Chapter 5

Improving the Face Cluster Construction Algorithm

5.1. Introduction

In this chapter I explore ways to improve the performance and quality of the original face clustering algorithm presented in **Section 3.3**, and in Garland's dissertation [Garl99]. The face clustering algorithm is a vital part of the radiosity algorithm described in this thesis. For our goal of handling complex scenes on a standard workstation to be achieved, it is imperative that we be able to produce the face cluster hierarchies required by the radiosity algorithm on such a workstation. To achieve this goal, I have reimplemented Garland's original face clustering algorithm, and made a number of changes to improve both general quality and performance, as well as suitability for use with radiosity. The resulting implementation seems similar in speed to the "oconv" octree builder of Radiance [Warda].

I will begin by discussing the various improvements that can be made to the algorithm, present some performance results, and conclude by discussing some other work on clustering strategies for polygonal models.

5.2. Speed

5.2.1. Evaluating Running Time

We start by evaluating the running time of Garland's cluster algorithm. We split this time into two operations, both of which can be carried out independently; construction of the actual hierarchy, and calculation of bounding box extents. In the following sections we shall show how these results can be improved.

Face Cluster Hierarchy Construction

The results of profiling the face cluster algorithm described in **Chapter 3** for a large, 100,000 polygon model, are shown in **Table 5**. Only four routines contribute significant computational cost; all the others contribute less than 1% each to the overall running time.

Time	Pctg.	Routine	Description
9.2s	55%	Jacobi	Find eigenvectors for PCA
4.8s	30%	FindCost	Calculate cost term for a dual edge
1s	6%	MergeEdgeLists	Merge the dual edge lists of two clusters
0.6s	4%	InitFromFace	Create initial leaf cluster
0.7s	5%	-	All other routines
16.3s	**100%**		

Table 5: Profiling Garland's face cluster algorithm

By far the most time spent in the algorithm is on the Jacobi eigensolver routine used to calculate the principle components of each cluster [Pres92]. This routine, the simplest possible for the problem, uses Jacobi transformations to find the eigenvectors of the covariance matrix of the cluster. (As discussed earlier, these components are used both to estimate the flatness of the cluster, and to provide an orientation for the bounding box of the cluster.) Obviously this routine is a natural candidate for optimization.

Bounding Box and Side Area Calculation

After the cluster hierarchy has been constructed, we must also calculate the actual extents of the bounding boxes corresponding to each node in the hierarchy. For each cluster, a simple hierarchy traversal is performed to find the corresponding leaf polygons, and the vertices of these polygons used to calculate its extents. This

is an $O(hn)$ operation in time, where h is the height of the hierarchy, and n is the number of cluster nodes. (h is $O(\log n)$ when the hierarchy is balanced, and $O(n^2)$ in the worst case.)

For use with vector radiosity, we must also calculate the projected area of the cluster in each direction corresponding to a face of the box. This can be done at the same time as the iterative bounding operation, and adds an overhead of approximately 30% over just bounding box calculation.

The times taken to calculate the bounding boxes for two cluster hierarchies are shown in **Table 6**. Also shown is the actual height of the hierarchies, and the height of each hierarchy if it were balanced.

Model	Bounding Box Time	Cluster Nodes	Tree Height	Balanced Height
Dragon	17.1s	102,264	342	17
Whale	41.44s	101,812	506	17

Table 6: Garland's hierarchy: results for bounding box and side area calculation for two models.

At first glance, it seems as if the bounding box calculation is a much more significant part than the actual hierarchy calculation of the overall time cost of creating a cluster hierarchy suitable for radiosity. These times show the bounding box calculations taking over twice as long as cluster creation in the case of the whale model. However, this need not be the case. The hierarchies produced by the original clustering algorithm can be extremely unbalanced; in the case of the results above, by balancing the hierarchies we could gain potential speed-ups of 20 and 30 times respectively. This would also have a positive effect on any query-based algorithm that might use face clusters, such as ray-tracing or collision detection, as these also have $O(h)$ complexity.

If the poor balance of these hierarchies were solely a result of the need to constrain clusters to be flat, there is not much we could do to improve the situation without seriously diluting the effect of our cost metric. Luckily, this is not the case. Much of the imbalance is due to highly tessellated flat regions in each model, where the cost metric approaches zero, and provides little guidance as to which clustering operation to perform first. In such situations the heap-based algorithm clusters faces in order, often in the most unbalanced way possible. (This is the reason the much smoother whale model is less balanced than the bumpier dragon model; see **Figure 79** for both models.) As we shall see later, we can

exploit this by altering our cost metric to produce more balanced hierarchies, and reduce drastically the time taken to calculate bounding boxes.

5.2.2. Speeding up Principle Component Analysis

A first approach to eliminating the bottleneck of principle component analysis in the cluster creation algorithm is speeding up the eigensolver at its heart. There are a number of more sophisticated algorithms for the eigenvalue/eigenvector problem than the Jacobi eigensolver, which was originally chosen for simplicity of implementation. Techniques based on QR iteration with shifting work well for medium size matrices, and for large matrices, divide-and-conquer works best [Demm97, Golu89].

Unfortunately, these algorithms are more efficient the larger the problem size, and our problem size is extremely small; 3×3 matrices. For such a small problem size, it turns out that brute-force Jacobi iteration is as fast as we can reasonably expect, and has the advantage of being more stable than the other routines [Demm97]. This is illustrated in **Table 7**, which shows the average performance of several eigensolvers over a number of random box-shaped covariance matrices on a Pentium II-class processor. Jacobi takes 60% of the time of the QR shift-based algorithm, where both are hard-coded for 3×3 matrices, and both are an order of magnitude faster than a general SVD routine used for the same purpose.

Clocks	**Routine**
218,347	Generalised SVD
14,102	QR-Shift
8,395	Jacobi

Table 7: Average performance for various eigenvalue solvers.

As a result, it appears we cannot reasonably expect to speed up the eigensolver routine itself.

5.2.3. Eliminating PCA for Flat Clusters

An alternative to reducing the time taken by the PCA algorithm is to instead reduce the number of such computations needed. The PCA algorithm has two uses in the face cluster algorithm; determining the best fit plane of the cluster, and

determining directions for the bounding box. We will examine its application to the bounding box further in **Section 5.3**.

An alternative to calculating the best fit plane of the cluster is to instead use the sum-area normal, $\mathbf{S}$, to determine the orientation of the cluster, and use that in our error metric. This has two advantages. Firstly, it is much easier to calculate, and secondly, it is much more stable than the PCA routine for calculating orientations (see **Figure 67**).

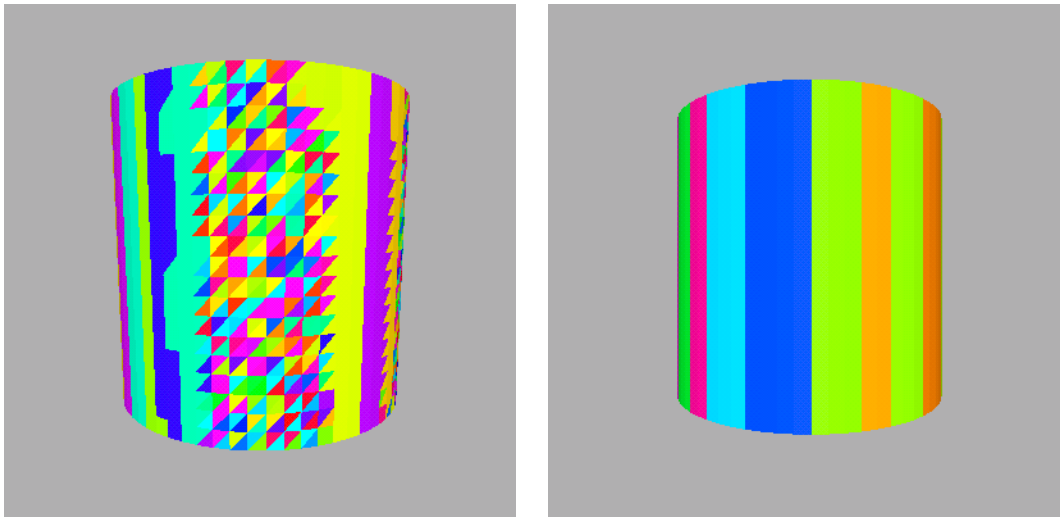

Figure 67: Determining cluster orientation. On the left is a cylinder clustered using PCA to determine the orientation of clusters. On the right are clusters created by using the sum area normal for this purpose. When clusters are reasonably flat, as in this case, doing so leads to more stable results than using PCA.

The disadvantage of using the sum-area normal is that, as clusters become less flat, the quality of the approximation to the best-fit plane falls off, and the estimate of the flatness of the cluster suffers. However, for such clusters we can revert to using the PCA algorithm. A good metric for deciding when to do so is to compare the length of the sum-area normal vector, namely, the projected area of the cluster in that direction, against the total area. This approaches one as the cluster

becomes completely flat, and zero as the cluster's surface becomes closed. I have had good results with using

$$\frac{\|\mathbf{S}\|}{A} > 0.5 \tag{78}$$

as a test for whether to use **S** instead of the principle component to determine the cluster's orientation. The impact on running time is shown in **Table 8**. The cost of the PCA analysis is much reduced, and in general the performance of the algorithm is at least twice as fast.

Time	Pctg.	Routine	Description
4.5s	58%	FindCost	Calculate cost term for a dual edge
1s	14%	Jacobi	Find eigenvectors for PCA
1s	11%	MergeEdgeLists	Merge the dual edge lists of two clusters
0.6	7%	InitFromFace	Create initial leaf cluster
1s	10%	-	All other routines
7.8s	**100%**		

Table 8: Profiling the new face cluster algorithm

5.2.4. The Impact of Balance on Running Time

At first glance, it would appear that the balance of the hierarchy would not have an effect on the running time of the clustering algorithm. As described by Garland, the clustering and edge-collapse algorithms have a complexity of $O(n\log n)$; there are $O(n)$ merge operations, and each such operation requires updating the heap, an $O(\log n)$ operation [Garl99].

However, on closer examination I have found that balance can have a considerable impact on the running time by affecting the *degree* of clusters in the model. Each face cluster node is connected to its immediate neighbours by dual edges, representing potential clusterings. The degree of a cluster node corresponds to the number of these dual edges. Whenever two clusters are merged, we must merge their dual edge lists (a constant time operation), and visit all of the edges in the new list, updating their corresponding costs in the heap. So the cost of a merge operation is $O(d\log n)$, where d is the number of dual edges in the new cluster.

If the algorithm is producing a balanced hierarchy, the number of dual edges belonging to any given cluster is approximately equal, and constant. Thus

d can be assumed to be constant, and Garland's analysis of $O(\log n)$ for a merge operation holds. In the worst case, however, polygons are iteratively collected into a single large cluster, and this assumption is no longer true. **Figure 68** shows this effect. As a consequence, performance suffers. In fact, for a planar regular grid, the accumulation cluster can have $O(\sqrt{n})$ dual edges incident on it in the worst case. As a result, merge operations are $O(\sqrt{n}\log n)$, and the algorithm as a whole is $O(n^{3/2}\log n)$ rather than $O(n\log n)$. We can expect similar worst-case behaviour for manifold surfaces.

The impact of this on clustering time is illustrated in **Figure 69**, which shows the time taken to cluster flat triangular meshes of varying sizes. (An example mesh is shown in **Figure 70**.) The results for both a constant-cost clustering metric, which results in an almost completely unbalanced hierarchy, and an area-based metric, which results in an almost completely balanced hierarchy, are shown. The reason for the differing performance is clearly illustrated in **Figure 71**, which shows the respective maximum dual edge adjacencies for the two metrics.

(I speculate that this is also the reason for the running-time jump for the Turbine model in figure 6.4 of Garland's dissertation [Garl99]. The turbine contains a number of highly tessellated flat surfaces; when the algorithm starts to simplify them, it seems probable that this will lead to vertices of extremely high degree. I have observed similar effects on smaller flat models.)

This result provides further motivation for altering the cluster algorithm to produce more balanced hierarchies. We will discuss how this can be accomplished in **Section 5.4.1**.

5.3. Calculating Oriented Bounding Boxes

In Garland's original method, the orientation of the bounding box is calculated by performing principal component analysis, and the extents of the box by iterating over each point contained in each cluster node. We shall briefly examine both of these approaches.

5.3.1. The Need for Robust Extent Calculations

Iterating over every face in a cluster for each cluster in order to calculate the extents of its bounding box is costly. In the best case (the hierarchy is balanced) it is $O(n\log n)$; in the worst, $O(n^2)$. A much cheaper approach that has been used elsewhere would be to calculate the extents from just the vertices that define the bounding boxes of its two child clusters, rather than every point in the cluster. This reduces bounding time to a constant per cluster, and overall time to $O(n)$.

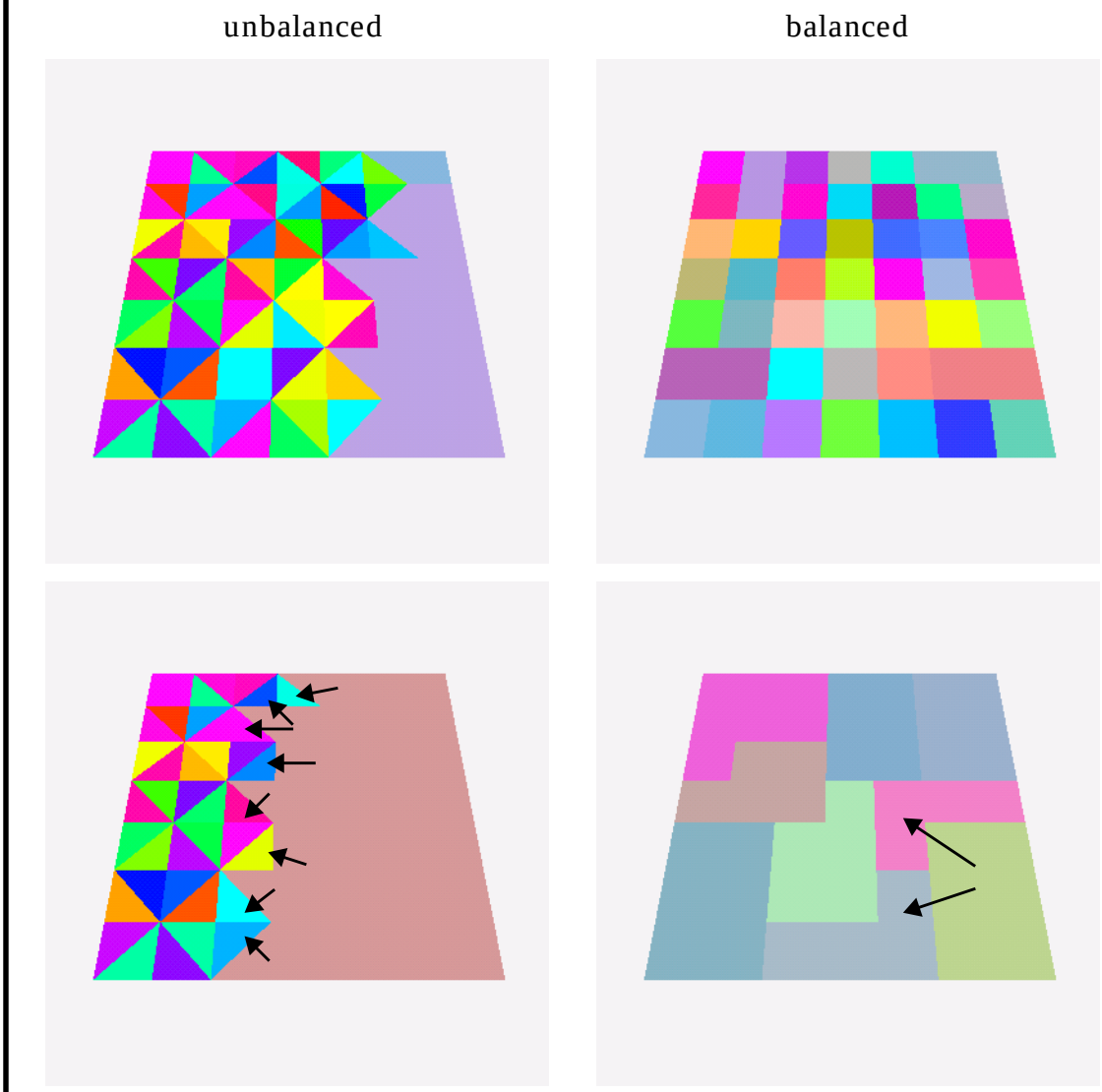

Figure 68: Hierarchy balance affecting dual-edge degree. On the left, in an unbalanced hierarchy, one large cluster slowly absorbs all other clusters (from top to bottom.) On the right, a balanced hierarchy leads to roughly similar-sized clusters as merging proceeds from smaller clusters (top) to larger ones (bottom).

A balanced hierarchy leads to a small, approximately constant number of dual edges per cluster (arrows, bottom-right), while in an unbalanced one, a single large cluster is adjacent to many small ones, leading to a non-constant dual edge count (arrows, bottom-left).

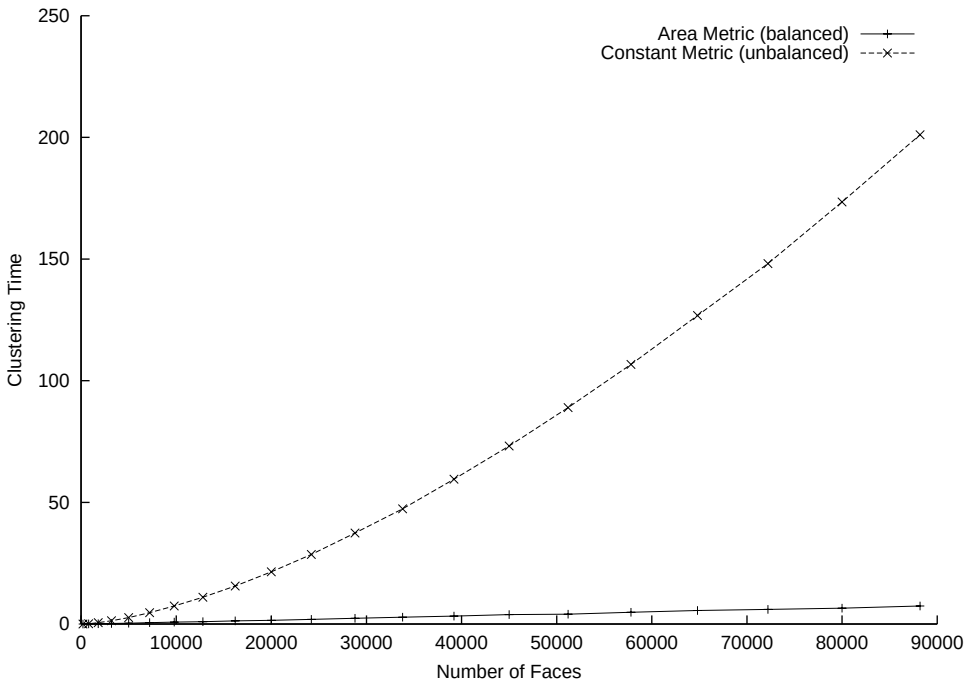

Figure 69: The effect of hierarchy balance on clustering time. An unbalanced hierarchy can make clustering very slow.

Unfortunately, in such an algorithm a little bit of error is potentially introduced in each bounding operation, typically making the parent cluster larger than it really has to be. Worse, the error accumulates up the tree, so that nodes near the root have very sloppy bounds. This effect is especially bad for the large models (and consequently large hierarchies) we must deal with. As the radiosity algorithm is most sensitive to error at the top of the hierarchy, due to its top-down refinement approach, any error, and especially any cumulative error, is unacceptable.

Any approximation algorithm we do choose to speed up bounding box calculation must be carefully designed to avoid cumulative approximation error.

5.3.2. Picking Bounding Box Orientations

A major problem for the use of face cluster hierarchies in radiosity simulations is that principle component analysis can fail to produce a reasonable bounding box

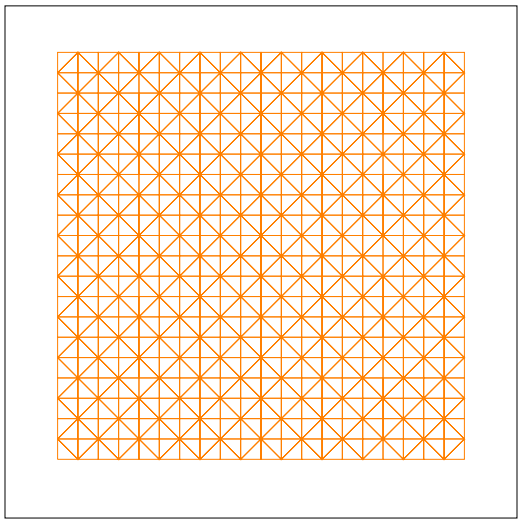

Figure 70: An example flat mesh of 800 faces.

in degenerate cases. While it handles objects with three differently-sized major axes well, if any two of these axes are close to equal, the algorithm becomes unstable. Worse yet, these are cases that occur commonly in radiosity scenes.

The problem with the principle component approach is that it is in essence fitting an ellipsoid to a cloud of points, and then taking that ellipsoid's principal axes as those of the oriented bounding box for the cluster. The worse case for this approach is a cube, where there is no preferred direction at all for the fit ellipsoid (the best fit being a sphere), resulting in an essentially random orientation for the bounding box. Because the algorithm used to calculate the eigenvectors of the covariance matrix is iterative, it is difficult to predict what this orientation will be.

While cubical objects don't occur often in practice, their two-dimensional equivalent, meshes in a rectangular shape, do. This leads to poor bounding boxes, as in **Figure 72**.

In some applications this is not critical. However, in radiosity the quality of our transfer approximations, and in particular the quality of our visibility calculations, is highly dependent on the tightness of the bounding boxes. Consider a floor in a square room; if the bounding box for the entire floor is oriented at 45 degrees to the floor, much of it will lie outside the walls of the room. Visibility tests against this bounding box will show the floor is partially occluded, even

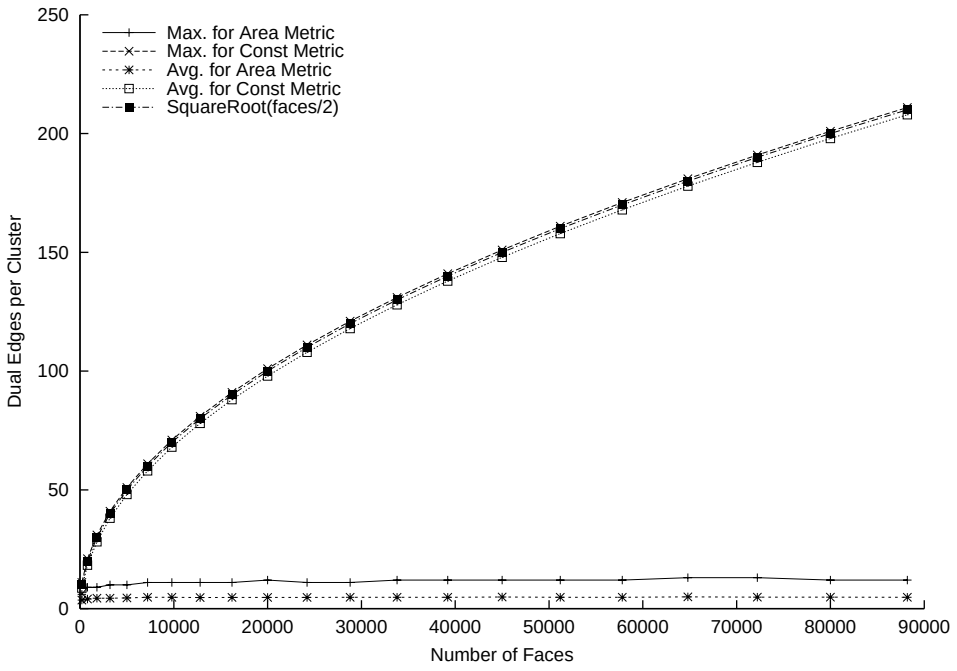

Figure 71: Dual edge degree increases when creating unbalanced hierarchies.

though in reality it is not occluded at all. Such problems lead to poor refinement decisions and unnecessary work on the part of the radiosity algorithm.

It is possible to calculate the optimal minimal-volume bounding box for a cloud of points in 3D, but this takes $O(n^3)$ time [ORo85']. (Principal component analysis is $O(n)$, and of course, we get it largely for free due to needing it also for our cost metric calculations.) This is far too expensive for our situation and geometry size.

A hybrid approach is to use PCA to establish a single principal axis, and then, after projecting all vertices onto a plane perpendicular to that axis, use a simpler algorithm to find the minimum-area enclosing rectangle (MER). The cost of finding the MER is $O(n\log n)$ for finding the two-dimensional convex hull, and $O(n)$ for finding the rectangle using a rotating callipers-type algorithm [Tous83]. (Briefly; it can be shown that a minimum-area enclosing rectangle must have one side coincident with an edge of the hull of the point set being enclosed. We can thus iterate through all such rectangles in linear time, and pick the one with the

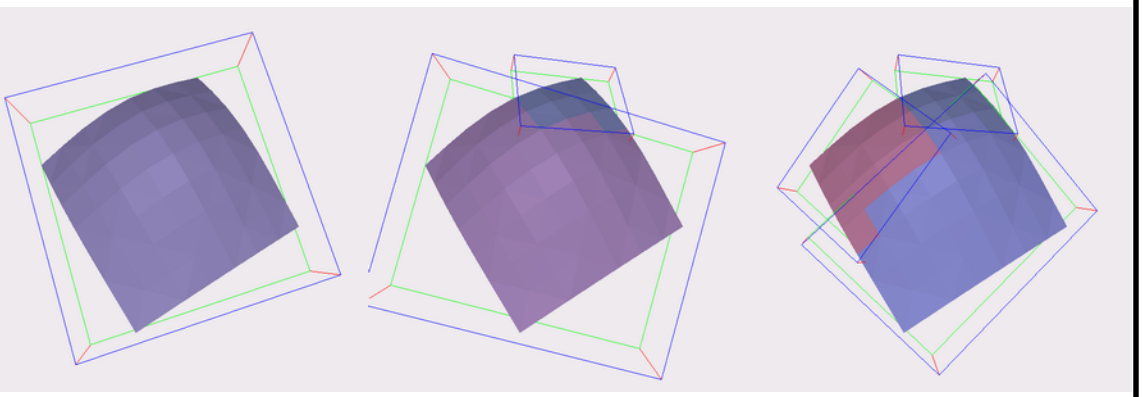

Figure 72: Badly oriented bounding boxes. From left to right we see the first three levels of subdivision of a square but curved surface. Because the clusters are close to symmetrically shaped, their bounding boxes have essentially random directions in the plane.

smallest area.) In practice we can merge hulls as we ascend the hierarchy, rather than creating a new one each time; the cost of merging two hulls is $O(n)$.

This approach works particularly well for flat clusters. For larger, non-flat clusters, the smallest-length component is no longer the most stable orientation feature of the cluster, and just using PCA works as well or better than this hybrid approach. Luckily, this meshes well with the approach to eliminating the need for PCA on flat clusters described in **Section 5.2.3**. For clusters meeting the condition of **Equation 78**, the **S** vector is used both to define a fit plane for the cluster, and to define the "flattest" axis of the bounding box, with the minimal-area enclosing rectangle algorithm used to find the remaining two axes. For all other clusters, PCA is used both for the fit plane, and to define the orientation of the bounding box. This approach works well, especially because it limits application of the MER algorithm to those clusters that genuinely need it for stability, and saves on the expense of PCA at the same time.

5.4. The Cost Metric

Garland's original cost metric, as outlined in **Equation 12**, sums three penalty terms to form the total cost of merging two clusters. These per-dual edge costs are then used to decide which two clusters to merge next.

This formulation suffers from a couple of problems. The first is that, because the cost terms are added, they can wind up interfering with each other in some situations. For instance, the original face cluster radiosity paper used weights on the three terms of $w_{fit} = 1.0$, $w_{dir} = 20.0$, and $w_{shape} = 1.0$ for some models, in an attempt to get the directional term to have the desired effect [Will99]. On further investigation, it turned out that the directional and shape costs interfered with each other—turning on the shape metric pretty much destroyed the effect of the direction metric.

Another problem is that the three cost terms have different units. E_{fit} is the variance of the cluster's points in one dimension, so it has units of area. Scaling the model by a factor s will lead to scaling of all associated fit costs by s^2. The E_{dir} term measures the variance of cluster normals from some average normal. This varies from 0, when all normals point in the same direction, to 1, when the normals are distributed randomly over the sphere of directions. Thus it is unitless. Finally, E_{shape} is also unitless, and varies from 0 (when two elongated clusters are merged into a fatter, more rounded one) to 1. Unlike the directional term, however, it tends to usually fall around 0.5, when no improvement to shape is made.

Thus, scaling the model can lead to different ratios of the cost terms, and thus different clustering behaviour, a situation we would like to avoid. We would also like to avoid the situation where the size of one term prevents the others from taking effect. To do this, I propose a modified cost metric:

$$E = E_{fit} \cdot (P_{dir} \cdot P_{shape}) \tag{79}$$

This metric works by taking E_{fit} as the base cost and scaling it according to the penalty terms P_{dir} and P_{shape}, which we will define to be 1 when both terms should have no effect, and to rise above 1 for clusters deemed “bad” by both terms.

5.4.1. Addressing Balance

As discussed previously, Garland’s original metric does not produce particularly balanced hierarchies. In the case of flat sections of a mesh, moreover, E_{fit} goes to zero everywhere, and we get extremely unbalanced hierarchies.

There is a simple fix to this problem. If we consider a flat mesh made up of evenly-sized polygons for the moment, a cost metric that would produce a balanced cluster hierarchy for such a mesh is just the surface area of the cluster. Such a metric ensures that clusters are merged evenly in order from small to large, producing a balanced hierarchy.

Of course, this approach only works if all leaf elements are approximately the same size. However, empirically this is the case for most detailed models. Also, where it is not the case, it usually suits the radiosity algorithm better to have approximately equal-area clusters, as this ensures that the depth of hierarchy that must be descended to reach a certain level of accuracy is relatively constant.

We must combine this fix for flat meshes with the usual E_{fit} metric in such a way that the changeover from curved surfaces, where the standard fit metric should predominate, to flat surfaces, where the area-based metric should predominate, is smooth. The approach I use is to combine them as follows:

$$E'_{fit} = E_{fit} + k \cdot area \tag{80}$$

Here the constant k controls the mix of the two metrics. As the variance of the points (roughly speaking, the height of the cluster squared) drops below k times its surface area (again, roughly, the cross-sectional area), balance of the hierarchy starts to take precedence over the flatness of clusters. Note that E_{fit} and $area$ both have units of distance squared. **Figure 73** shows the effects of this new fit term on hierarchy balance.

5.4.2. The Directional Term

The calculation of Garland's E_{dir} term is expensive, especially in terms of memory use, which it doubles. Also, its attempts to minimise variation in normals is not so well suited to our radiosity method. We are generally happy to have large amounts of micro-scale normal variation, as long as the surface is relatively flat, and does not fold over on the macro scale. A cluster can meet both of these conditions, and still be heavily penalised by Garland's variance-of-normals E_{dir} term, because of small bumps on the surface.

An alternative is to use the ratio of the projected area in the direction of the area-weighted normal to the total area of the cluster as a measure of how much it folds over. (This is just $\|\mathbf{S}\|/A$.) In the case of a completely flat cluster this is one; as the cluster folds over and the projected areas on each side of the fold cancel out, it drops, reaching zero in the case of a completely closed surface. We can thus define the penalty term of the previous section as

$$P_{dir} = \frac{area}{projArea}. \tag{81}$$

The major advantage of this is that, while it gives similar results to the original E_{dir}, as seen in **Figure 74**, there is no required additional storage cost, as we must

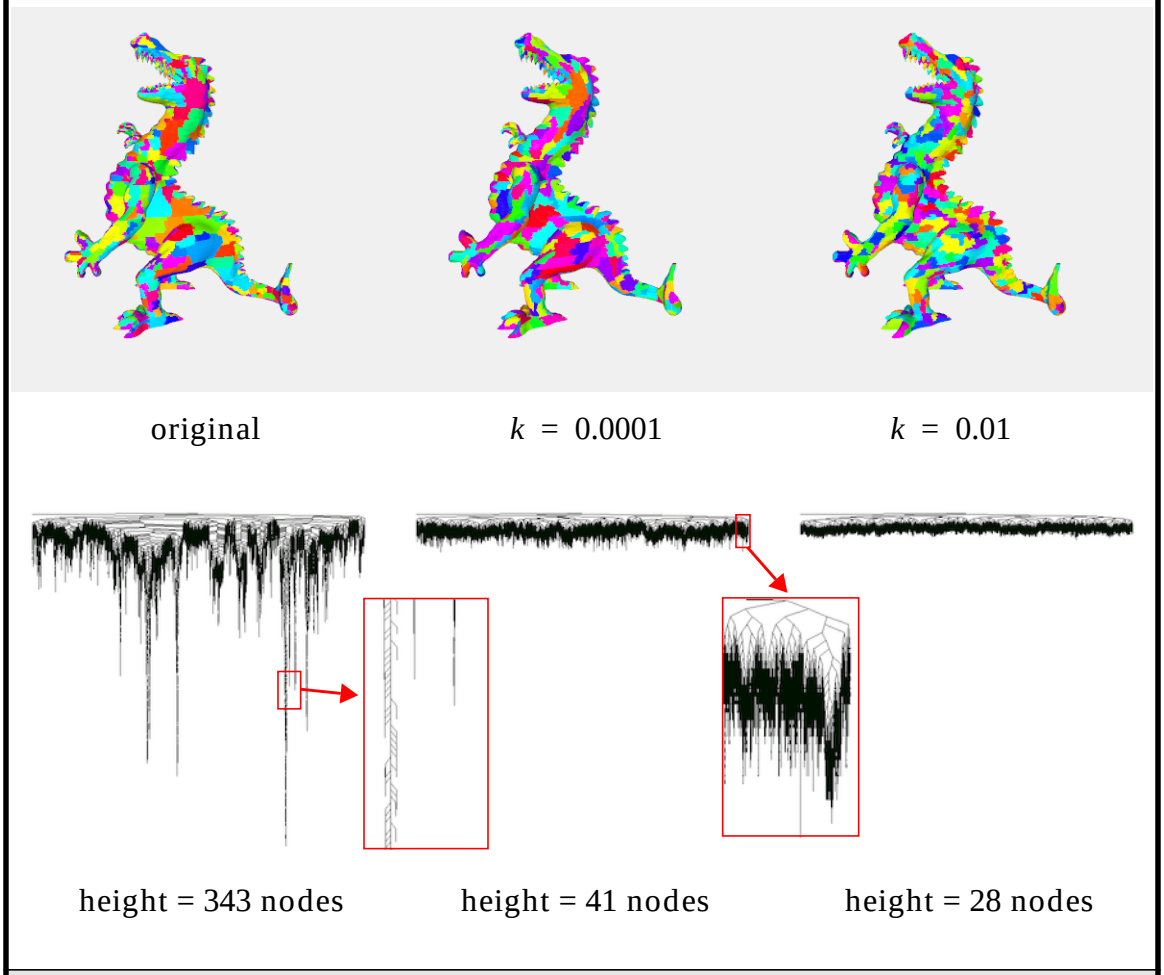

Figure 73: Enforcing hierarchy balance. Top: a dragon model clustered using, from left, the original cost metric, and the modified metric for two values of k. Bottom: the corresponding cluster hierarchies.

already track the area and sum area normal of each cluster. This can lead to space savings of almost 40% over Garland's original method, and can make the difference between being able to cluster the dragon model in under a minute on a 128MB laptop, and having to wait half an hour for results while it thrashes heavily.).

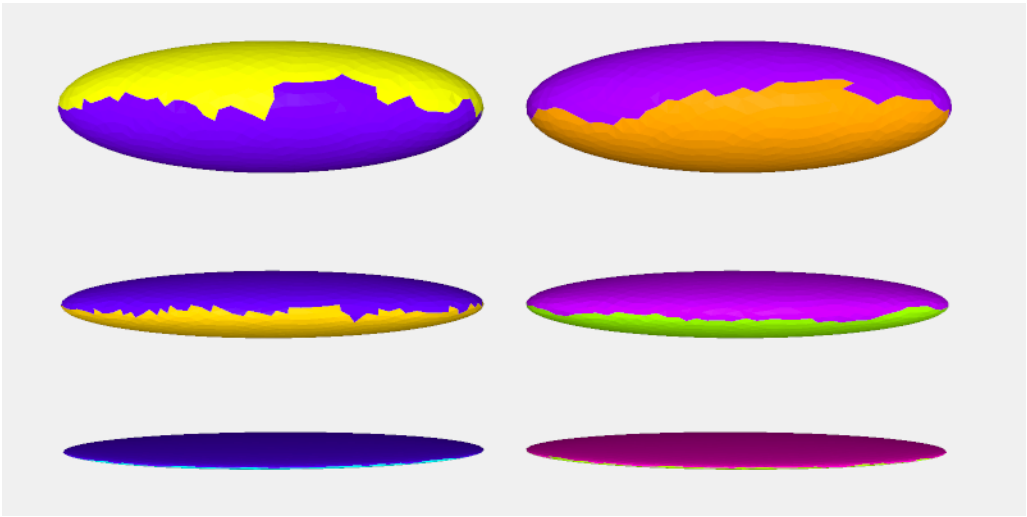

Figure 74: The directional term. Top to bottom: the first level of the cluster hierarchy for successively thinner ellipsoids. Left, the original E_{dir} metric based on normal quadrics. Right, the new metric based solely on projected area.

Ideally, we would like this first split in the hierarchy to divide the ellipsoid cleanly into top and bottom halves. For this particular example, we see that the new metric is no worse (and arguably a little better) than the original metric.

5.4.3. The Shape Term

In tests I found that the difference between using the γ of the cluster (the ratio of squared perimeter to area) and Garland's more complicated E_{shape} term seemed to be minimal; see **Figure 75**, for instance. The γ term works better with the scaling penalty approach outlined above; we can define the shape penalty simply as:

$$P_{shape} = \frac{perim^2}{4\pi \cdot area} \tag{82}$$

Again, this is one for a perfectly circular cluster, the most compact shape possible, and increases as the cluster's shape moves away from that ideal.

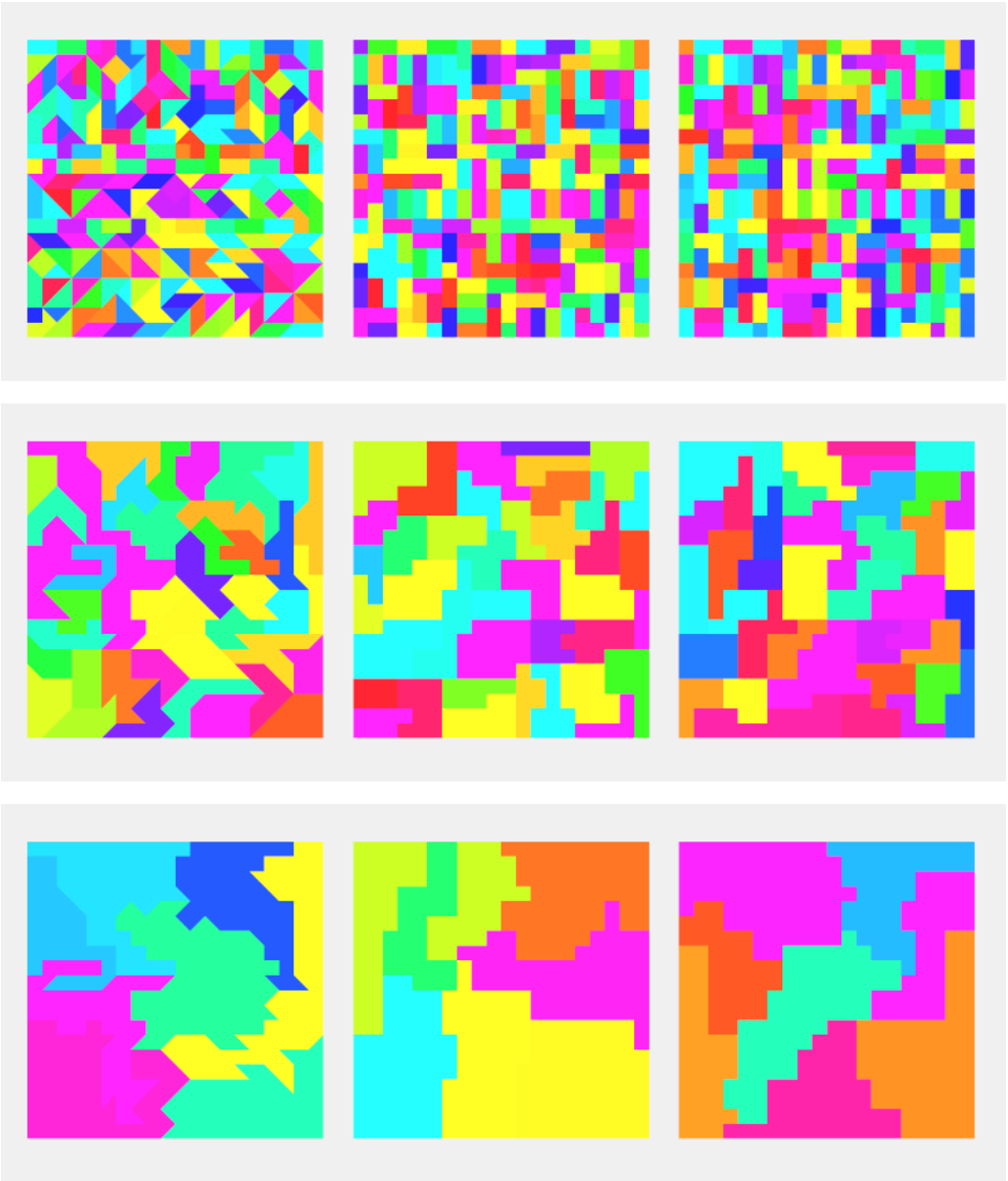

Figure 75: Effect of the shape term. Top to bottom: progressively coarser clusterings of three flat meshes. Left, no shape term; middle, the new term; right, Garland's E_{shape}.

5.4.4. A New Cost Metric

Putting it all together, we can write the complete cost term as:

$$(eFit + k \cdot area)\left(\frac{area}{projArea}\right)\left(\frac{perim^2}{4\pi area}\right) \tag{83}$$

which can be rewritten more simply as

$$\text{cost} = (eFit + k \cdot area)\frac{(perim)^2}{projArea} \tag{84}$$

where as usual, $projArea = \|\mathbf{S}\|$. (We can drop the 4π because the cost term is scale independent.) The k term controls the mix between balancing the hierarchy and obeying the $eFit$ metric; it should be set to some small value (I use 0.0001) for good results.

This metric has proven to be simple and robust, and much less sensitive to the effects of conflicting purposes for the metric. For instance, even if an area-weighted term is added to Garland's original "summed" metric, its effect on balance is easily overwhelmed by the $eDir$ and $eShape$ terms. The approach of scaling the basic cost, $eFit + k \cdot area$, by penalty terms, suffers much less from this.

Finally, while trying to construct a cost metric capable of guiding face cluster creation according to some notion of "goodness", it is tempting to engage in micro management, adding a succession of terms. One must be careful about this; the face cluster algorithm is a greedy algorithm, and trying to get subtle effects via the relatively blunt tool of the cost metric can be a frustrating exercise. Keeping the metric simple and straightforward seems the best approach.

5.5. Time and Space Trade-Offs

The cluster files for complex models can be large. For instance, the geometry of the whale model shown earlier in this chapter requires 3MB of storage (1.1 MB compressed). Its corresponding binary cluster file (which includes the geometry as well as the face clusters) is 15MB (9MB compressed.) While these files can be compressed for storage, they must be uncompressed for use by the radiosity simulation. This is not really a problem for moderate numbers of cluster files given the current cost of hard disk space; approximately US$5/GB for IDE disks. However, while I make an effort to limit the amount of cluster overhead for my particular application, radiosity, it would be nice to have a method for reducing the size of these files in situations where either we wanted to store more information per

cluster, or many (different) models must be used. Also, there are some situations in which we must traverse the entire face cluster hierarchy; in such cases, overly large files can lead to long disk read times, and problems with thrashing.

We look at two possible approaches to trading off cluster file size in return for more computation time: increasing the branching factor of the hierarchy, and truncating the hierarchy.

5.5.1. Picking a Branching Factor

The standard face cluster hierarchy is a binary tree; it has a branching factor of $b = 2$. It is relatively simple to recast this hierarchy as one with a larger branching factor; $b = 4$ would give us a quadtree, and $b = 8$ an octree. This can lead to space savings, but we must analyse the corresponding impact on performance.

Let us briefly recap the situation where we wish to build a hierarchy with a branching factor of b above n leaf polygons. The total number of internal nodes (and thus clusters) will be

$$i = \frac{n-1}{b-1}. \tag{85}$$

The height of the corresponding balanced hierarchy is

$$h_{bal} = \left\lceil \frac{\log n}{\log b} \right\rceil + 1, \tag{86}$$

and that of the corresponding unbalanced hierarchy is $h_{unbal} = i + 1$.

Ignoring storage for leaf nodes, which remains constant, the storage required by our hierarchy is $S = (S_{node} + bS_{pointer})i$, where S_{node} is the internal node storage (e.g., a bounding box for a bounding volume hierarchy), and $S_{pointer}$ is the storage size of a pointer. The query time for each node is simply bQ_{node}, where Q_{node} is the time taken to test a single node—we must test each of a node's b children[1]. Thus the worst-case query time is

$$Q_{bad} = bQ_{node}(h_{unbal} - 1) = \frac{b}{b-1} Q_{node}(n-1), \tag{87}$$

1. For a bounding volume hierarchy we must test each child because there is a chance more than one child will match.

and the best-case time is

$$Q_{best} = bQ_{node}(h_{bal} - 1) = \frac{b}{\log b} Q_{node} \log n . \tag{88}$$

In the case of the face cluster hierarchy used by the radiosity algorithm, S_{node} is an order of magnitude larger than $S_{pointer}$, so we approach space savings of $1/(b-1)$ over a binary tree. Thus, a quadtree will take $1/3$ the space of the original binary tree, and an octree $1/7$ the space, ignoring the storage for leaf faces. These are considerable space savings, and they increase with b.

The drawback to a higher branching factor is slower spatial-data structure queries. It is often assumed that a randomly-built hierarchy has height $O(\log n)$, and thus is on average close to being balanced. We will assume that here, especially as we have a specific term to promote balance in our cost equation. (See **Section 5.6** for empirical evidence that our hierarchies are always close to the balanced case.) In such a case, it is Q_{best} that determines query performance. **Figure 76** shows the b-dependent term of Q_{best}. For quadtrees and higher, the cost increases close to linearly with b. Interestingly, quadtrees have the same query cost as binary trees, and a hierarchy with a branching factor of three has a slightly lower cost[1].

This analysis assumes we have built a hierarchy with branching factor b from scratch, guaranteeing that every internal node except one has b children. However, constructing an algorithm that directly produces such a face cluster hierarchy is challenging. We would have track all possible b-way grouping operations on adjacent nodes, rather than using simple bi-directional links. For a node with m neighbours, there would be $m!/((b-1)!(m-b+1)!)$ possible groupings. For example, for $m = 6$ and $b = 4$, we would need to represent (and calculate the cost of) 20 potential groupings rather than 6. Also, if we start from a triangular mesh, for the initial leaf nodes $m \leq 3$, and many internal nodes would have only m children, with $m < b$.

In general it is simpler to recast the binary hierarchy produced by the original algorithm into one with a higher branching factor. In doing so, we must maintain the structure of the original hierarchy, so that only groups of adjacent nodes are produced. If this is not done, we could have clusters containing disconnected surface areas, violating the assumptions of the face cluster hierarchy.

---

1. A three-way hierarchy is only 5% faster in this situation. In general binary trees work better because of the extra cost of three way group-selection or comparison operations. A binary comparison requires fewer gates to implement that a trinary one.

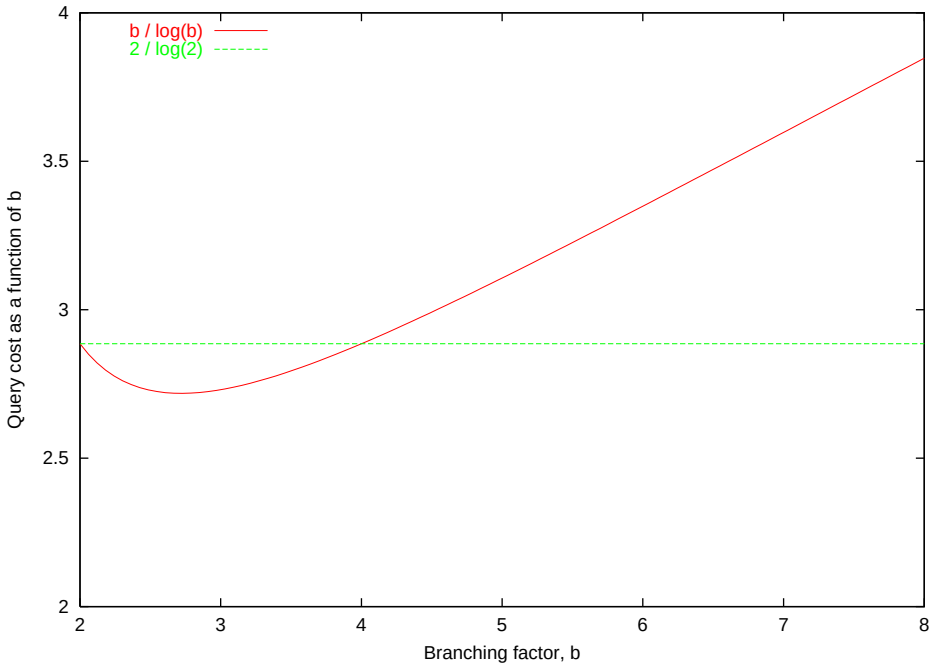

Figure 76: The b-dependent factor in the query cost for a balanced spatial data structure with a branching factor of b.

In practice this leads to the requirement that a node and its children in the transformed tree correspond to the root node and leaf nodes of a valid subtree of the original binary hierarchy. This restriction can produce a number of internal nodes that have less than b children, with two consequences: potentially wasted pointer space on one hand, but fewer nodes to test against on the other.

I will not go into details here, but it can be shown that:

- the smallest possible number of children in such a tree is $b' = b/2 + 1$;
- any completely balanced or completely unbalanced tree can be transformed into one where all but one of the internal nodes have the full set of b children;
- in the worst case, it is possible for all internal nodes in the new hierarchy to have b' children;

- the maximum height of such a worst-case tree is $2(n-1)/b+1$, and the number of internal nodes is $2(n-1)/b$[1].

The consequences of this are that the best-case storage and performance are the same as for our original analysis, but in the worst case performance and pointer storage costs of the transformed tree match that of the original binary tree. However, the total internal node storage is reduced by a factor of $2/b$, so we still save some space with a higher branching factor. For $b = 4$, whereas we were originally guaranteed to reduce our internal node storage to $1/3$ of its original size, with the transformed tree this is just the best case, with the worst case being $1/2$.

In summary, by increasing the branching factor we can decrease the storage needed for face clusters, ignoring pointer storage, to at worst $2/b$ of that required by the original binary hierarchy. This can increase the query time of the face cluster data structure by a factor of up to $b/(\log b)$. However, this is important only if we are using the data structure for visibility queries, or proximity or collision detection. In my current implementation of face cluster radiosity, the data structure is only used for hierarchical radiosity, and a nested grid data structure is used for visibility queries. Thus the only performance consideration is that of the radiosity algorithm, which we will look at next. For face cluster applications which are primarily query-based, there is no performance drawback and considerable space savings to be had by recasting the hierarchy as a quadtree. Higher branching factors are possible, but the space savings become progressively smaller and performance is impacted; finding the ideal branching factor would require further experiment or analysis.

5.5.2. The Impact of Branching Factor on Radiosity

For the face cluster radiosity algorithm, we must examine the impact of the branching factor on the following stages of the radiosity algorithm (see **Section 2.4.2**).

Gather

The cost of the gather operation in radiosity depends solely on the number of transport links currently existing in the simulation. This number depends on the refinement epsilon, ε, as well as the quantization of surfaces—how we discretize them, and how we organize them into a hierarchy. The branching factor affects

1. Briefly, the smallest binary tree corresponding to this worst case contains a left subtree that is unbalanced and has b' leaf nodes, and a right subtree that has b/2 leaf nodes. Larger such trees can be produced by repeating this base tree.

the set of possible links between hierarchical surface regions. Larger branching factors correspond to a smaller set of possible links, that cover the possible transport space more coarsely. If there is some theoretically optimal number of transport links l_ε that exactly meets the desired refinement threshold, the imposition of any hierarchy leads to approximation error; we still meet the threshold, but it requires l_ε^b links, where $l_\varepsilon^b > l_\varepsilon$. The larger the branching factor, the larger the approximation error, and thus l_ε^b.

Unfortunately, given the wide range of transport schemes, and the dependence of transport error on the current state of the radiosity solution in the popular brightness-weighted schemes, the relation of these quantities is hard to analyse except in trivial, unrealistic cases. We can make the following simplistic analysis, however.

Assume that all nodes have an equal number of links pointing to them (**Section 2.4.1**). We will compare a hierarchy with branching factor $b = 2^m$ to one with branching factor 2, when both have the same n leaf nodes. From **Equation 85** the binary hierarchy must have $(n-1)((b-2)/(b-1))$ more nodes than the hierarchy with higher branching factor. For each of these nodes, the corresponding links must be pushed down to the first descendent node that has a counterpart in the non-binary hierarchy. This must result in the generation of at least one new link, and at most 2^{m-1} new links, for each of the links to be pushed down. Thus we can conclude that, assuming a constant refinement ε and the same scene, the links formed in a hierarchy with branching factor b will have the following relation to the number of links formed in a binary hierarchy:

$$l_\varepsilon^b \geq \left(1 + \frac{b-2}{b-1}\right) l_\varepsilon^{b=2}. \tag{89}$$

This increase in the number of links, and consequently link storage and gather time, quickly approaches a factor of two for branching factors of eight and higher.

Push-Pull

The cost of a push-pull operation for an internal node is linear in the branching factor. (We must push irradiance down to b children, and then average their radiosities during the pull phase.) We can write this cost as $C_P b$. For each leaf there is an additional constant cost in transforming irradiance into radiosity, C_R. This gives us a total cost for a push-pull operation over the solution hierarchy of

$C_P bi + C_R n$. In the balanced and unbalanced cases, where almost all nodes have b children, we have

$$C_{best} = k_P \frac{b}{b-1}(n-1) + k_R n, \tag{90}$$

and in the worst case mentioned above, where all nodes have b' children, we find

$$C_{worst} = k_P \frac{2b'}{b}(n-1) + k_R n = k_P \frac{(b+2)}{b}(n-1) + k_R n. \tag{91}$$

Thus by increasing the branching factor, we can decrease the cost for the internal nodes, by a maximum factor of 2. The cost for a binary tree is $2k_P(n-1) + k_R n$, and for an octree it lies between $1.143k_P(n-1) + k_R n$ and $1.25k_P(n-1) + k_R n$.

We can conclude that for face cluster radiosity, increasing the branching factor would decrease cluster memory use, and speed up the push-pull section of the radiosity algorithm, but likely increase transport link memory use, and slow down the gather section of the algorithm. My feeling is that a low branching factor at the top of the hierarchy is important, as accuracy at the top of the hierarchy is crucial to any top-down algorithm. This hypothesis needs to be tested empirically, however. In the meantime, it seems that increasing the branching factor to at least four is an experiment well worth pursuing, especially considering that previous hierarchical radiosity algorithms typically have branching factors of four or more, and that face cluster storage is typically much greater than link storage. An alternative to increasing the branching factor of the hierarchy everywhere might be to do so at lower levels of the hierarchy only, leaving the higher levels with the original low branching factor.

5.5.3. Truncating the Hierarchy

One of the particular problems with large cluster files for the radiosity application is that typically a large portion of the cluster file goes unused. This is especially the case with models containing large flat areas; a single cluster representing the flat area suffices for the radiosity simulation, and the only use for the rest of the hierarchy in that area is for the final push-to-leaves step.

A useful way to take advantage of this is to truncate the face cluster file, and thus the hierarchy. The cluster file can be organised so that the leaf nodes belonging to any given cluster are numbered consecutively. Thus if we store face ranges with each cluster, it is possible to go directly to the leaf polygons enclosed by that cluster without traversing the hierarchy below it. Because the cluster file is

generally stored in a reverse-log format, where clusters occur in reverse order from the order in which they were generated, it is possible to simply eliminate all nodes below a certain cut through the hierarchy by discarding all nodes after a certain point in the file.

The result is a trade-off between memory or disk space and speed. If the radiosity algorithm needs to descend beyond the cut at which the rest of the hierarchy has been removed, it will incur a speed hit as it must switch directly to using the leaf polygons in the simulation. The truncation can also be viewed as a form of lossy compression of the hierarchy. Because the cluster file is written out in order of clustering, the first clusters to be discarded are those most likely to be largely flat, and thus ideal candidates to be discarded.

For technical details on how this truncation can be carried out, see **Section 6.2.3**.

5.5.4. Out of Core Face Clustering

Unfortunately the face cluster algorithm has poor performance when the model being clustered will not fit in main memory. Because the algorithm starts by creating dual edges between all adjacent faces, its memory consumption is greatest during initialisation, when it must access the entire model. Finding adjacency relationships between faces for such large models is also an expensive operation. These problems are similar to those faced by geometric simplification methods based on edge contractions when models grow too large.

One solution that has been presented for geometric simplification is to use a variant of Rossignac and Borrel's vertex decimation [Ross93, Lind00]. In vertex decimation, we place a grid around the model, and then collapse all model vertices in one cell to a single vertex, discarding all triangles that become degenerate. (That is, all triangles that have two or more vertices lying in the same cell.) We can adapt this algorithm to the dual-space face-clustering algorithm, by adding all triangles in the same cell to a single face cluster. The algorithm is outlined in **Listing 2**.

This approach has the advantage that, like the out-of-core simplification algorithm, the memory use is determined completely by the grid size used, and thus can be constrained to match the available in-core memory. Moreover, it generates the appropriate initial set of dual edges as part of the triangle addition step; there is no separate, triangle adjacency-finding algorithm needed.

The main disadvantage of this approach is that it is no longer strictly guaranteed that the surface within a cluster is a manifold connected surface, although

```
OutOfCoreCluster()
{
     Engrid(); // place grid around model

     foreach (cell in grid)
          initialise a cluster quadric

     foreach (triangle in model)
          find cell coordinate of triangle's centre
          add triangle to that cell's cluster, updating quadric

          find cell coordinate for each vertex
          foreach (edge with endpoints in different cells)
               add dual edge between those cells
                    if not already present

     Initialise(dualEdges);  // find error term for each edge
     FaceCluster(dualEdges); // add edges to heap and
                             // cluster as usual.
}
```

Listing 2: Out-of-core face clustering. Each grid cell is treated as a cluster. The process of adding triangles to cells also identifies neighbour relations between those clusters.

it is the usual case. (This is the corresponding problem to the way Rossignac and Borrel-style vertex decimation can produce a non-manifold output surface from manifold input.)

The other disadvantage is that we limit the depth of the face cluster hierarchy. At the leaf clusters corresponding to the original grid cells, we go directly from a cluster to all the faces contained in that cluster. However, this is similar to the approach outlined in **Section 5.5.3**, where we deliberately truncate the hierarchy to save space. In fact, the approach outlined there is needed to handle such a cluster hierarchy, where a cluster can contain more than two leaf faces.

5.6. Performance of Face Clustering

In this section I will give some results for the modified face cluster algorithm presented in this chapter, for the kinds of polygonal models I'm most interested in being able to handle well with radiosity algorithms. **Figure 79** shows sixteen such models, ranging in complexity from a few thousand polygons (the cup and ellipsoid) to over one million (the buddha).

Figure 77 shows log-log plots for both memory and time use by the cluster algorithm. The models in both graphs fall along a line of slope 1, suggesting both are roughly $O(n)$. In the case of memory consumption, this is expected. In the case of time consumption, we expect $O(n\log n)$ best case performance, and get it for this range of models. (For large numbers, $O(n\log n)$ looks much the same as $O(n)$ on any plot.) Crucially, the time results are always close to this best-case, as opposed to the worst-case $O(n^{3/2}\log n)$ performance that would be represented by a line of slope $3/2$.

The reason for this good performance is shown in **Figure 78**, which shows the average and maximum dual-edge degree during the clustering process for each model. The average degree always falls between 4 and 6 edges, which is consistent with the lowest curve of **Figure 71**, and thus the assumption about the dual-edge degree being constant that underlies the theoretical best-case $O(n\log n)$ performance of the algorithm are satisfied. The maximum degree is a measure of the largest cluster boundary. It is particularly high in the case of the buddha; I speculate that this is because of its planar base remaining as one cluster until relatively high in the cluster.

Finally, free source code implementing the face cluster algorithm is available on the internet, at http://www.cs.cmu.edu/~ajw/thesis-code/. As well as its use in radiosity, this code should prove valuable for other spatial data structure-oriented algorithms that need to be able to handle large detailed models well, such as collision detection or ray-tracing.

5.7. Comparison to Other Techniques

5.7.1. Hierarchies of Oriented Bounding Boxes

Hierarchies of oriented bounding boxes have proven useful in other contexts besides radiosity, in particular for query-based operations on spatial data structures, such as collision detection [Ball81]. Barequet et al. present the BOXTREE data structure, which is essentially a hierarchy of oriented bounding boxes, con-

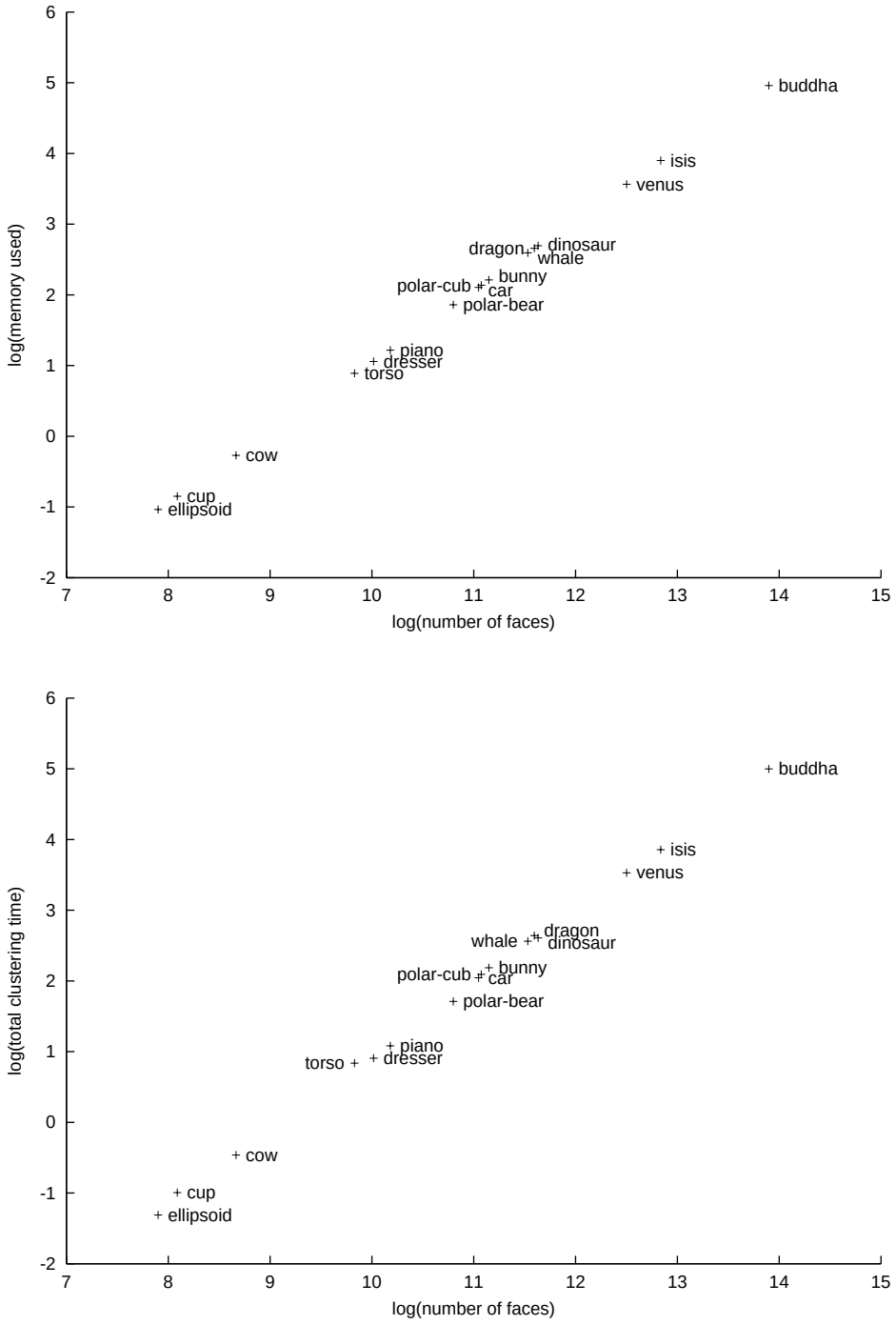

Figure 77: Performance of the new face cluster hierarchy creation algorithm for a variety of models.

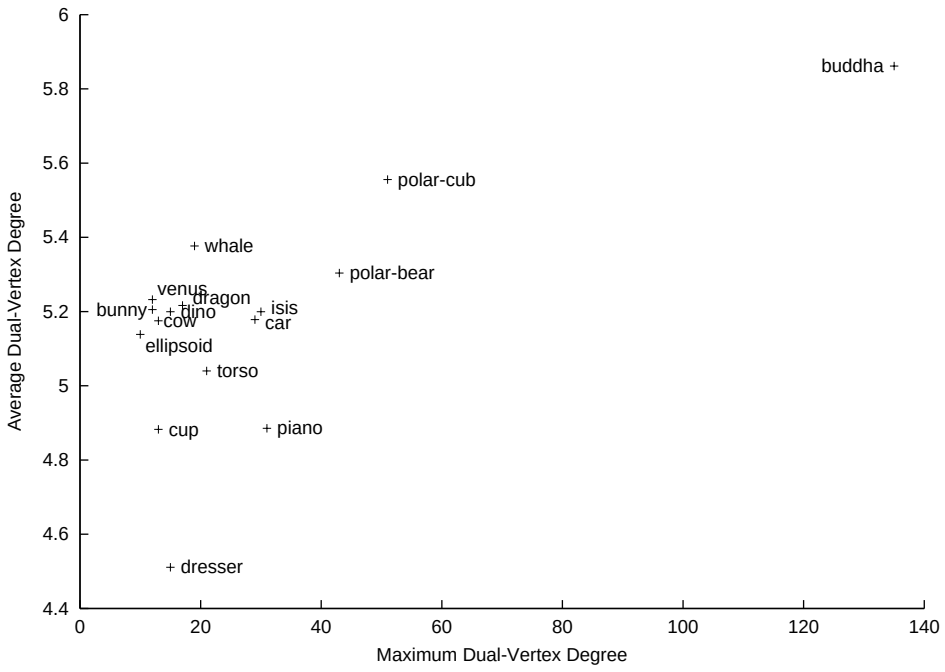

Figure 78: Variance in dual-vertex degree for a variety of models.

structed top-down [Bare96]. They also discuss several methods for both creating the hierarchy and picking appropriate bounding boxes. They conclude that the minimal-component pie-slice works better than oriented bounding boxes for their test models. I suspect that this is because:

- Their box-tree hierarchies calculate bounding boxes for the leaves of the hierarchy, i.e., the original triangles of the model. In this case, the advantage of a triangularly-shaped bounding volume is obvious.
- The triangular shape is more oriented than a box; this could lead to fewer stability problems and more accurate PCA directions for their strategy of using child bounding boxes to form the covariance matrix.

Pie slices also take slightly more storage than oriented bounding boxes (8 floating point numbers plus a rotation, as opposed to 6 floats plus a rotation.)

The BoxTree paper also presents results showing that, as we might expect, the strategy of using PCA to select one direction rather than all directions of the bounding box can lead to better performance.

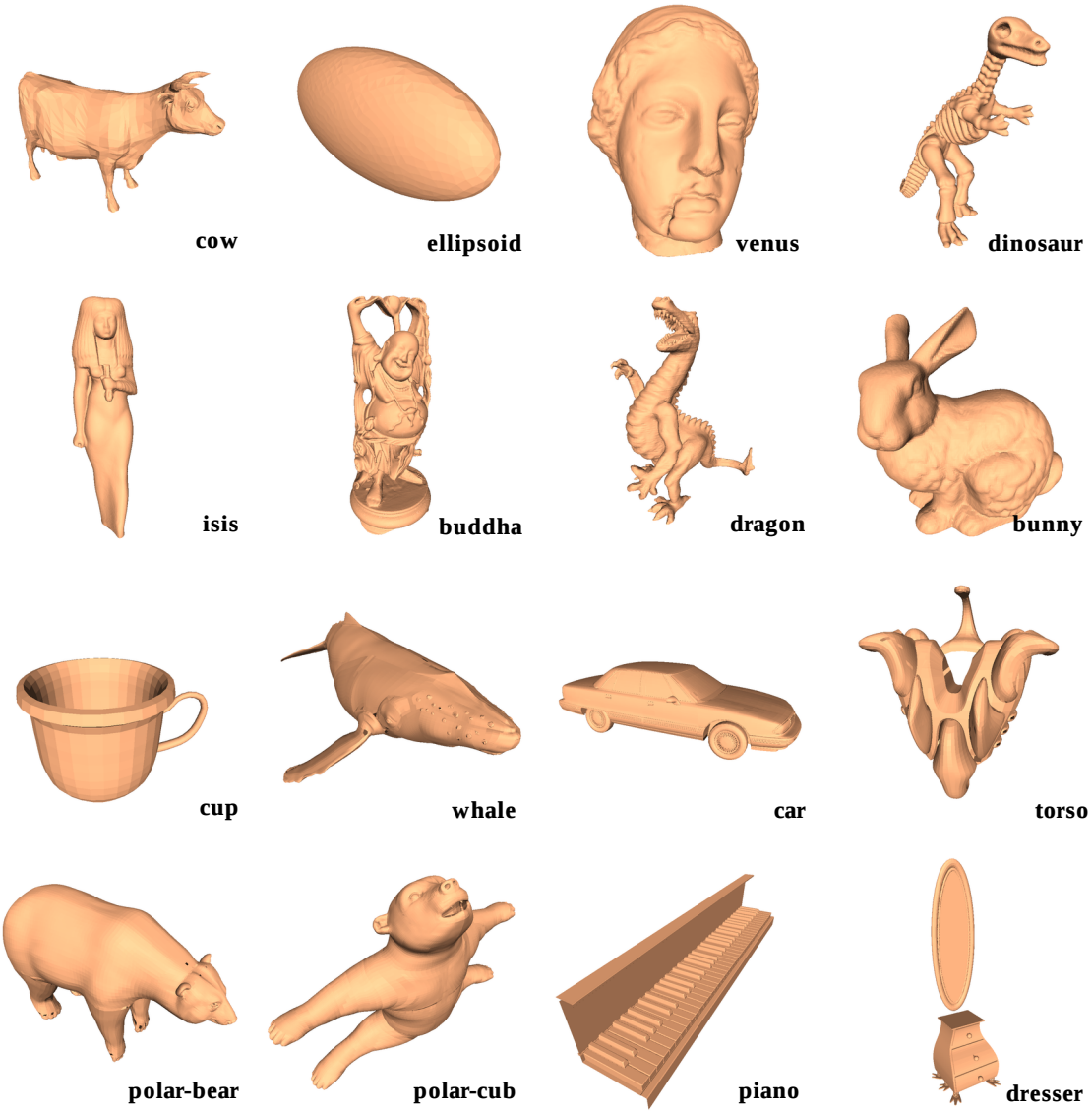

Figure 79: Test models. Shown are a selection of models of varying complexity. The new face cluster algorithm was run on each model and test results collected.

5.7.2. Volume Clustering for Radiosity

There are a number of different approaches to clustering in radiosity. Hasenfratz et al. have studied a number of the data structures and strategies used in volume

clustering [Hase99]. They classify clustering approaches by data structure used, and type of construction algorithm. Data structures are classified as either regular structures, such as octrees or *k*-d trees, that recursively subdivide space, or bounding volume hierarchies, which adapt to object geometry using optimization criteria. Construction algorithms are classified as either top-down (the simplest) or bottom-up. Within this schema, the face clustering algorithm is most closely related to bounding volume hierarchies, as it tries to adapt to object geometry, and it uses a bottom-up construction algorithm.

Face clusters have some of the advantages of both kinds of data-structure. Their speed of construction is similar to regular subdivision-based data structures, such as octrees, however their bounding boxes are of much higher quality. Their adaptiveness to geometry is similar to that of bounding volume hierarchies, but they do not suffer from overlapping clusters or mixing of surfaces in single clusters, because they are constrained to share only connected geometry.

The authors pay some attention to why cluster overlap in bottom-up cluster constructions is a problem. It leads to situations where scattered elements of a surface end up in different clusters, leading to disturbing visual artifacts. The face cluster algorithm, because it always merges topologically adjacent surface clusters, avoids this problem.

The bottom-up construction of face clusters means the algorithm is more complex to implement than simple top-down clustering methods, because adjacency relationships must be tracked. However, the constant-time nature of the quadrics used to evaluate the goodness of potential clusterings, and the use of a greedy iterative clustering routine, means it is much faster than most volume clustering methods that are trying to optimize for some feature of the hierarchy. These methods tend to be quadratic in the number of input surfaces, whereas face clustering is close to linear.

Finally, **Figure 80** compares the time taken for face clustering and volume clustering two of the models from **Figure 79**; the dragon and whale models. The volume clustering method used is one of the simplest (and fastest) possible: the tightest-fit octree. It proceeds by recursively subdividing a subset of the model into eight octants, and placing a cluster around each octant. Polygons in the model that are larger than the size of an octant are kept at the same level as the cluster being processed.

As can be seen, the two algorithms have roughly the same asymptotic complexity. The face cluster algorithm has a larger constant, but as a hierarchy need only be generated once per model, rather than on a per-scene basis, its amortised clustering time may well be smaller in many situations.

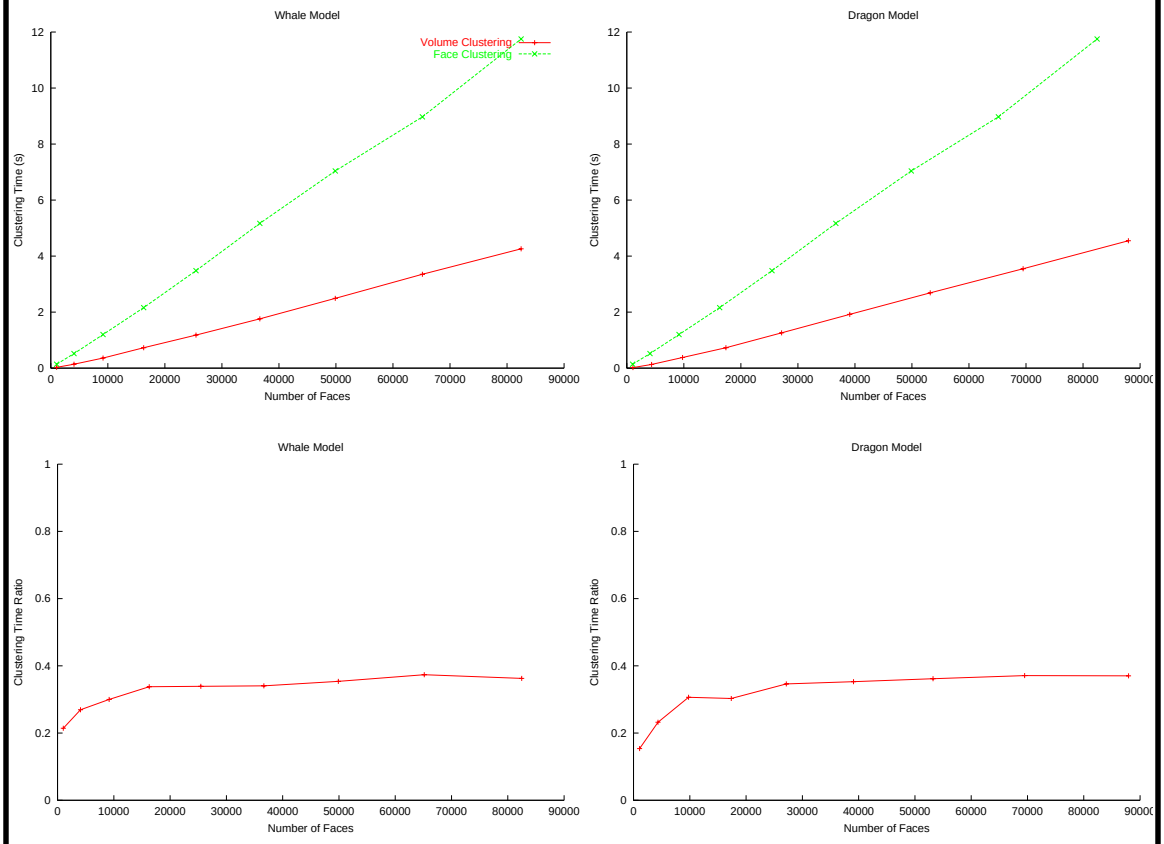

Figure 80: Speed comparison to simple volume clustering. Top: the time taken to volume cluster and face cluster two models, for a number of different resolutions of each model, generated by Garland's simplification algorithm. Bottom: the ratios of these times. The volume-clustering method used is the fastest possible; tightest-fit octree.

Face clustering takes around 2.5 times as long as face clustering, a figure that stays steady with increasing model size. This is a good result considering that the face cluster algorithm is trying to optimize the hierarchy according to flatness and shape criteria, whereas the volume clustering method is simply partitioning space.

5.8. Summary

In this chapter I have analysed the performance of the previous face cluster creation algorithm, in particular showing how hierarchy balance can have an adverse

effect on the construction algorithm, and presented a new algorithm that performs considerably better. The algorithm is based on a new cost metric, which

- produces more balanced hierarchies;
- is simpler and cheaper to evaluate, with quality comparable or better than the previous metric.

I have also modified the algorithm, resulting in

- faster bounding box calculations, by avoiding principle component analysis for flat clusters;
- better (tighter) bounding boxes for flat clusters, by using a minimal-area bounding rectangle algorithm.

Finally, I have provided results for the new algorithm for a wide range of models, showing that its running time is close to linear in the number of input polygons, and reasonably independent of model geometry. I have also compared face cluster hierarchies to other hierarchies of oriented bounding boxes, as well as those types of hierarchy commonly used in radiosity.