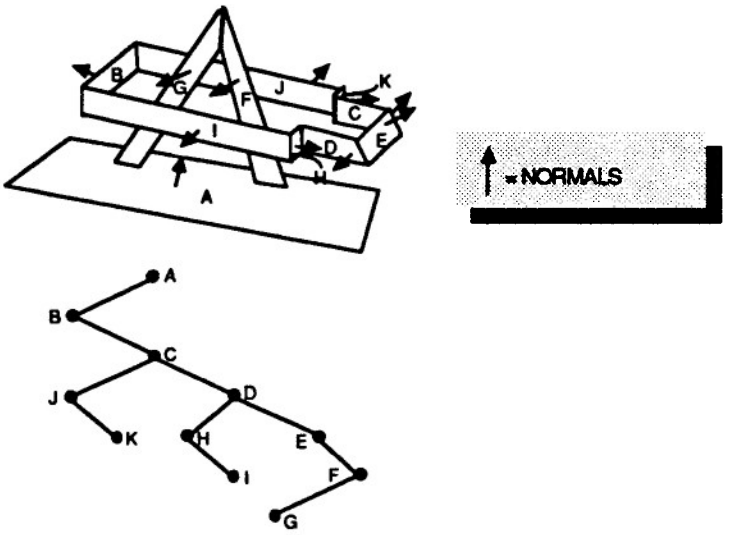


Figure 3: Example Model and Corresponding BSP Tree

- PREPROCESSING: In this step, the BSP tree is actually created. This involves recursively dividing the polygon space according to which side of a dividing plane each polygon falls on. Polygons incident with this plane are split in this step. Polygon splitting is performed only in this stage.
- IMAGE GENERATION: This step occurs repeatedly during the display process. For each frame, a complete traversal of the tree occurs with the traversal order according to the viewpoint location. During each traversal, the polygons are either assigned priorities for later display or are painted during the traversal.

Each stage is formalized below.

4.3.1 Preprocessing

The preprocessing stage consists of creation of the BSP tree given the set of directional[2] polygons $P = \{p_1, p_2, \ldots, p_n\}$. In choosing a polygon p_k, the space P is divided into two half-spaces S_k and $S_{\bar{k}}$, indicating positive and negative sides of plane on which polygon p_k lies, respectively. All polygons are classified according to in which half-space they lie. If a polygon q falls incident to this dividing plane, it is divided at the line of intersection into two new polygons q_{S_k} and $q_{S_{\bar{k}}}$, each of

[2]We assume each polygon is directional according to the direction of its normal

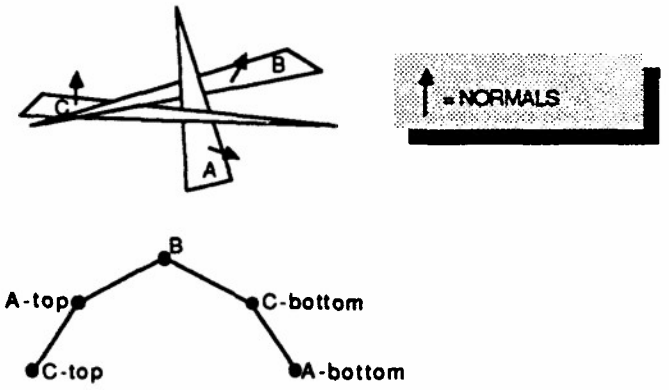


Figure 4: Example Model and Corresponding BSP Tree with Split Polygons

which gets placed in its respective space (see Figure 4). Each half space is similarly divided selecting a polygon from the respective subspace. This recursive process is repeated until all polygons have been selected as a dividing polygon. This is demonstrated in the following pseudo-code:

```
Tree Make_BSP_tree(List polygon_list)
{
   Polygon  qpos,qneg;
   List_index i,k;

   k=Select_dividing_polygon(polygon_list);     /* k : polygon's index   */
   pk=polygon_list[k];                          /* pk : dividing polygon */
   pos_list=NULL;                               /* Sk : positive subspace*/
   neg_list=NULL;                               /* Sk-: negative subspace*/
   FOR i=1 TO Size_of(polygon_list) DO {        /* All polygons of space */
      IF (i <> k) THEN {                        /*  except dividing poly */
         /*Sets qpos to polygon i if entirely in positive subspace.
          *Sets qneg to polygon i if entirely in negative subspace.
          *If polygon i is in both subspaces, splits polygon into
          *two polygons at line of intersection and sets qpos to
          *positive half and qneg to negative half.
          */
          Split_polygon(polygon_list,i,k,qpos,qneg);
          Add (qpos, pos_list);                 /* Put pos poly in Sk */
          Add (qpos, pos_list);                 /* Put neg poly in Sk-*/
```

```
      }END IF;
   }END FOR;
   tree.left_child = Make_BSP_tree(pos_list);   /* left child        */
   tree.node = pk;                              /* node              */
   tree.right_child= Make_BSP_tree(neg_list);   /* right child       */
   RETURN (tree);
}
```

As can be infered from this process, the possible splitting of each polygon during each iteraction can result in exponential growth of the display set. This resulting BSP tree is created **once** during the preprocessing stage and is subsequently traversed **once per frame** during the image generation phase.

4.3.2 Image Generation

At each frame, the BSP tree created above is traversed for the current viewpoint. As noted previously, this can produce a priority ordering or directly drive a *painter's* algorithm[3], which paints each polygon onto the screen as it is encountered. The pseudo-code for the latter approach follows.

```
void Back_to_front(Position eye; BSP_tree tree)
{
   IF Not_null(tree) THEN {
      IF pos_side_of(tree.root, eye) THEN {    /*Eye in pos half  */
         Back_to_front(eye, tree.right_child); /* negative branch */
         Display_polygon(tree.node);           /* current polygon */
         Back_to_front(eye, tree.left_child);  /* positive branch */
      }ELSE{                                   /*Eye in neg half  */
         Back_to_front(eye, tree.left_child);  /* positive branch */
         Display_polygon(tree.node);           /* current polygon */
         Back_to_front(eye, tree.right_child); /* negative branch */
      }ENDIF;
   }ENDIF;
}
```

[3]A painter's algorithm renders polygons in a back-to-front order with closer polygons overwriting any pixels previously drawn

The analogous front-to-back algorithm for writing to a pixel only if it has not been written is directly derivable from above.

4.4 Characteristics of BSP Tree

As seen in Section 4.3.1, the possible splitting of the polygons can result in a k-exponential growth in the number of polygons, where k is the dimension of the space. This growth can be limited by several heuristic approaches. Among these are choosing the root polygon at each stage to be the one which splits the minimum number of polygons in its list, or in concert, maximizing the number of "polygon conflicts" eliminated as detailed in [FKN80]. Testing only a few polygons (five or six) provides a good approximation to the best selection [FAG83].

In addition, the use of the k-dimensional BSP tree for space subdivision results in a worst-case number of subspaces (leaves) given n polygons defined by:

$$f_k(n) = \sum_{i=0}^{k} \binom{n}{i}$$

resulting in $2f_k(n) - 1$ nodes in the tree.

Fuchs' also gives formulas for the number of interior nodes given unbounded and bounded polygons. These formulas were given as

$$\sum_{i=0}^{k} i\binom{n}{i}$$

and

$$\binom{n}{2} + n^{k-1}$$

respectively.

The second equation as given is incorrect, partially due to an assumed typing mistake. The

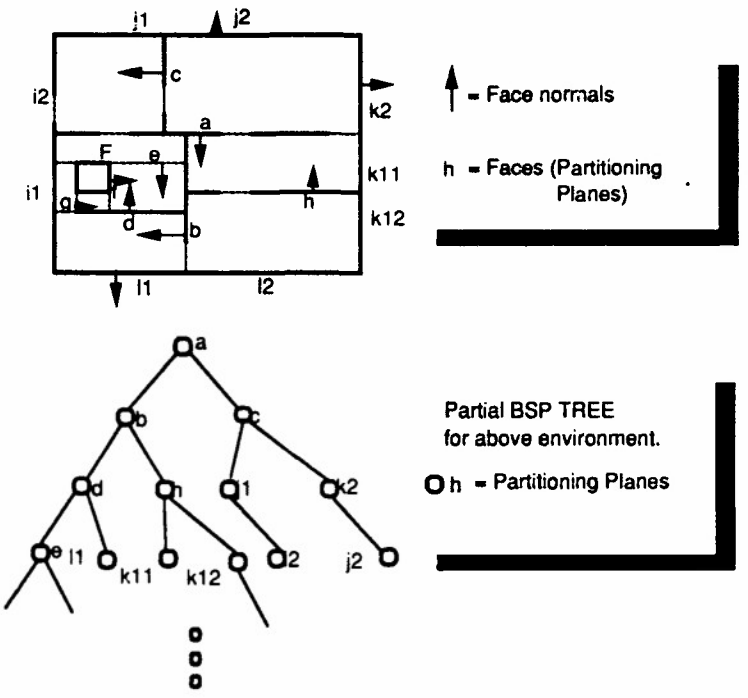


Figure 5: Fuch's Model Subdivision Using BSP Tree

corrected equation is

$$\binom{n}{k} + n^{k-1}$$

although this too overstates the bound, as seen by the fact that the sum of the inner nodes and the leaves should equal the total nodes, which for the three-dimensional case does not. Regardless, this gives a close approximation to the maximum number of interior nodes.

Although the above worst-case representations display polynomial growth, experimental evidence suggests typical increases of only single digit multiples on n [FAG83]. This implies a practical use of this implementation. A sample virtual building model partitioning using a partitioning heuristic shows a well-balanced tree as seen in Figure 5.

heuristic shows a well-balanced tree as seen in Figure 5.

4.5 Extended Uses of the BSP tree

If the initial set of polygons is equivalent to the extended planes, this results in maximum tessellations. This enables precomputation of each area's priority ordering, since it is not possible to be on two sides of a polygon extension in the same area. This application, however, is extremely memory intensive.

Fuchs implies the use of the BSP tree to assist in clipping. This is addressed in [FvDFH91] where any polygon extension not intersecting the view volume contains one subtree lying outside of the volume. This branch can therefore be eliminated from further consideration. This is indeed a powerful tool in reducing the display list at a particular viewpoint.

4.6 Comments on BSP Trees

The two described BSP tree uses demonstrate different approaches and applications for the structure. While Fuchs' use of the model's polygons as the partitioning planes provides a useful foundation for automatic partitioning, the resulting priority polygon ordering presents little savings in today's Z-buffering superworkstations. Instead, Schumacker's use of the BSP tree for creating coherent PVS regions is far more advantageous in VBW applications. A compromise of the two techniques using the model polygons as the partitioning planes and obtaining priority orderings of the subspaces proves to be the best tradeoff. Fuchs' solution only becomes effective for the VBW application when used to assist in clipping, for great reductions in the display list can be accomplished. This is also directly applicable to Schumacker's use if priorities are computed at display time instead of being pre-stored.

A possible extension of the clipping application is to make use of the frame coherence that

typifies the VBW system. Since view areas typically do not drastically change from frame to frame, it might be possible to use the previous viewcone location to facilitate BSP tree pruning, thereby saving location checking for each partitioning plane node. It is not clear whether the costs associated with such a method outweigh the iterative savings.

5 Airey's Approximate PVS Solution

While the BSP tree method is a general hidden surface method with functionality applicable to virtual buildings, it is not specifically tailored to the VBW problem. In virtual building environments, the visible polygon set does not change much from frame to frame. Since in Fuchs' application the BSP tree is traversed for each frame, it does not take into consideration this inherent *frame coherence* in such cell-based environments[4]. An alternative method which uses and relies on this coherence is Airey's automatic model-space subdivision PVS system [Air90]. This system divides the environment into intuitive rooms or *cells*, and uses intra-cell view coherence in its PVS calculations.

In *Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments* [ARB90], Airey presents a cohesive rendering package specifically geared toward producing real-time user interaction in a virtual building. This system uses extensive pre-computation before display to reduce the search space for the PVS problem. It also relies on adaptive refinement to produce highly detailed scenes whenever possible. The latter functionality is addressed in Section 7.

Whereas the BSP tree uses pre-computation to produce a traversal order for the entire model space for each view, Airey's method uses pre-computation to reduce the model space to potentially visible sets for related views. During the image generation stage, only the reduced PVS for each

[4]A modified use of the BSP tree makes use of frame coherence for maximum tessellation environments. This method involves storing last priority orderings for use if the next viewpoint is located in same coherent area

view is rendered in contrast to Fuchs' traversal of the entire model space.

5.1 Model Assumptions

Airey used several characteristics of the virtual building environment to guide the design decisions of the system. The relevant properties for the PVS problem are

- The model is changed much less often than the viewpoint
- Buildings have high average depth complexity and are therefore nearly isotropic.
- Most of a model is not visible from any given viewpoint
- There is strong frame coherence **except** when crossing *portals*, e.g. doors.

These features enable significant reductions in the average display lists at any given view. The first feature suggests that pre-computation is a viable method for speedup. The next two items suggest that reduction of the display list is valuable on average for all views. The last characteristic implies a natural decomposition of the building space into rooms for reduction of the PVS.

5.2 Subdivision Method

As noted above, rooms in a building represent natural cell space subdivisions which have strong intra-cell frame coherence. For all viewpoints within a cell, the union of all possible visible polygons is significantly smaller than the entire environment space. This reduced set of polygons is the potentially visible set (PVS) for that cell.

While the decomposition of a building space into rooms seems intuitive, there is no simple algorithm for computer implementation. Rather, Airey relies on a heuristic approach to accomplish this partitioning. His approach circumvents the major problem facing Schumacker's clusters by providing automatic division into cells (clusters). This partitioning process attempts to strike a balance between two conflicting objectives, namely

- Minimize the size of the potentially visible set to get the maximum space reduction for that set
- Minimize the number of sets to keep the space requirements at reasonable levels.

In addition, calculation of the PVS for each cell is of limited use unless the cell containing the current viewpoint is easily determinable. Otherwise this causes additional system latency since this determination is required for each frame. Therefore an efficient data structure is needed for location searching. For this reason, Airey uses a BSP tree compromise of Schumacker's and Fuchs' approaches. As noted earlier BSP trees inherit the k-d tree properties and therefore location can be determined in logarithmic time.

Like Fuchs, Airey's approach uses the polygons themselves as the partitioning planes. The objective of this partitioning is to generate volumes representing the cells, analogous to Schumacker's division of clusters. This partitioning is guided by the above objectives, and does so by critiquing each axial-aligned polygon as a possible partitioning plane at each recursion. The criteria considered are

- Balance: how evenly the plane separates the model
- Occlusion Factor: how well the plane hides the two sides from each other
- Split Factor: how little the plane splits individual polygons, which requires split polygons being put in both partitions

Each criteria is quantified between 0 and 1. These values are weighted and linearly combined to produce the following partition equation:

$$partition\ priority = .5 * occlusion + .3 * balance + .2 * split.$$

This heuristic was devised based on Airey's experience in hand-tooling the divisions and based on statistics on balance, split and occlusion factors in sample environments. This equation is only

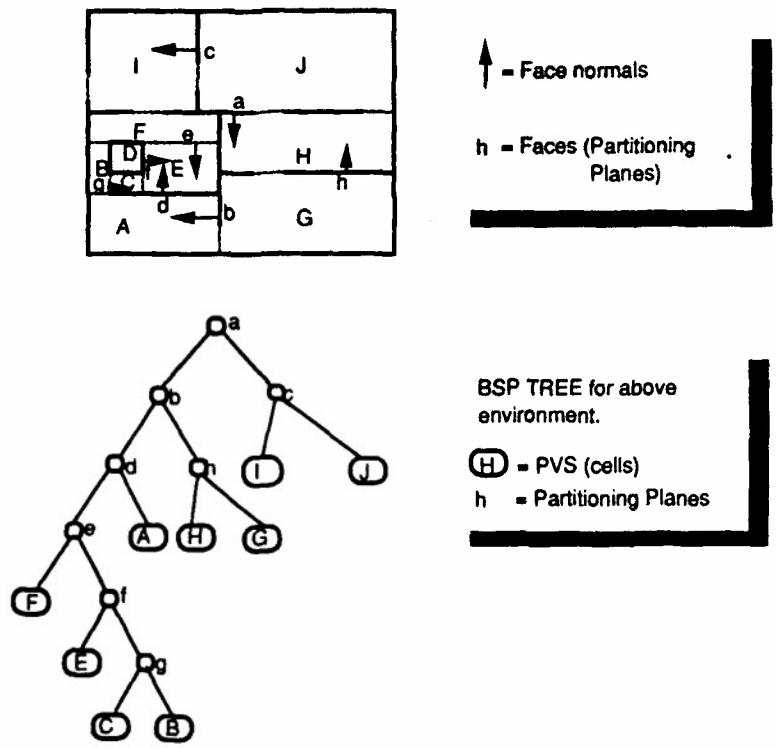


Figure 6: Airey and Teller's Model Subdivision Using BSP Tree

an approximation of the true relation. Because of a quadratic expense for the balance and split calculations, Airey uses an approximation which is linear in the number of faces [Air90]. Because this approximation's accuracy declines as the cell is further divided, the associated weights of these criteria are attenuated as the BSP tree depth increases. Airey readily admits other good partitioning methods may exist; Section 6 details an alternative approach.

Since the objective of the partitioning process is to produce regions, not an exhaustive partition, this algorithm must make determinations as to when an acceptable level of PVS coherence has been reached. Again, Airey uses a heuristic approach to terminate cell division based on the following criteria [Air90]:

- No partitioning plane is found whose partition priority exceeds a user-defined threshold
- The volume of the cell is below a certain value based on an "average" size room
- A predefined limit on the depth of the BSP tree has been reached, effectively limiting the viewpoint location time during display.
- The number of polygons in the cell is less than a preset value.

The thresholds provide control over the resulting partitioning of the model space into the leaves of the BSP tree. A sample partitioning is shown in Figure 6.

5.3 PVS Calculation

The use of model space subdivision as described above is to provide coherent areas or cells in terms of the PVS of that cell. The display system then renders only the PVS of the cell in which the viewpoint is located. This requires precomputation of the PVS for each cell. If a cell in enclosed, the PVS for that cell is simply the set of polygons comprising that cell, much as in a room with no doors or windows. If there are holes or *portals* in the cell boundary, i.e. doors and windows, then polygons from other cells are potentially visible and the set becomes more complex.

The visibility set of this cell, however, is equal to the visibility set of the portals. This in itself is a complex problem, especially since the portals are unrestricted in structure. To simplify this problem, Airey developed a method of portal decomposition based on a plane sweep algorithm [Meh84] which reduces the possibly concave portal to a collection of convex triangular regions. The visibility of the portal is therefore reduced to the union of the visible sets of all triangles.

To summarize the above procedure, the model space is divided into cells, and the PVS of each cell is exactly the union of the visible polygons of each of the triangular decompositions of the cell's portals. This problem of determining the visible polygons is equivalent to determining the set of polygons illuminated by an area light source [NN85]. For this reduced problem, an exact solution is very costly and difficult. Even an good approximate solution may take a month to process a

7000 polygon model on a 10 MIPS workstation [Air90].

Airey therefore relies on using estimation approaches which border the true solution. One approach produces an underestimation of the PVS, possibly resulting in "holes" in the final display. The second approach produces an overestimation of the PVS, resulting in reduced speedup results. Airey compares several tested methods, summarized in Table 1. The environment-independent sampling methods are ray-casting sampling methods based on radiosity hemicube [CG85] and hemisphere solutions. Variations of these methods such as rotated sampling and jittering are shown to increase performance. Airey also explores the use of environment-dependent sampling based on targeting other cells' portal triangles in a centroid-gravitated sampling. Airey only implements one overestimation method, although he details several in [Air90]. His solution is based on the computation of shadow volumes introduced by [Cro77] and extended by [NN83]. Shadow volumes compute the polygons shaded (occluded candidates) by a shadow-casting polygon (occluding polygon set) from an area light source (portal triangle). Due to Airey's simplification of the process, his method produces an overestimation of the polygons "shaded".

Airey notes that although the exact solution is bounded by overestimation and underestimation methods, he has no current method to effectively combine the results. In practice, his system uses only the sampling based underestimation methods. This is glossed over in [ARB90], but more fully explored in [Air90]. Depending on timing restrictions, Airey first "tweaks" run-time parameters to get the desired cell partitioning resolution. The PVS calculations are then distributed to as many workstations as possible. After most machines are finished, the remaining processes are killed and restarted with less demanding parameters. He then views the results, noting cells with "noticeable" missing polygons. For these cells, he recomputes the PVS using edge-deletion or overestimation methods.

PVS Computation Methods				
Method	CPU Time (seconds)	Relative Time	PVS Size	Relative PVS Size
392 ray cosine-weighted hemisphere. Source points spaced 24"	2235.81	1.637	1026	0.647
450 ray linear-radius hemisphere. Source points spaced 12"	7137.24	5.225	1259	0.793
5000 ray linear-radius hemisphere. Source points spaced 12"	54909.91	40.198	1490	0.939
1 targeted ray per polygon. 1 source point per portal	1365.99	1.00	1509	0.951
7 targeted ray per polygon. 1 source point per portal	7366.80	5.393	1531	0.965
7 targeted ray per polygon. Source points spaced 24"	47,297.51	34.625	1.0	1587
Over-estimation	109,936.34	80.481	3211	2.023

Table 1: Preprocessing PVS Comparisons for One Cell (Airey)

5.4 Comments

As can be seen from Table 1, the precomputation costs of Airey's method are enormous. These costs are extremely prohibitive to a practical application [TS91].

There are other obstacles which Airey hardly mentions but which also inhibit an automatic PVS system. Foremost of these problems is the automation of the detection of missing polygons due to the underestimation procedure. While Airey "eyeballs" the correctness of the resulting display, this is hardly an adequate solution. Airey states the automation of his methodology simply amounts to managing a generic large scale computation, yet he provides no insights into the lone step of detecting these underestimations.

Another facet of Ariey's procedure which is not expounded upon is the treatment of non-axial aligned polygons. While Airey notes the use of a BSP tree for detection of intersections in the non-axial set, he himself notes that the algorithm would have to be extended to run efficiently [Air90].

Airey's approach also does not use viewing direction to limit the search space. He does mention the possible modification to store the PVSs for each portal instead of each cell and crop the list according to which portals lie in the viewing frustum. While this approach could produce significant display list savings, handling the overlap in portal lists could prove too costly due to the number of portal regions due to triangularization. A possible solution to this would be to provide a grouping of close portal regions to limit the overlap.

6 Teller and Sequin's Exact PVS Solution

Another approach which borrows concepts from the two previous approaches is Teller and Sequin's visibility preprocessing method described in *Visibility Preprocessing for Interactive Walkthroughs* [TS91]. Like Airey, Teller uses model space subdivision to construct coherent cells. Like Fuchs, Teller uses the model polygons to partition the space into a BSP tree. The goal of this approach and the aforementioned is different, however. Whereas Airey's approach computes a set of *polygons* which are seen from a portal, Teller's method computes a set of *cells* which can be seen from a portal.

6.1 Model Assumptions

As with Airey's approach, Teller's is tailored specifically for architectural floorplans. Consequently, his assumptions of the model follow Airey's. He also limits his focus to axial line faces, as does Airey in his choice for partitioning planes. Teller's method does impose one further restriction on the model; coordinate data occurs on a grid. This restriction is enforced to permit exact comparisons between positions, lengths, and areas and to simplify calculations. In the case of architectural models, the grid data requirement is very reasonable.

6.2 Subdivision Method

The subdivision of the model space into cells follows Airey's approach of using the polygons themselves as partitioning planes to form a BSP tree with leaves representing the resulting cell areas. Teller uses a different method to determine the best partitioning polygon at each step, however.

Before examining this procedure, some notation of Teller must first be introduced. At each step in the partitioning process, each polygon F under consideration is classified with respect to that node's cell as being one of the following:

- disjoint : F has no intersection with the cell
- spanning : F partitions the cell interior into components that intersect only on their boundaries
- covering : F lies on the cell boundary
- incident : F is none of the above (enclosed entirely in the interior)

In addition, Teller uses the notion of *cleaving* in which face A cleaves face B if the extension of face A intersects B on B's relative interior. This is analogous to Airey's *split factor* criteria.

Teller's choice of the partitioning plane is subject to the following procedure:

if a spanning face exists, split on the median spanning face;
otherwise split on a sufficiently obscured minimum cleaving abscissa.

Again, this is analogous to Airey's formulation. Here, sufficiently obscured minimum cleaving abscissa means the split factor is above some threshold. If there exists several minimum cleaving abscissa, the one closest to the median face is choosen.

The split factor threshold value is used to terminate the partitioning branches, as in Airey's method. In addition, partitioning ceases if the current cell has no incident faces which could possibly obscure a viewpoint.

As noted above, Teller's choice is in direct correspondence with Airey's except the weighting

scheme is different. A spanning face is simply a face with an occlusion factor of 1. Choosing the median spanning face is an attempt to account for the balance factor. The minimum cleaving criteria represents the split factor.

As Airey notes in [Air90], the costs in finding the balance and split criteria is quadratic in the number of faces. Teller's approximation in taking the median diminishes the balance complexity, while Teller's requirement that all polygons be axial-aligned reduces the complexity of determining the split factor to linear time in the number of faces in the cell. This is possible through performing an initial sort of all the faces, whereas this is not possible in Airey's system since his requirement is that only partitioning planes must be axial aligned. Although Airey's split-factor calculations might appear to be more unr[illegible]ive, the additional complexity actually would be of no benefit since Teller's goal is not concerned with internal cell occlusion.

The resulting BSP tree from this method again contains leaves which represent areas or cells. Along with each cell, the portals are enumerated and stored along with an identifier of the neighboring cell to which the portal leads. This in effect creates an adjacency graph over the leaf cells. The determination of the portals is simplified in creating a bounding-box around the portal or by decomposing the portal into rectangular regions similar to Airey's approach.

6.3 PVS Calculation

After the decomposition of the model space into room-like cells, Teller computes inter-cell visibility for each cell as part of the preprocessing step. This is in contrast to Airey's approach, which calculates cell-to-polygon visibility and not cell-to-cell visibility. Teller's result is therefore a superset of Airey's solution[5].

Teller's system does not stop there, however. During the display process, the PVS is further

[5] Assuming Airey's use of an exact or overestimation method

reduced by computing eye-to-cell visibility, a subset of the precomputed cell-to-cell visibility reduced by culling against the viewcone. The resulting set is still a superset of the true visible set, and consists entirely of the union of cells.

6.3.1 Cell to Cell Visibility

Cell-to-cell visibility is only possible through a line of sight which passes through portals. If a non-neighboring cell is visible, this *sightline* must pass through a portal sequence, i.e. two open doors. Therefore, the problem of determining cell-to-cell visibility reduces to determine sightlines which traverse portal sequences.

To accomplish this, Teller performs a depth-first search on the adjacency graph of the portals which produces an ordered portal sequence at each incremental step. At each increment, the sequence is checked for admittance of a sightline. Since the portals are axial aligned, the existence of a sightline can be solved in linear time to the sequence length [Meg83] [Sie90] in two-dimensional space and $O(n \log n)$ in three-dimensions. If no sightline exists at some stage, that particular recursive branch of the traversal terminates. The valid sequences are stored as a *stab tree* of reachable cells (see Figure 7). The nodes of the stab tree represent the potentially visible set of cells from the root cell, and the edges represent the portal sequence to reach the cells.

6.3.2 Eye to Cell Visibility

While the cell-to-cell visibility preprocessing can significantly reduce the display list of viewpoints located in that cell, it does not fully exploit the potential of the stab tree. Since the stab tree tells not only visible cells but also the portals which leads to each, it can be used to further cull member cells if they fall out of the current viewing area. That is, because a viewpoint has an associate field of view or view cone, some cells in the PVS might not be visible for a particular viewpoint and

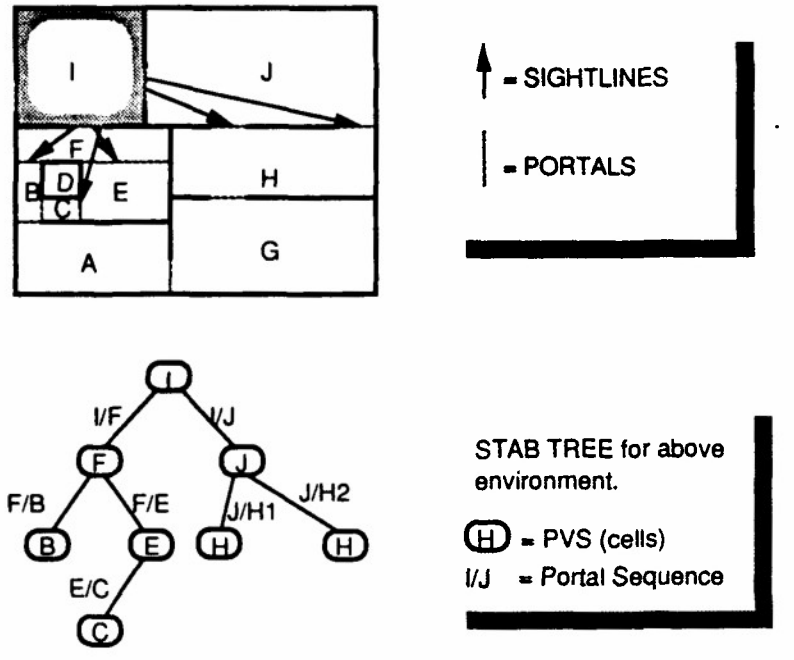


Figure 7: Sightlines and Stab Tree from Cell I

direction. Using the stab tree rooted at the viewpoint containing cell, cells outside the viewcone can be eliminated by several methods. These are listed below, in order of increasing effectiveness and computational complexity.

- Disjoint Cell Cull : removes cells which fall outside the view cone.
- Connected Component Cull : removes branches of the stab tree which have the branch root cell outside of view cone.
- Incident Portal Cull : removes branches of the stab tree which have the portal edge outside of the view cone.
- Exact Eye-to-Cell Cull : removes cells which have no sightline from the viewpoint lying inside of the view cone.

The exact approach uses a depth-first search traversal of the stab tree to successively narrow the visible area "wedge" (see Figure 8, top). The resulting pruned stab tree is shown in Figure 8, bottom. The result of this reduction in model space is seen in Table 2 for a two-dimensional model

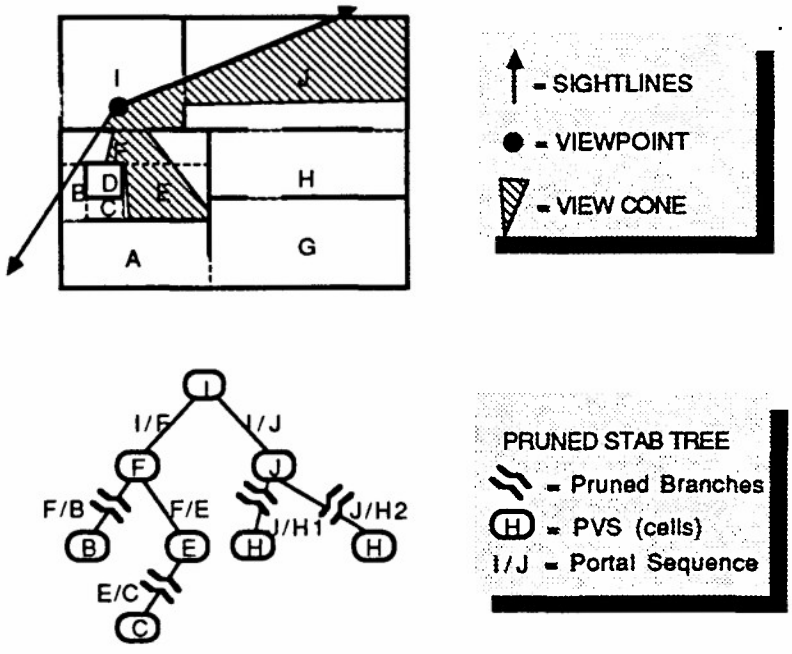


Figure 8: View Cone and Pruned Stab Tree

consisting of 1000 axial faces and 10,000 visibility queries. The precomputation of this model took a total of about 45 CPU seconds, and resulted in roughly 10,500 stab tree vertices. Teller provides no empirical data on three-dimensional computations.

6.4 Comments

As Teller's three-dimensional implementation was not complete at the time of the paper, no speedup vs. cost figures are available. The two-dimensional data does suggest a significant potential savings. The ability of the system to handle environments with large numbers of portals is yet to be seen.

Teller discusses the extension to non-axial systems and the major problems in implementation. A compromise solution could follow Airey's example and permit only axial-aligned partitioning planes. While this does not permit non-axial cells, it does support more generic models.

Teller also notes the use of frame coherence to expedite the display process. Stab trees are valid

PVS (Cell) Computation Methods				
Culling Method	360 deg view cone		60 deg view cone	
	vis. area	reduction factor	vis. area	reduction factor
none (cell-to-cell visibility)	8.1%	10x	8.1%	10x
disjoint cell	8.1%	10x	3.1%	30x
connected component	8.1%	10x	2.4%	40x
incident portals	8.1%	10x	2.2%	40x
exact eye-to-cell	4.9%	20x	1.8%	50x
exact visible area	2.1%	50x	0.3%	300x

Table 2: Preprocessing PVS Comparisons for One Cell (Teller)

within cells, and the eye-to-cell culling might benefit from this property as well.

7 Adaptive Refinement : Realism vs. Time

While the above precomputation strategies reduce the demands on the display system, they cannot be applied to all facets of the this process. In order to increase performance during times when more computation time is available such as with slow or non-moving viewpoints, adaptive refinement is used. This strategy uses hierarchies of model space representations and *sieve* technique formally defined in [LNL87], which pass an input through refinement sieves so long as time is available.

Airey's system relies on using a radiosity lighting model due to its linear properties and precomputation capabilities. Radiosity models subdivide model polygons into smaller *patches* and computes color values for these patch vertices during a precomputation step. Since typical hardware is time dependent on the number of rendered polygons, smaller patches increase display time. For this reason, a grosser resolution of patches is precomputed for the model and a secondary level of patch refinement is available as time permits.

Further refinement of the image is done through functional sieves. The first sieve is an intelligent triangulation of the quadrilateral patches for bilinear interpolation of the color vertices. Normal

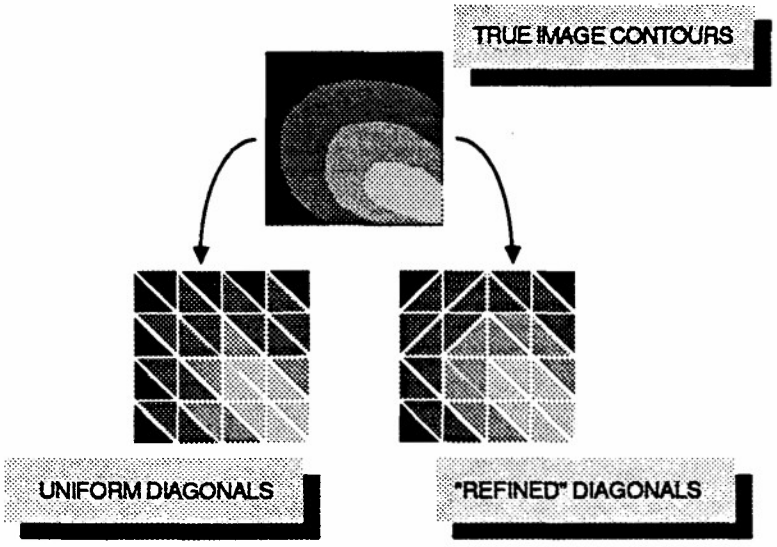


Figure 9: Uniform Quadrilateral Diagonals vs. Difference Directed Choice

triangulation may not follow the natural color coherence of the vertices. If time permits, vertices are compared and the triangulation occurs according to color closeness of opposing vertices as in Figure 9.

The second refinement sieve is anti-aliasing through supersampling methods. Additional images for the current viewpoint are calculated and blended smoothly into the accumulated image using appropriate filter weights.

8 Comparisons and Conclusion

While the above methodologies convey a chronological progression of PVS problem solving, it is interesting to note that the most current method of Teller has come full circle back to Schumacker's use of the BSP tree for separating clusters. While neither Schumacker nor Fuchs formulated the BSP tree around VBW applications, its application is readily apparent in Airey and Teller's systems.

Teller's method appears to hold the most promise of a real-world application. While Airey's

method is the more powerful in terms of preprocessing reduction, his system suffers from exorbitant precomputation costs as well as unresolved necessary user interaction. Teller's method may not initially provide the reduction in space matching Airey; however the eye-to-cell culling provide a powerful tool for display-time reductions.

Airey's system is however the only complete VBW system of the systems named. Valuable insights into the problems of such a system are presented in [Air90], although the precomputation problems are somewhat belittled. The paper does offer significant contributions in the area of portal decomposition as well as presenting some practical adaptive refinement techniques. While his system does suffer from serious shortcomings, he provides a solid foundation for possible future work as well as an overview of a project of such large scope. He details many of the problems in implementation, such as floating point size limits and relative versus absolute position comparisons.

Teller's system uses many of the same techniques of model subdivision as Airey, yet his system trades accuracy for simplicity and speed. His additional model requirements facilitate faster calculations, and his simpler partitioning plane selection process is faster for almost every evaluation criteria. While this speedup is only for the model partitioning stage of preprocessing, this is important because it is largely sequential in nature while the PVS calculations can be accomplished in parallel. Whether Teller's tradeoff helps or hurts in producing a reasonable subdivision at faster speeds is yet to be seen. A possible extension to this and Airey's system would be to parallelize the subdivision process by spawning off subprocesses to handle each subspace partitioning. This could be distributed over many machines.

Teller's system does suffer from potential problems in three-dimensional implementation. Foremost of the problems is the portal decomposition which Airey addresses. Teller suggests the use of decomposition into rectangles, but an implementation similar to Airey's triangulation would be necessary. He also notes the possible use of bounding boxes for the portals, yet this suffers from

additional overestimation consequences.

Teller's system also suffers from problems inherent to the approach. Since the system is only concerned with inter-cell visibility, minimal reduction is possible in models containing room-connecting hallways. The view from the end of a hallway cell contains all of the cells adjacent to the hallway. Therefore all cells must be processed. While eye-to-cell culling alleviates some of this during walking, the potential problem for certain models still exists. The extent of this potential will not be known until a three-dimensional implementation is complete. Even the two-dimensional data is minimal compared to Airey's testbed.

While the focus has been on the two latter methods, Fuchs' method still holds promise for an effective VBW implementation. As already noted, the viewcone culling can significantly reduce the display lists at any given viewpoint. In addition, this approach might benefit from Teller's use of viewcone narrowing from portals. By identifying portals during precomputation, the viewcone could be narrowed during BSP traversal as portals are reached to further prune the display list. This method has apparently not been mentioned in the literature. An advantage of Fuchs' method is that it also permits non-axial polygons, a major hurdle in the prior mentioned implementations.

All of the above strategies rely on some form of BSP tree. While both Teller and Airey attempt an intelligent heuristic for building the tree, more emphasis needs to be placed in all systems on examination and reconstruction of the tree for maximal balance.

Since precomputation is the basis of all of the presented methods, modifications to the model are very costly. Research into making minor modifications as well as limited interaction is still an open area. These interactions could include opening and closing doors, moving furniture, or changing lighting colors or intensities.

Further adaptive refinement techniques also need to be applied in future systems. Potential applications such as hierarchical building models and the use of varied lighting models could aid in

speedup and realism. This area holds great promise in permitting acceptable motion rates while maintaining high image resolution when necessary. Any effective VBW system must make use of these techniques as well as significantly reduce the display list at each viewpoint, for at least on current systems, hardware limitations remains the major consideration.

References

[Air90] John M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations.* PhD thesis, UNC Chapel Hill, 1990.

[ARB90] John M. Airey, John H. Rohlf, and Fredrick P. Brooks. Towars image realism with interactive update rates in complex virtual building environments. *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24(2):41–50, 1990.

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[CG85] Michael F. Cohen and D.P. Greenberg. A radiosity solution for complex environments. *ACM Computer Graphics (Proc. SIGGRAPH '85)*, 19(3):31–40, 1985.

[Cro77] F.C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (Proc. SIGGRAPH '77)*, 11(2):242–248, 1977.

[FAG83] Henry Fuchs, G.D. Abram, and E.D. Grant. Near real-time shaded display of rigid objects. *ACM Computer Graphics (Proc. SIGGRAPH '83)*, 17(3):65–69, 1983.

[FKN80] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. *Computer Graphics (Proc. SIGGRAPH '80)*, 14(3):124–133, 1980.

[FvDFH91] James Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics Priciples and Practice.* Adderson/Wesley, 1991.

[LNL87] *Imprecise results: utilizing partial computations in real- time systems*, December 1987.

[Meg83] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal Computing*, 12:759–776, 1983.

[Meh84] K. Mehlhorn. Data structures and algorithms 3: multi-dimennsional searching and computational geometry. In *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, 1984.

[MF89] S. Molnar and H. Fuchs. *Fundamentals of Computer Graphics*, chapter 18, Advanced Raster Graphics Architecture. Springer-Verlag, 1989.

[NN83] T. Nishita and E. Nakamae. Half-tone representation of three-dimensional objects illuminated by area sources or polyhedron sources. Proceedings of the IEEE Computer Society's International Computer Conference and Applications Conference (COMPSAC), 1983.

[NN85] T. Nishita and E. Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. *ACM Computer Graphics (Proc. SIGGRAPH '85)*, 19(3):23–30, 1985.

[SBGS69] R.A. Schumacker, B. Brand, M. Gilliand, and W. Sharp. *Study for Applying Computer-Generated Images to Visual Simulation.* Technical Report TR-69-14, U.S. Air Force Human Rescources Laboratory, September 1969.

[Sie90] R. Siedel. Linear programming and convex hulls made easy. Proc. 6^{th} ACM Symposium on Computational Geometry, 1990.

[SSS74] I.E. Sutherland, R.F. Sproull, and R.A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys*, 6(1):1–55, 1974.

[TS91] S.J. Teller and C.H. Séquin. Visibility preprocessing for interactive walkthroughs. *ACM Computer Graphics (Proc. SIGGRAPH '91)*, 25(4):61–69, 1991.