3-D Multiview Object Representations for Model-Based Object Recognition

by

Matthew R. Korn
Charles R. Dyer

Computer Sciences Technical Report #602

June 1985

3-D Multiview Object Representations for Model-Based Object Recognition

Matthew R. Korn*
Charles R. Dyer

Computer Sciences Department
University of Wisconsin
Madison, WI 53706

Abstract

3-D multiview object representations are presented as an alternative approach to traditional 3-D volumetric object representations. 3-D multiview models store features in a viewer-centered representation and thus can be immediately used to match features derived from 2-D images. Algorithms are presented that construct, search, and perform region growing on 3-D multiview object models.

Index Terms: Model-based image analysis, property spheres, modeling, neighbor finding, and 3-D object representations.

---

* Currently on leave from: Computing Systems Department, IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

This research was supported in part by an IBM Resident Study Fellowship and in part by the National Science Foundation under Grant ECS-8301521. David P. Anderson's contribution to Section 4.6 is gratefully acknowledged.

Table of Contents

1. Introduction

Model-based object recognition requires matching features measured in an observed image, with models of objects. The goal of the system is to return the identity, position, and orientation of each object in an image. When 3-D objects may appear at any position and orientation, such as in a robotics bin-picking task, 3-D object models must be used. Usually, objects are modeled using either one object-centered representation of many viewer-centered descriptions (one for each viewpoint).

In 3-D object-centered representations, objects are modeled by 3-D volumetric or surface models, which are rotationally-invariant and viewpoint-independent [Bro81a, Mar78]. Features extracted from a 2-D image (in the viewer-centered coordinate system) do not immediately correspond to the object models (in the object-centered coordinate system). 3-D to 2-D, or 2-D to 3-D transformations must be performed before the observed features can be matched with the model. This transformation can be time consuming; in applications where processing speed is important, simplifying assumptions that limit the generality of the viewpoint-independence assumptions are made to speed up recognition (e.g. [Bro81b]).

3-D multiview representations model objects by a finite set of viewer-centered descriptions. Each member of the set models the object by its 2-D projection as seen from one *viewpoint* on the *view sphere*. The entire set of 2-D models constitute an approximate model of a 3-D object.

3-D multiview object representations offer one major advantage over 3-D object-centered representations: features extracted from images can be directly matched with features associated with each member of the multiview model set, because both the model and the image represent features in the viewer-centered coordinate system. The shift from object-centered models to viewer-centered models shifts processing requirements from object recognition time to object modeling time. A significant amount of processing time is spent (once) to build the 3-D multiview representation of each object.

There is a trade-off between matching time and storage space: 3-D object-centered models can be compactly stored, but require significant resources to compute features that can be matched with features extracted from images. 3-D multiview models can be quickly searched for matching features, but require large amounts of storage. As the number of views in the model set of a 3-D multiview representation increases, the storage space and matching time required for the model increases. In the past, these two factors have served to limit the number of views in the 3-D multiview object model.

The number of views that comprise a 3-D multiview object model has been steadily increasing. Lieberman [Lie79] computes silhouettes of objects assuming objects can only occur in a few stable orientations. Wallace et al. [Wal81] compute the contours of aircraft from 143 viewpoints. Goad [Goa83] computes a list of visible object features (edges) from each of 216 viewpoints. Horn and Ikeuchi [Hor84, Ike82, Ike81] compute the extended gaussian image [Smi79, Hor83, Hor79] of an object from each of 240 viewpoints. Fekete and Davis [Fek83, Fek84] compute the silhouette of an object as it would be seen from each of 320 viewpoints. In all these methods, each member of the set is treated independently of the others. Object recognition algorithms based on each of these models sequentially search through the members of the model set to find a *best match* with features from the observed image.

Silberberg et al. [Sil84] do not precompute a 3-D multiview object model; at object recognition time a variable resolution multiview model is built on the fly. Views of the object model are computed from 80 viewpoints and are matched (using a generalized Hough technique) with features from an image to find the *most probable* match. Additional viewpoints are generated in a small neighborhood around the most probable viewpoints to increase the accuracy of the match. This process is iterated until a solution is converged upon.

The following basic operations have been previously defined and implemented for 3-D multiview object representations:

(1) Hierarchically sample the view sphere [Sil84, Fek84]

(2) Find the viewpoints that are adjacent to a viewpoint [Fek84].

(3) Compute the 3-D coordinates of a viewpoint [Fek84].

(4) Find all viewpoints with associated property P [Goa83].

(5) Find viewpoint V with associated property closest to property P [Hor84, Ike81, Ike82, Wal81, Cha82, Lie79, Fek84, Sil84].

We propose *region growing* as a solution to some of the major drawbacks of 3-D multiview representations discussed above. Adjacent views that are feature equivalent are grouped into *regions* or *neighborhoods*. Only one representative member of each region will then be stored, thereby reducing storage requirements and search time. Furthermore, regions will represent a continuous range of viewpoints; recognition of an object from viewpoints that are not members of the initial modeling set will now be possible.

Koenderink and van Doorn [Koe76a, Koe76b, Koe79] have shown that from almost all viewpoints about an object, small movements can be made without affecting the observed topological features of the object. These viewpoints are called *stable vantage points* and correspond precisely to the regions we are proposing.

Based on the same observations, Chakravarty and Freeman [Cha82] divide the continuous view sphere into a finite set of *vantage-point domains*. Within a vantage-point domain, all views contain the same junctions and lines (the projection of vertices and edges) of a given polyhedral model. Object recognition proceeds by classifying an unknown view as a member of one of the *characteristic view partitions* (the discrete representation of a vantage-point domain). Vantage-point domains also correspond to the regions we are proposing.

In this paper we present the operations that are required to perform region growing on a 3-D multiview representation. We also present several other basic 3-D multiview techniques. In particular, the new operations that we define are:

(1) Produce a depth-first/breadth-first ordering of the views.

(2) Partition the viewpoints into maximally connected regions such that all viewpoints of a region have the same value of property P.

(3) Find all viewpoints at distance D (on the surface of the view sphere) from a given viewpoint V.

(4) Find a pair of viewpoints V_1, V_2 such that V_1 has property P_1 and V_2 has property P_2 and V_1 is related to V_2 by spatial relation (constraint) S.

To clearly define these operations on 3-D multiview object models, we first describe one particular implementation of the representation. Section 2 describes one method that can be used to pick viewpoints. Section 3 describes three methods that can be used to implement a view sphere data structure based on the viewpoints chosen in Section 2. In Section 4 operations on the 3-D multiview object representation are defined.

2. Sampling the View Sphere

3-D multiview object models are constructed by *sampling* the continuous view sphere surrounding an object at discrete viewpoints. Sampling consists of (a) computing a 2-D projection of the object from each viewpoint and (b) extracting features from each 2-D projection. In the modern manufacturing environment, machine parts (objects) are initially designed with the aid of a CAD/CAM (computer aided manufacturing/computer aided design) system that stores the 3-D description of a part in an object-centered representation. A 2-D projection of an object from any viewpoint can be computed by most CAD/CAM systems. Features such as object silhouette [Wal81, Fek84, Lie79], extended gaussian image [Hor84, Ike82, Ike81], or edges [Goa83, Sil84] are computed from each 2-D projection. In this section, we describe how to select sample points (viewpoints) on the view sphere. In Section 3 we discuss appropriate data structures for storing features that have been extracted from each 2-D projection.

To select a set of viewpoints on the surface of the view sphere, start with a regular polyhedron, such as a cube [Goa83], a dodecahedron [Hor84, Ike82, Ike81], or an icosahedron [Bro77, Bro84, Sil84, Fek84]. The vertices of the polyhedron lie on the surface of the view sphere. The faces of the regular polyhedron are subdivided such that the projection of these facets on the view sphere divides the surface into many approximately congruent bins (i.e. they tesselate the surface of the sphere). The centers of these bins are chosen as the viewpoints. No method produces bins that (a) enclose equal solid angles, (b) are congruent, and (c) are separated from the centers of neighboring bins by equal amounts. (These properties only exist in five regular polyhedra, Euclid's Platonic solids: tetrahedron, cube, octahedron, dodecahedron, and icosahedron.)

2.1. Icosahedron Subdivision

We will choose viewpoints by recursively subdividing the faces of an icosahedron. This tesselation technique has the following advantages:

(1) All the bins are approximately congruent to each other in shape (i.e. they are all equilateral triangles). (In contrast to a dodecahedron subdivision, where the initial polyhedron faces are

pentagons, and the subdivided facets are isoceles triangles).

(2) The recursive subdivision is regular, can be infinitely repeated (i.e. the sample points can be made arbitrarily close together), and it need not be performed to the same depth on each facet.

(3) There are well known ways to calculate the position of the bin centers such that the bin centers are approximately equidistant from each other [Bro77, Bro79a, Bro79b, Bro84, Cli71].

The first viewpoints are the points on the view sphere that correspond to the centers of the icosahedron faces. See Figure 1. If additional viewpoints are required, then each face of the icosahedron can be subdivided: each edge of a face is divided into n equal segments (we choose $n=2$), and $n^2=4$ equilateral triangles are constructed on the face. These triangles are then "pushed out" so that their vertices lie on the surface of the view sphere.

Each time a face is subdivided in this fashion, 4 new faces are constructed. However, only 3 additional viewpoints are found because the viewpoint corresponding to the old face is the same as the viewpoint corresponding to one of the new faces. See Figure 2.

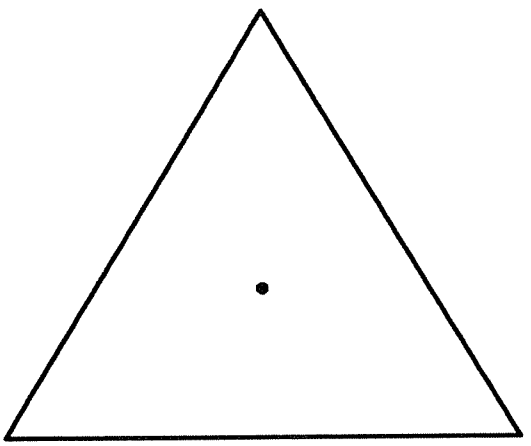

Figure 1: Icosahedron face: center of face is 1st viewpoint.

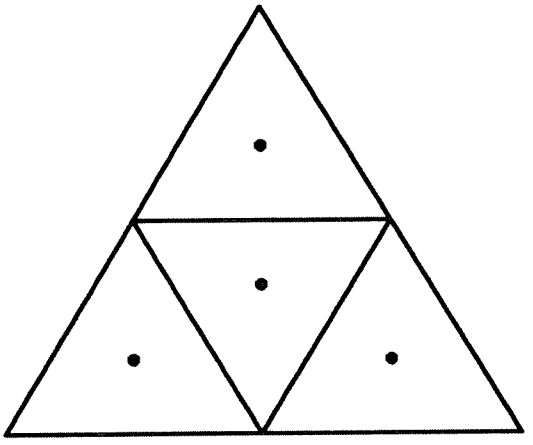

Figure 2: Icosahedron face, subdivided once, yields 4 viewpoints, 3 of which are new.

There are other tesselation techniques that will yield facets of more uniform size [Bro79a, Bro79b, Bro84, Cli71]. We have chosen the simple method above to demonstrate one particular tesselation.

2.2. Parents, Children, Neighbors

The icosahedron subdivision procedure described in the previous section generates a hierarchy of viewpoints. The initial 20 faces of the icosahedron form the highest *level* (level 0) of the hierarchy. The 80 facets derived from the 20 icosahedron faces form the next level (level 1) of the hierarchy. When a facet at level L is subdivided, 4 new facets at level $L+1$ are formed. The facet at level L is called the *parent* of the 4 facets at level $L+1$. The 4 facets at level $L+1$ are called the *children* of the facet at level L. Facets at level 0 do not have parents, and facets at the lowest level do not have children.

The 4 children are individually known as *child 0*, *child 1*, *child 2*, and *child 3* of the parent. See Figure 3 for the layout of the 4 children in both *up* and *down* facets. The viewpoint derived from child 0 is the same as the viewpoint derived from its parent. In the associated data structure storing

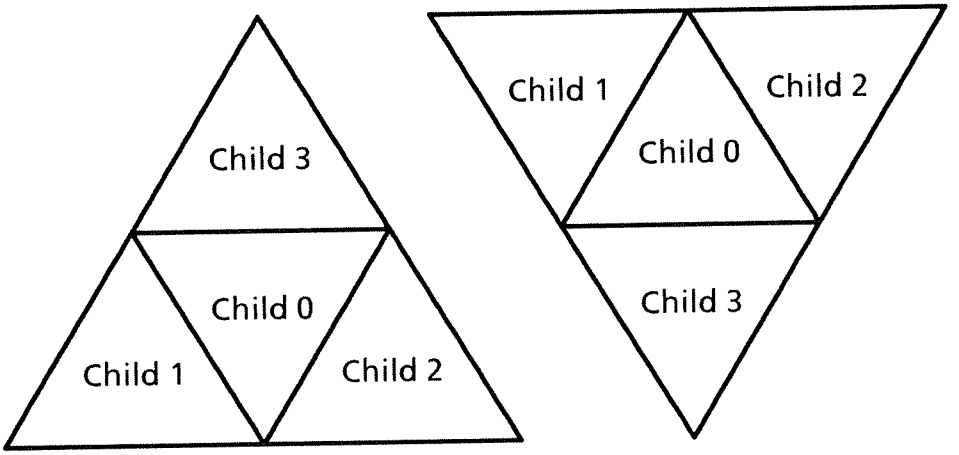

Figure 3: Children of *up* and *down* facets.

features associated with viewpoints, storage need not be allocated for a single viewpoint at *multiple* levels; it is only necessary to allocate storage for each distinct viewpoint at one level.

Edge adjacent facets on the view sphere are known as *neighbors*. Each facet has 3 neighbors at the same level as itself. Figure 4 shows the *1 neighbor*, *2 neighbor*, and *3 neighbor* of both an *up* and a *down* facet. Neighbors can exist at different levels when adjacent facets have been subdivided to different levels. An algorithm for finding all neighbors of a facet (at all levels) is presented in Section 4.2.

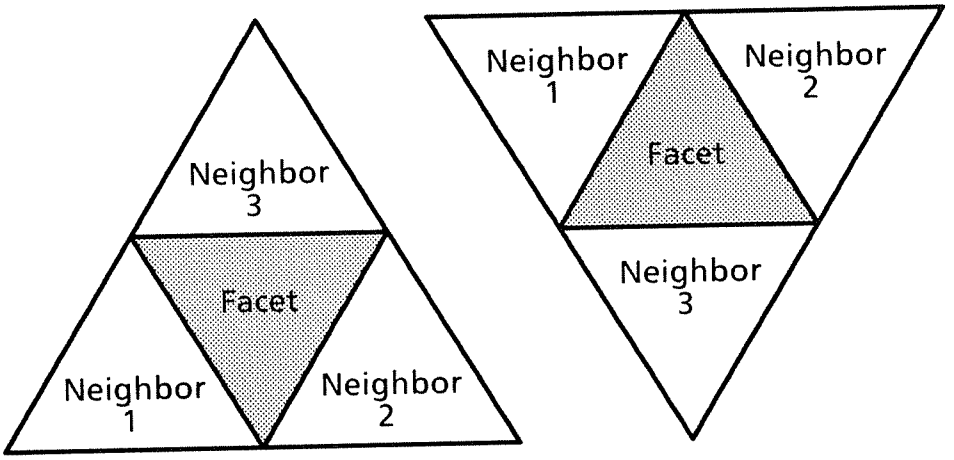

Figure 4: Neighbors of *down* and *up* facets.

3. Constructing a Property Sphere

Section 2 described a method to find sample points on the surface of a view sphere. A 2-D projection of an object centered in the sphere is computed at each sample point, features are extracted from each of the 2-D projections and are stored at the associated viewpoint. In this section we describe the implementation of several data structures that could be used to store features associated with each viewpoint.

The hierarchical viewpoint creation process (Section 2.1) is conveniently modeled as the expansion of 20 quadtrees [Fek84]. Fekete and Davis [Fek84] call this set of 20 quadtrees a property sphere. Each quadtree in the property sphere corresponds to the viewpoints obtained from the subdivision of a single icosahedron face. The quadtree with a single node corresponds to the original icosahedron face. Each time a facet is subdivided, 4 nodes are added to the quadtree as children of the node corresponding to the parent facet. See Figures 5 and 6 for the quadtrees corresponding to the facet subdivisions shown in Figures 9 and 10 respectively.

Since the viewpoint associated with a parent is the same as the viewpoint associated with its 0 child, it is not necessary to allocate storage space for feature data at non-leaf nodes in a property sphere. Feature data are stored only at leaf nodes in a property sphere. In Sections 3.1 - 3.3 we describe three possible data structures for a property sphere. In Section 3.4 we compare the space

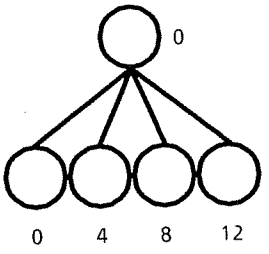

Figure 5: Quadtree corresponding to level 1 facet subdivision (M = 3).

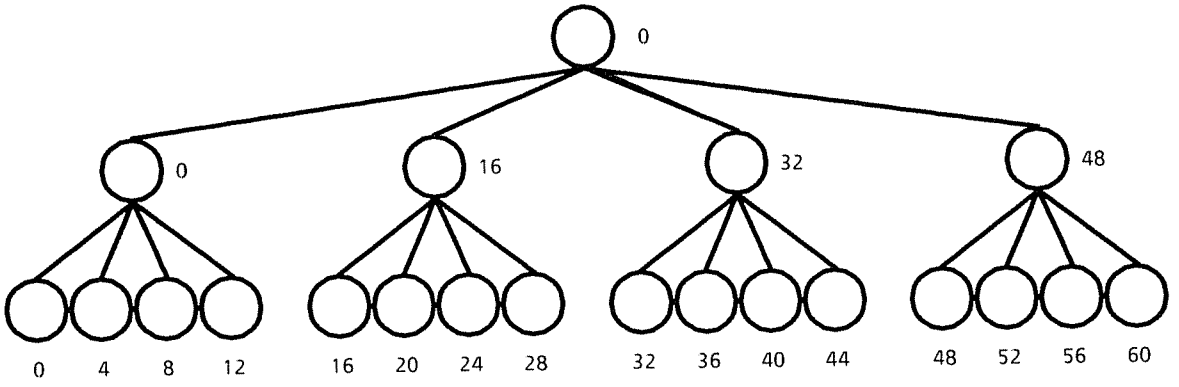

Figure 6: Quadtree corresponding to level 2 facet subdivision (M = 3).

efficiency of these three methods. The following sections rely on these definitions:

M = maximum subdivision level (from Section 2.2)

4^M = maximum number of viewpoints in each icosahedron face

= maximum number of leaf nodes in a single quadtree of the property sphere

$20 \cdot 4^M$ = maximum number of viewpoints on entire view sphere

= maximum number of leaf nodes in the property sphere

P = number of viewpoints that have been sampled

= number of leaf nodes in the property sphere

C = number of bits of feature data stored at each leaf node

A = number of bits in a data address (i.e. the number of bits in a pointer)

3.1. Pure Dynamic Method

A forest of 20 quaternary (4-ary) trees is built dynamically as the view sphere is sampled. If there are P leaf nodes in the forest, then there are $\frac{P-1}{3}$ non-leaf nodes [Knu68]. Each non-leaf node in the tree stores 5 pointers (to its parent and its 4 children), requiring $5A$ bits. For simplicity we assume that each leaf node uses the same $5A$ bits of space plus C bits for feature data.

Therefore, the total amount of space required to store a property sphere using this method is:

$5A \cdot \left(\frac{P-1}{3} \right) + (5A + C) \cdot P$ bits.

The parent-child relationships between viewpoints are made explicit by pointers stored in the nodes. The pure dynamic method is the only way to implement a property sphere if M is not known a priori.

3.2. Pure Static Method

The pure static method relies on the predetermination of M. A 2-dimensional array with 20 rows (one for each icosahedron face) and 4^M columns (the maximum number of viewpoints derived from each face) is statically preallocated. Each cell of the array is C bits wide. The array is indexed by the addressing scheme described in Section 3.2.1. If some of the 4^M viewpoints on each face are not sampled, then storage space will be wasted. The total amount of space required to store a property sphere using this method is: $20 \cdot 4^M \cdot C$ bits.

The parent-child relationships between viewpoints are implicit in the addressing scheme described in Section 3.2.1.

Static preallocation of storage space for feature data has been used by Fekete and Davis [Fek84] and Goad [Goa83], for example.

3.2.1. Addressing the Viewpoints

Every facet of the subdivided icosahedron can be assigned a unique address, derived from (a) the face of the icosahedron from which the facet originated, (b) the level of the facet, and (c) the number of the facet. Figure 7 shows an icosahedron that has been "unfolded" and "flattened" out. The 20 faces of the icosahedron are numbered 1 through 20. Any assignment of numbers to faces will do, however, the assignment shown in Figure 7 has some interesting properties described in Section 4.2.2.

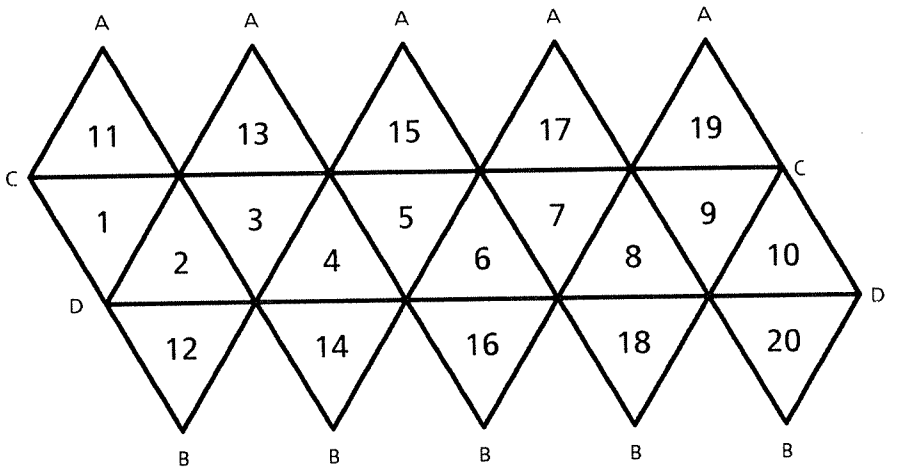

Figure 7: 20 Faces of Flattened Icosahedron. Identically lettered vertices correspond to the same point.

The level of the facet will be a number between 0 and M. A facet at level L ($0 \leq L \leq M$) arises from L subdivisions of an icosahedron face. Up to 4^M facets (and corresponding viewpoints) can be constructed on each icosahedron face. Facets on each face will be numbered 0 to $4^M - 1$. $20 \cdot 4^M$ facets will be constructed over the entire icosahedron. The following table shows the number of viewpoints generated for several values of M.

Levels of Subdivision	Number of Viewpoints per Icosahedron Face	Number of Viewpoints on entire Icosahedron
0	1	20
1	4	80
2	162	320
3	643	1280
4	256	5120
5	1024	20480
6	4096	81920

Of course, this subdivision need not be performed to the same level on all facets, so the numbers in the table are upper bounds on the number of viewpoints at each level.

For example, assume $M = 3$. By definition, the facet number of an original icosahedron face (at level 0) is 0 (see Figure 8). The facet numbers of the 4 children ($0 \leq i \leq 3$), are computed from

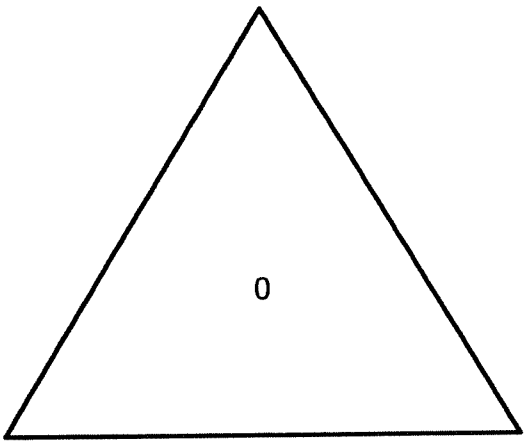

Figure 8: Facet 0 at level 0.

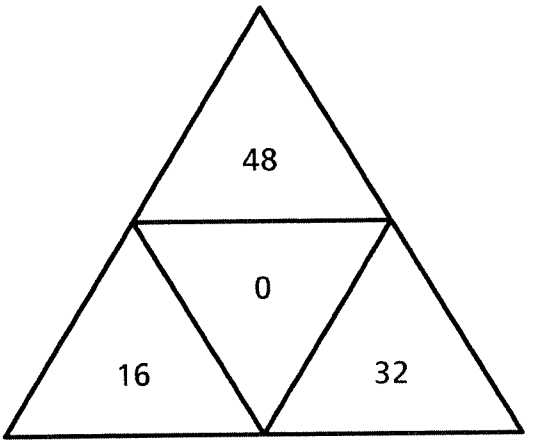

Figure 9: Facet numbers of level 1 facets (M = 3).

the facet number of the parent, the level of the parent ($0 \leq parent.level < M$), and M:

$$child[i].number = parent.number + i \cdot 4^{M - parent.level - 1}$$

$$child[i].level = parent.level + 1$$

$$child[i].face = parent.face$$

Conversely, the facet number of a parent can be computed from the facet number of a child, the level of the child ($0 < child.level \leq M$), and M:

$$parent.number = \mathbf{int}\left(\frac{child.number}{4^{M - child.level + 1}}\right) \cdot 4^{M - child.level + 1}$$

$$parent.level = child.level - 1$$

$$parent.face = child.face$$

The facet numbers of the 4 children of facet 0 at level 0 are 0, 16, 32, and 48 at level 1 (see Figure 9). Figure 10 shows the facet numbers of the level 2 facets. Figure 11 shows the facet numbers of all facets obtained by subdividing an icosahedron face three times.

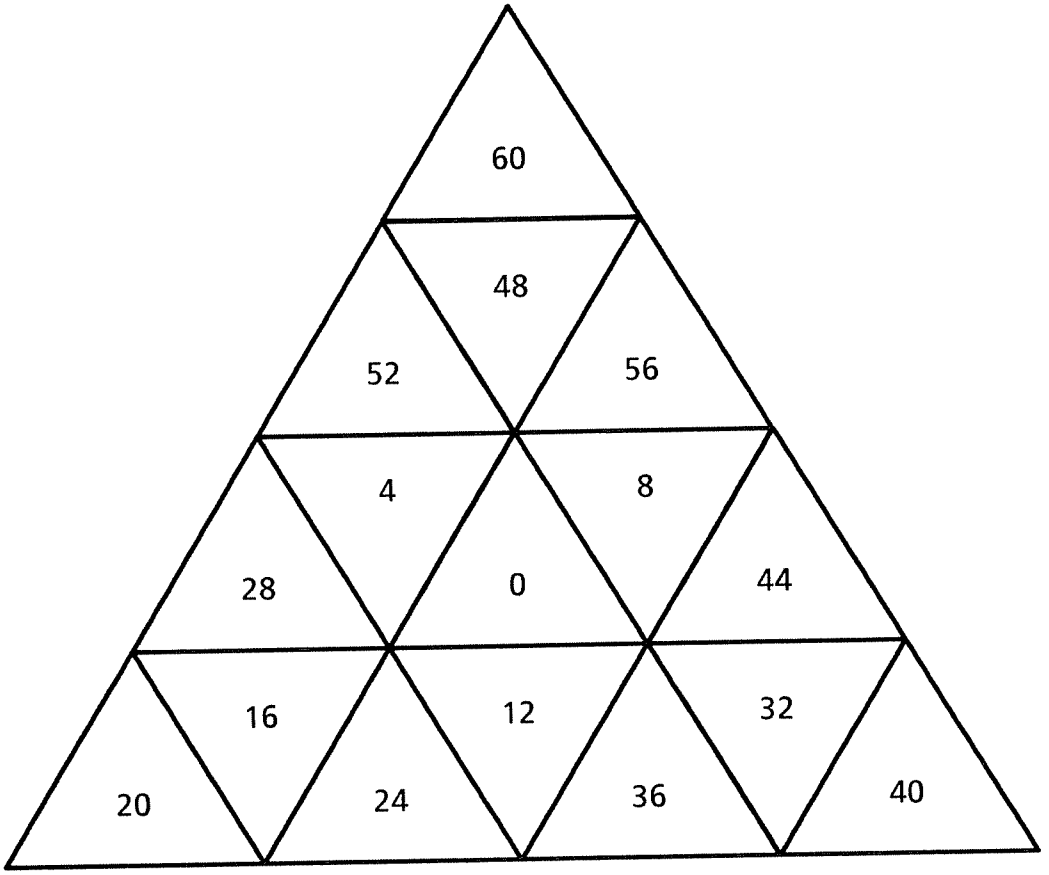

Figure 10: Facet numbers of level 2 facets (M = 3).

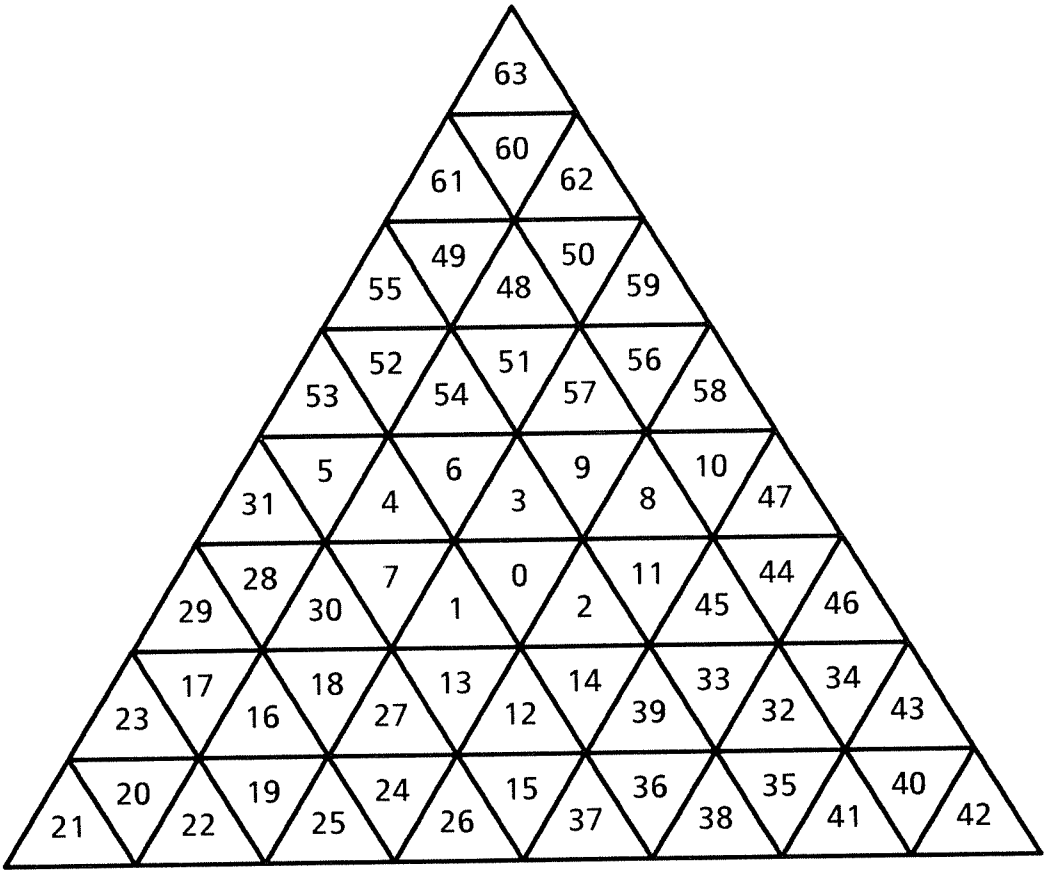

Figure 11: Facet numbers of level 3 facets (M = 3).

In Section 2.2 we pointed out that the viewpoints corresponding to a parent, and its 0 child are the same. Notice that the facet numbering preserves this correspondence - a facet and its 0 child will both have the same facet number. This is important because data associated with the viewpoint will be accessed using this address.

Fekete and Davis [Fek84] present a similar scheme for computing facet addresses. Their method does not take advantage of the fact that the maximum subdivision depth is known a priori. Their addresses are strings of length proportional to the current subdivision level; address strings must be converted to and from integers for use as array indices.

3.3. Hybrid Static/Dynamic Method

When some of the 4^M viewpoints on each face are not sampled, the pure static method wastes storage space. The hybrid static/dynamic method tries to save space by only allocating A bits (instead of C bits) in each cell of the 2-dimensional array with 20 rows and 4^M columns. The A bits will be used for a pointer to dynamically allocated blocks of C bits. The C bit blocks will only be allocated for the P viewpoints that are actually sampled. If $A << C$ then the hybrid static/dynamic method will require less space than the pure static method. The total amount of space required to store a property sphere using this method is: $20 \cdot 4^M \cdot A + P \cdot C$ bits.

The parent-child relationships between the viewpoints are implicit in the addressing scheme described in Section 3.2.1.

3.4. Storage Considerations of Property Sphere Implementaion

In this section, the three property sphere implementation methods (pure static, pure dynamic, hybrid static/dynamic) are compared to determine which method provides the most space efficient data representation. The most space efficient method can be chosen based on the range of $\frac{P}{20 \cdot 4^M}$, the percent of viewpoints that will be sampled.

First, an expression is derived for the case when the pure static method should be used instead of the hybrid static/dynamic method:

$$20 \cdot 4^M \cdot C < 20 \cdot 4^M \cdot A + P \cdot C$$

$$20 \cdot 4^M \cdot (C - A) < P \cdot C$$

$$\frac{C - A}{C} < \frac{P}{20 \cdot 4^M}$$

$$1 - \frac{A}{C} < \frac{P}{20 \cdot 4^M}$$

Thus the pure static method requires less storage than the hybrid static/dynamic method when the percent of sampled viewpoints, $\frac{P}{20 \cdot 4^M}$, is greater than $1 - \frac{A}{C}$. Generally the number of bits in a pointer (A), will be very small compared to C, so the fraction $\frac{A}{C}$ will be very close to 0, and $1 - \frac{A}{C}$ will be very close to 1. The pure static method is the most space efficient representation when 100% (or nearly 100%, based on A and C) of the viewpoints will be sampled.

Next, an expression is derived for the case when the pure dynamic method should be used instead of the hybrid static/dynamic method:

$$5A \cdot \left(\frac{P - 1}{3} \right) + (5A + C) \cdot P < 20 \cdot 4^M \cdot A + PC$$

$$5A \cdot \left(\frac{P - 1}{3} \right) + 5AP + PC < 20 \cdot 4^M \cdot A + PC$$

$$5A \cdot \left(\frac{P - 1}{3} \right) + 5AP < 20 \cdot 4^M \cdot A$$

$$\frac{P - 1}{3} + P < 4 \cdot 4^M$$

$$4P - 1 < 3 \cdot 4^{M+1}$$

$$P - \frac{1}{4} < 3 \cdot 4^M$$

Since P, and M are integers, this inequality can be re-expressed as:

$$P < 3 \cdot 4^M$$

This inequality indicates that when P, the number of sampled viewpoints, is less than $3 \cdot 4^M$ then the dynamic method should be used instead of the hybrid static/dynamic Method. If both sides of the inequality are divided by $20 \cdot 4^M$ (the maximum number of leaf nodes in the property sphere):

$$\frac{P}{20 \cdot 4^M} < \frac{3 \cdot 4^M}{20 \cdot 4^M}$$

$$\frac{P}{20 \cdot 4^M} < \frac{3}{20}$$

In other words, when less than 15% of the maximum number of possible viewpoints are actually sampled, then the dynamic method should be used to implement a property sphere. Combining this result with the comparison of pure static verses hybrid static/dynamic produces the following table for values of $\frac{P}{20 \cdot 4^M}$:

Value of $\frac{P}{20 \cdot 4^M}$		
$< .15$	$\geq .15$ & $< 1 - \frac{A}{C}$	$\geq 1 - \frac{A}{C}$
Pure Dynamic	Hybrid Static/Dynamic	Pure Static

A and C are always known for a given application. The percent of viewpoints that will be sampled, $\frac{P}{20 \cdot 4^M}$ may not be known a priori for every application. If the maximum subdivision depth M is not known or cannot be approximated, then only the dynamic method can be used.

4. Property Sphere Operations

In this section we present several basic operations on property spheres. These are some of the operations that will be required when implementing an object recognition program based on a 3-D multiview object representation.

The operations will be described in an implementation independent form — the property sphere can be implemented using any of the three methods described in Section 3.

4.1. Finding Parents and Children

Parent and children finding is the most primitive of all property sphere operations. Neighbor finding (Section 4.2), coordinate computation (Section 4.3), and node enumeration (Section 4.4) operations all rely on the ability to quickly and easily determine the parents and children of a node.

When the property sphere is implemented using the dynamic method, parent-child relationships are made explicit by the pointers stored in the tree nodes. When the property sphere is implemented using the pure static or hybrid static/dynamic method, parents and children are calculated using the addressing scheme described in Section 3.2.1.

4.2. Finding Neighbors

Neighbor finding is a basic property sphere operation that returns a list of viewpoints derived from facets that share an edge with a given viewpoint's facet. This is the 3-D equivalent of neighbor finding techniques for images represented by quadtrees [Sam82]. One application of the neighbor finding operation is region growing, which is discussed in Section 4.5.

In Section 4.2.1 we describe a simple algorithm to find neighboring viewpoints that are contained in the same icosahedron face as the given node. In Section 4.2.2 we modify this algorithm to permit neighboring viewpoints to lie on different icosahedron faces. This algorithm will only find neighbors that are at the same level as the given viewpoint. In Section 4.2.3 we describe an algorithm that will find all neighboring viewpoints, regardless of level.

4.2.1. Finding Neighbors on a Single Icosahedron Face

The following table-lookup can be performed to find the neighbors of a viewpoint that lie on the same icosahedron face and are of the same size as the given face.

Type of child	1-Neighbor	2-Neighbor	3-Neighbor
0	child 1 of parent	child 2 of parent	child 3 of parent
1	child 3 of parent's 1-Neighbor	child 0 of parent	child 1 of parent's 3-Neighbor
2	child 0 of parent	child 3 of parent's 2-Neighbor	child 2 of parent's 3-Neighbor
3	child 2 of parent's 1-Neighbor	child 1 of parent's 2-Neighbor	child 0 of parent

This definition of neighbors is recursive. If the level of a node is L, then at most L of the node's ancestors will need to be visited (i.e. $L+1$ calls to the neighbor procedure will be made) to find all three neighbors of a node.

When a neighbor lies on an adjacent icosahedron face, this table-lookup will fail, because it will try to find the parent of a level 0 node, which is undefined.

4.2.2. Finding Neighbors Across Icosahedron Faces

When the neighbor finding algorithm needs to find the neighbor of a level 0 node, we simply use this table:

Face Number	Nghbr 1	Nghbr 2	Nghbr 3	Face Number	Nghbr 1	Nghbr 2	Nghbr 3
1	10	2	11	11	19	13	1
2	1	3	12	12	20	14	2
3	2	4	13	13	11	15	3
4	3	5	14	14	12	16	4
5	4	6	15	15	13	17	5
6	5	7	16	16	14	18	6
7	6	8	17	17	15	19	7
8	7	9	18	18	16	20	8
9	8	10	19	19	17	11	9
10	9	1	20	20	18	12	10

Figure 7 shows the corresponding numbering of icosahedron faces. This table is easily summarized by the following expressions based on F (the face number):

let $T = (F-1)$ **div** 10

The 1-neighbor of face F is face = $((F+8-T)$ **mod** $10)+(T\cdot 10+1)$

The 2-neighbor of face F is face = $((F+T)$ **mod** $10)+(T\cdot 10+1)$

The 3-neighbor of face F is face = $F-20\cdot T+10$

Example: The 1-neighbor of facet 0 at level 0 on icosahedron face 2 is facet 0 at level 0 on icosahedron face 1.

Based on the layout of neighbors shown in Figure 4, one can easily determine the neighbors of faces 1 - 10 by examining Figure 7. It is also easy to find neighbor 3 of faces 11 - 20 (simply choose the face above or below). To find the 1 and 2 neighbors of faces 11 - 20, one must make the observation that faces 11, 13, 15, 17, and 19 all wrap together at the top of the icosahedron (and faces 12,14,16,18, and 20 wrap at the bottom). When the faces are wrapped together, you can see that face 11 is the 1-neighbor of face 13, turned on its side. All faces on the ends (faces 11-20) must be treated as though they have been turned on their sides when calculating 1 and 2 neighbors. This observation leads to the correction:

if the initial viewpoint is on any of faces 11 - 20, **and** the 1 or 2 neighbor of the viewpoint is on a different face **then**

if the neighbor is a 1 or 2 child **then** make the neighbor the 3 child of its parent;
else if the neighbor is a 3 child **then**
if this is neighbor 1 of the viewpoint **then** make the neighbor the 2 child of its parent
else if this is neighbor 2 of the viewpoint **then** make the neighbor the 1 child of its parent

4.2.3. Finding Neighbors When All Viewpoints Are Not At The Same Level

The neighbor finding algorithm presented in Sections 4.2.1 and 4.2.2 finds only the three neighbors of a node that are at the same level as the given node. But the neighbor of a node at level L may not exist if the property sphere does not contain the maximum number of leaf nodes ($20 \cdot 4^M$). There are 3 distinct cases:

Case 1:	Node is at level L, and neighbor is at level L	(Figure 12)
Case 2:	Node is at level L, and neighbor is at level K, $K < L$	(Figure 13)
Case 3:	Node is at level L, and neighbors are at levels $> L$	(Figure 14)

When the neighbor of a node at level L is a node at level K, and K is less than L, then the table look-up presented in Section 4.2.1 will fail when trying to find the non-existent level L descendants of the level K node. The level K ancestor is simply returned as the neighbor.

When the neighbors of a node at level L are nodes at levels higher than L, then the table look-up presented in Section 4.2.1 returns a common ancestor of the neighboring leaf nodes. Case 3 is handled by enumerating the descendants of this ancestor according to the

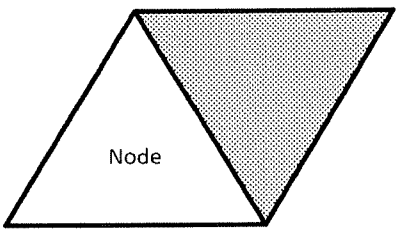

Figure 12: Node and neighbor at same level.

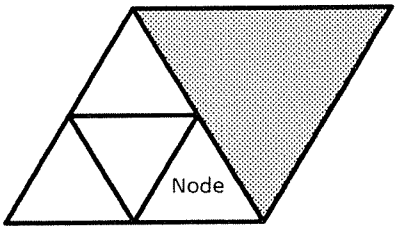

Figure 13: Node at level L, neighbor at level K, K<L.

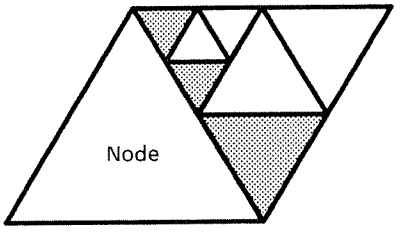

Figure 14: Node at level L, neighbors at levels >L.

following table:

Ancestor is:	Neighbors are:	
1-neighbor	all existing 2- and 3-children	of the ancestor
2-neighbor	all existing 1- and 3-children	and its
3-neighbor	all existing 1- and 2-children	descendants

4.3. Computing the Coordinates of a Viewpoint

It is often necessary to compute the Cartesian coordinates of a viewpoint. For example, to direct a CAD/CAM system to generate 2-D projections of an object, the 3-space coordinates of the camera positions must be computed. As another example, assume that properties of an object in an image match properties with one viewpoint in the property sphere. The coordinates of the viewpoint, along with the known orientation of the object, can be used to retrieve the orientation of the object in the image.

Coordinate computation is performed in two steps. First the Cartesian coordinates of the vertices of a facet are computed. Second, the coordinates of the center of the facet (the viewpoint), are computed from the coordinates of the vertices of the facet.

The coordinates of the three vertices of a facet at level L are computed from the coordinates of the vertices of the facet's ancestors at levels $L-1, L-2, \ldots, 0$. The ancestor at level 0 is an icosahedron face; The coordinates of the twelve icosahedron vertices are placed at $(0, \pm a, \pm b)$, $(\pm a, \pm b, 0)$, and $(\pm b, 0, \pm a)$ [Bal82] where:

$$golden\_ratio = \frac{1+\sqrt{5}}{2},$$

$$a = sphere\_radius \cdot \left(\frac{\sqrt{golden\ ratio}}{5^{\frac{1}{4}}} \right), \text{ and}$$

$$b = sphere\_radius \cdot \left(\frac{1}{\sqrt{golden\_ratio} \cdot 5^{\frac{1}{4}}} \right)$$

Given the array *icos_vertices* (*array*[1..12] *of coordinates*), initialized with values from $[0, \pm a, \pm b]$, and an icosahedron face number F, its three vertices are given by:

corner_1 = *icos_vertices*[((*F* + 7) **mod** 10) + 1]

corner_2 = *icos_vertices*[*F* − ((*F* − 1) **div** 10)·10]

corner_3 = **if** $F \leq 10$ **then** *icos_vertices*[((*F* + 8) **mod** 10) + 1]

else *icos_vertices*[12 − (*F* **mod** 2)]

Next, for every level K between 1 and L, the coordinates of the vertices of the ancestor at level K must be computed from the coordinates of the vertices of the ancestor at level $L-1$. These three coordinates are chosen from the set of six coordinates (the three coordinates of the ancestor at the previous level plus the midpoints between these three coordinates) depending on whether the ancestor at the current level is a 0, 1, 2, or 3 child. The coordinates of the vertices of each ancestor are *pushed out* to the radius of the view sphere by normalizing the directed vector that they lie on to be the length of the view sphere radius.

Once the coordinates of the vertices of the level L facet have been computed, the center of the facet is found by averaging together these three coordinates. The center point is also normalized so that it lies on the surface of the view sphere.

4.4. Enumerating Viewpoints

There are many applications where one needs to enumerate the viewpoints in the property sphere, e.g. when directing a CAD/CAM system to generate 2-D projections from each viewpoint; as part of a region growing operation (Section 4.5); when searching for a viewpoint with a particular associated property [Hor84, Ike 81, Ike82, Wal81, Cha82, Lie79, Fek84, Sil84].

Node enumeration corresponds to a traversal of the property sphere that visits every node. A breadth first traversal of the property sphere will enumerate the viewpoints in a hierarchical manner: level 0, then level 1, . . . , level M, as follows:

```
procedure Breadth_First_Property_Sphere_Traversal;

    for i := 1 to icosahedron_faces do begin
        node.number := 0;
        node.level := 0;
        node.face := i;
        push_fifo(node)
    end;

    search_depth := M;
    while not queue_empty do begin
        pop_fifo(node);
        visit(node);
        if node.level < search_depth then begin
            children := children_of(node);
            for j := 0 to 3 do push_fifo(children[j])
        end
    end;
```

To visit nodes in a depth-first ordering, replace the push_fifo, pop_fifo, and queue_empty functions with push_lifo, pop_lifo, and stack_empty functions, respectively.

4.5. Region Growing

Region growing is a property sphere operation that merges viewpoints into maximally connected regions such that all viewpoints of a region have the same value of a given property P. This operation is of course analogous to region growing in 2-D images [Bal82]. Region growing is a useful, basic operation to be performed on 3-D multiview object representations that leads to a more space and search-time efficient representation.

Region growing algorithms for 2-D images are usually based on grey level properties, textural properties, and gradient space properties. What properties should be used to segment property spheres? Koenderink and van Doorn [Koe76a, Koe76b, Koe79] proposed *stable-vantage points* (regions) where all views from that region exhibit the same topological properties. Chakravarty and Freeman [Cha82] proposed *vantage-point domains* (regions) where all views from that region contain the same vertices and edges. Any partitioning property must be one that can be derived from the 2-D viewer-centered representation of each viewpoint.

To partition the property sphere into regions based on the value of property P, it is first necessary to compute the value of property P associated with each viewpoint in the property sphere. Larger numbers of viewpoints (in the property sphere) will lead to more precise region boundaries.

The extension of 2-D region growing to property spheres is straightforward. Each leaf node of the property sphere is visited (using the algorithm in Section 4.4) and if the node is not already a member of some region, then label it the "seed" node for a new region r and propagate this label to all unlabeled neighboring viewpoints that have the *same* value of property P:

```
push(seed_node)
while node_stack < > empty do begin
    pop(node)
    if node is not a member of any region then
        if same features as the region r seed node then begin
        mark this node as a member of region r
            find all neighbors of this node
            for each neighbor, if it is not a member of any region then push(neighbor)
        end
end
```

To determine if two viewpoints are part of the same region, the values of their property P are compared. This algorithm compares property values exactly once for each pair of adjacent viewpoints. When there are $20 \cdot 4^M$ viewpoints in the property sphere (i.e. the property sphere is complete to level M), each viewpoint is adjacent to three other viewpoints. In this case $\frac{20 \cdot 4^M \cdot 3}{2} = 1.5 \cdot 20 \cdot 4^M$ property value comparisons will be made. When the property sphere is not complete, there are less than $20 \cdot 4^M$ viewpoints, and less than $1.5 \cdot 20 \cdot 4^M$ pairs of adjacent viewpoints, where M is the level of the lowest level viewpoint. Therefore, region growing can always be performed in time proportional to the maximum number of nodes in a complete property sphere.

If two non-connected regions both have the same value of partitioning property P, then the orientation of the object cannot be uniquely determined based on P alone.

4.6. Finding Viewpoints at a Distance D from a Given Viewpoint

An object recognition system based on property spheres might try to deduce the identity, or orientation of an object from multiple views of the object. Multiple views $V_1, \ldots, V_n$ of the object

are taken from camera positions $C_1, \ldots, C_n$. Features derived from view V_1 are matched against features stored in the property sphere; each node in the property sphere that matched view V_1 is added to a list L_m. Views $V_2, \ldots, V_n$ are used to verify or contradict each member of list L_m. Spatial constraints $S_2, \ldots, S_n$ are derived from the geometric spatial relationships between camera position C_1 and camera positions $C_2, \ldots, C_n$. That is, for each node N that is a member of L_m, compute a list L_p of all nodes in the property sphere that are related to node N by spatial constraint S_i $(2 \leq i \leq n)$. If there exists a member of L_p with features that match view V_i, then node N is confirmed as a candidate match for view V_1. With this stereo vision application as motivation, we discuss in this section the computation of list L_p given a node N and a spatial constraint S.

Figure 15 shows a pair of camera positions C_1 and C_2 separated by angle θ $(0 \leq \theta \leq \pi)$. If L_m is the list of viewpoint nodes that match the features associated with the view from C_1, then for each node N that is an element of L_m we want to compute the list L_p of nodes separated from N by angle

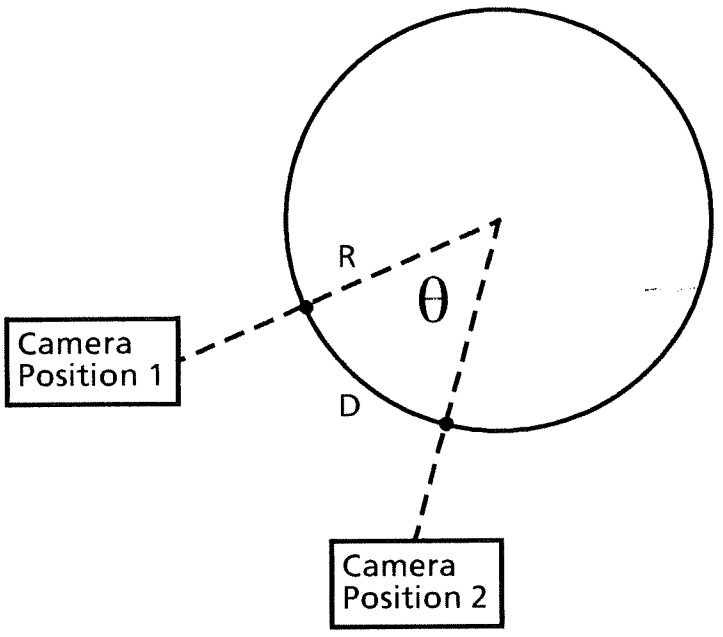

Figure 15: D = Rθ, for θ in radians.

θ. The spatial constraint S between N and the nodes in list L_p is that they are separated by angle θ. Since all nodes in the property sphere correspond to viewpoints taken from points that lie on the surface of the view sphere, spatial constraint S is equivalently stated to be the spherical distance corresponding to θ. Spherical distances between any two points on the surface of a sphere are measured along the great circle that connects them. So the spherical distance between two points separated by angle θ is $D = 2\pi R \cdot \frac{\theta}{2\pi} = R\theta$, where R is the radius of the view sphere.

To find all the nodes at distance D from node N, conceptually, we want to anchor a string of length D at the point on the view sphere corresponding to the center of node N (call this point N_c). The string is pulled tight on the surface of the view sphere and swept around in a circle. Each triangular patch T_p that the free end of the string passes through is considered to be at distance D from node N. The property sphere node corresponding to each T_p is added to list L_p.

One way to find all the T_p patches would be to check the triangular patch corresponding to each node in the property sphere (i.e the nodes in the property sphere are enumerated using the algorithms of Section 4.4). If the free end of the string of length D passes through triangular patch T_p, then the corresponding node in the property sphere is added to list L_p. In other words, if D is greater than or equal to the shortest distance between N_c and T_p, and D is less than or equal to the greatest distance between N_c and T_p, then the property sphere node corresponding to T_p is added to L_p.

The shortest distance between N_c and triangular patch T_p is the distance between N_c and the point T_p that is closest to N. The greatest distance between N_c and T_p is the distance between N_c and the point on T_p that is farthest from N. So the problem is reduced to finding the points on T_p that are closest and farthest from N_c. There are three cases, depending on the region point N_c lies in with respect to triangular patch T_p. See Figure 16.

To decide which region point N_c lies within, compute V_1, V_2, and V_3, the coordinates of the triangular patch T_p (see Section 4.3). Then compute the normal of the great circle GC_1 that passes through V_1 and V_2: $G\vec{C}_1 = \vec{V_1} \times \vec{V_2}$. Similarly: $G\vec{C}_2 = \vec{V_1} \times \vec{V_3}$; $G\vec{C}_3 = \vec{V_2} \times \vec{V_3}$. Next compute the

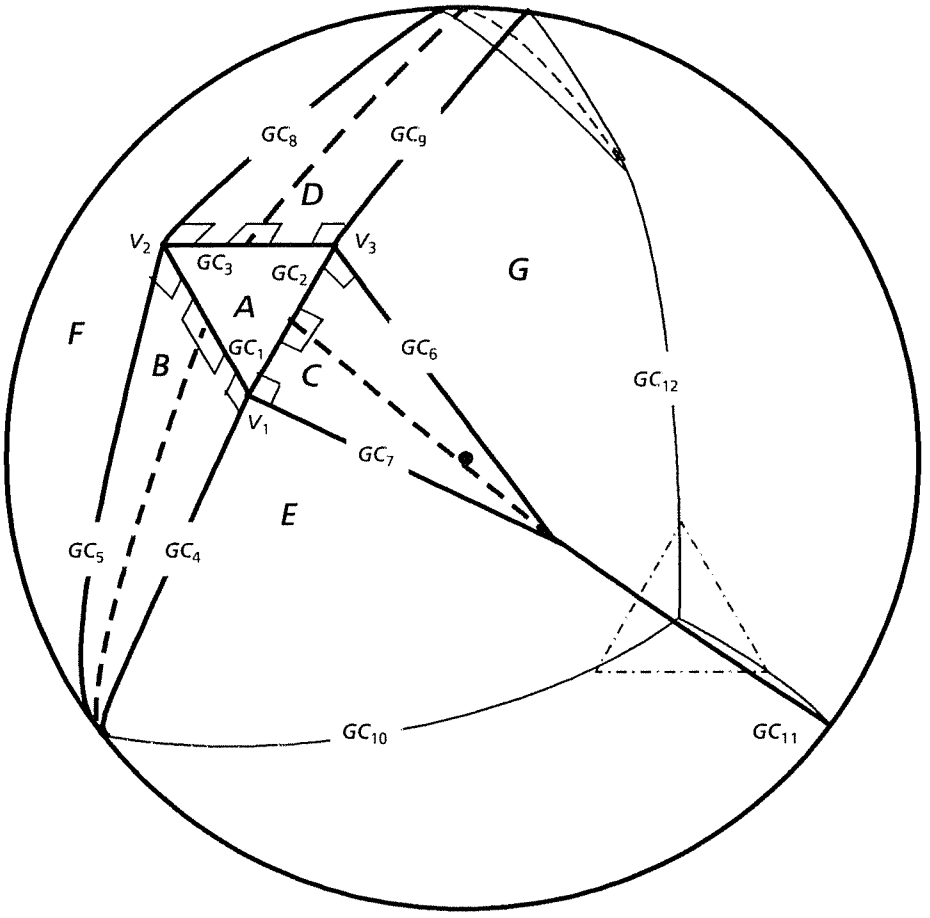

Figure 16: Triangular patch T_p (defined by V_1, V_2, V_3), its projection on the opposite side of a sphere (dot-dash), and great circles $GC_1,\ldots,GC_{12}$ which divide the sphere into regions *A*,*B*,*C*,*D*,*E*,*F* and *G*. Regions are bounded by solid lines. Bold lines are on the front surface of the sphere; thin lines are on the back surface of the sphere.

normal of the great circle GC_4 that passes through V_1 and is perpendicular to GC_1: $\vec{GC}_4 = \vec{V}_1 \times \vec{GC}_1$. Similarly: $\vec{GC}_5 = \vec{V}_2 \times \vec{GC}_1$; $\vec{GC}_6 = \vec{V}_3 \times \vec{GC}_2$; $\vec{GC}_7 = \vec{V}_1 \times \vec{GC}_2$; $\vec{GC}_8 = \vec{V}_2 \times \vec{GC}_3$; $\vec{GC}_9 = \vec{V}_3 \times \vec{GC}_3$. Finally, compute the normals of the great circles perpendicular to and passing through the midpoints of the sides of triangular patch T_p. The midpoint between $\vec{V}_1$ and $\vec{V}_2$ is computed by averaging $\vec{V}_1$ and $\vec{V}_2$. $\vec{M}_{12} = \frac{\vec{V}_1 + \vec{V}_2}{2}$; $\vec{M}_{13} = \frac{\vec{V}_1 + \vec{V}_3}{2}$; $\vec{M}_{23} = \frac{\vec{V}_2 + \vec{V}_3}{2}$; $\vec{GC}_{10} = \vec{M}_{12} \times \vec{GC}_1$; $\vec{GC}_{11} = \vec{M}_{13} \times \vec{GC}_2$; $\vec{GC}_{12} = \vec{M}_{23} \times \vec{GC}_3$.

Point N_c is within region A if it lies on the same side of GC_1 as V_3, **and** it lies on the same side of GC_2 as V_2, **and** it lies on the same side of GC_3 as V_1. (To determine if two points both lie on the same side of a great circle, compute: (a) the dot product of the first point and the normal of the great circle and (b) the dot product of the second point and the normal of the great circle. If the signs of the two dot products are equal, then the points both lie of the same side of the great circle.) Similarly, we can compute whether a point lies within regions B, C, D, E, F or G.

Case 1: Point N_c lies within region A. Since region A is the triangular patch T_p, the closest point on T_p is N_c itself.

Case 2: Point N_c lies within region B, C or D. The closest point on T_p is at the intersection of the perpendicular between N_c and the closest edge of T_p (GC_1, GC_2, or GC_3, respectively).

Case 3: Point N_c lies within region E, F or G. The closest point on T_p is V_1, V_2, or V_3, respectively.

To find the point on triangular patch T_p that is farthest from point N_c, project N_c to the opposite side of the view sphere; call this point N_c'. Then find the point on triangular patch T_p that is closest to N_c' using the same procedure described above. This point is the point on triangular patch T_p that is farthest from point N_c.

This algorithm requires testing the distance between each node N of L_m and each triangular patch T_p corresponding to a node in the property sphere. The time to compute list L_p for each node N is proportional to the number of nodes in the property sphere.

5. References

[Bal82] D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, N.J., 1982, 492-493.

[Bro81a] R.A. Brooks, Symbolic Reasoning Among 3-D Models and 2-D Images, *Artificial Intelligence, Vol. 17*, August 1981, 285-348.

[Bro81b] R.A. Brooks and T.O. Binford, Geometric Modeling in Vision for Manufacturing, *Proc. SPIE, Vol. 281*, 1981, 141-159.

[Bro84] P. Brou, Using the Gaussian Image to Find the Orietation of Objects, *International Journal of Robotics Research, Vol. 3, No. 4*, 1984, 89-125.

[Bro77] C.M. Brown, Representing the orientation of dendritic fields with geodesic tesselations, Technical Report TR-13, Department of Computer Science, University of Rochester, Rochester, NY, February 1977.

[Bro79a] C.M. Brown, Two descriptions and a two-sample test for 3-D vector data, Technical Report TR-49, Department of Computer Science, Universtiy of Rochester, Rochester, NY, February 1979.

[Bro79b] C.M. Brown, Fast Display of Well-Tesselated Surfaces, *Computers and Graphics, Vol. 4*, 1979, 77-85.

[Cha82] I. Chakravarty and H. Freeman, Characteristic Views as a Basis for Three-Dimensional Object Recognition, *Proc. SPIE, Vol. 336*, Robot Vision, 1982, 37-45.

[Cli71] J.D. Clinton, Advanced structural geometry studies, part I: polyhedral subdivision concepts for structural applications, NASA CR-1734/35, September 1971.

[Fek83] G. Fekete, Generating Silhouettes of Polyhedra, Technical Report CAR-TR-48, Center for Automation Research, University of Maryland, College Park, MD, December 1983

[Fek84] G. Fekete and L.S. Davis, Property Spheres: A New Representation for 3-D Object Recognition, *Proc. Workshop on Computer Vision: Representation and Control*, 1984, 192-201.

[Goa83] C.A. Goad, Special Purpose Automatic Programming for 3-D Model-Based Vision, *Proc. Image Understanding Workshop*, June 1983, 94-104.

[Hor79] B.K.P. Horn, Sequins and Quills, MIT A.I. Memo No. 536, May 1979.

[Hor83] B.K.P. Horn, Extended Gaussian Images, MIT A.I. Memo No. 740, July 1983.

[Hor84] B.K.P. Horn and K. Ikeuchi, The Mechanical Manipulation of Randomly Oriented Parts, *Scientific American, Vol. 251, No. 2*, August 1984, 100-111.

[Ike81] K. Ikeuchi, Recognition of 3-D Objects Using the Extended Gaussian Image, *Proc. IJCAI*, 1981, 595-600.

[Ike82] K. Ikeuchi and Y. Shirai, A Model Based Vision System for Recognition of Machine Parts, *Proc. AAAI*, 1982, 18-21.

[Knu68] D.E. Knuth, *The Art of Computer Programming, Vol 1: Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.

[Koe76a] J.K. Koenderink and A.J. van Doorn, The singularities of the visual mapping, *Biological Cybernetics, Vol. 24*, 1976, 51-59.

[Koe76b] J.K. Koenderink and A.J. van Doorn, Visual perception of rigidity of solid shape, *Journal of Mathematical Biology, Vol. 3*, 1976, 79-85.

[Koe79] J.K. Koenderink and A.J. van Doorn, The Internal Representation of Solid Shape with Respect to Vision, *Biological Cybernetics, Vol. 32*, 1979, 211-216.

[Lie79] L.I. Lieberman, Model-Driven Vision for Industrial Automation, in *Advances in Digital Image Processing*, P. Stucki, ed., Plenum, New York, 1979, 235-246.

[Mar78] D. Marr and K.H. Nishihara, Representation and Recognition of the Spatial Organization of Three-dimensional Shape, *Proc. of the Royal Society of London, Series B, Vol. 200*, 1978, 269-294.

[Sam82] H. Samet, Neighbor Finding Techniques for Images Represented by Quadtrees, *Computer Graphics and Image Processing, Vol. 18, No. 1*, 1982, 37-57.

[Sil84] T.M. Silberberg, L.S. Davis and D. Harwood, An Iterative Hough Procedure for Three-Dimensional Object Recognition, *Pattern Recognition, Vol. 17*, 1984, 621-629.

[Smi79] D.A. Smith, Using Enhanced Spherical Images for Object Representation, MIT A.I. Memo No. 530, May 1979.

[Wal81] T.P. Wallace, O.R. Mitchell, and K. Fukunga, Three-Dimensional Shape Analysis Using Local Shape Descriptors, *IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 3*, 1981, 310-323.