Taming Large 3D Models : Approximate Convex Decomposition

Ashley Tharp [*], Mukulika Ghosh [†], Nancy M. Amato [†]

August 7, 2012

Abstract

3D models are becoming larger and more complex due to hardware advances and higher demands for realism in 3D models and simulations. Decomposition can be used as a preprocessing step for making large model data sets manageable by splitting a large model into smaller sub models.

Since Exact Decomposition can be costly to do, Approximate Convex Decomposition (ACD) decomposes a model/polygon into "approximately" convex parts. By sacrificing some convexity, computational cost is greatly reduced and the number of resultant components is more manageable. There are many applications of ACD including particle simulation, rendering, computer vision, texture mapping, Minkowski Sum Generation, mesh generation, motion planning, and skeletonization.

In this paper, our approach involves simplifying the input geometry first, and then decomposing. Making a decomposition on a model that has been simplified has several advantages. By simplifying the model first, noise, fissures, and gorges on the surface of the model can be ignored, resulting in better and more consistent decompositions. (The quality of a decomposition cannot yet be quantified.) After the simplification, a mapping process is used to carry a decomposition back over from a simplified model to the unsimplified model. The mapping technique calculates a shortest path on the unsimplified model surface for every edge in the cut on the simplified model. Using simplification prior to decomposition decreases the overall computation time for large complex models as shown in our results.

Keywords: approximate, convex, concave, concavity, decompose, decomposition, model, geometry, simplification, simplify, simplifying

1 Introduction

3D models are becoming larger and more complex due to hardware advances and higher demands for realism in 3D models and simulations. Decomposition can be used as a preprocessing step for making large model data sets manageable by splitting a large model into smaller convex sub models. This is beneficial because convex shapes are easier for many computational geometry algorithms.

Our approach involves simplifying the input geometry first, and then decomposing. After decomposition, a mapping process maps a decomposition from the simplified version to the original version. This method results in decompositions of higher quality with less computational expense.

1.1 Contribution

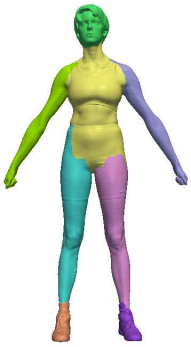
As shown in our results, simplifying a model before decomposing drastically improves computation time. The reduction in computation time is nearly exponential. Despite great reductions in computation time, this method has some drawbacks. While simplifying has the benefit of reducing complexity, there is a point of diminishing return where oversimplification causes poor quality decompositions and the reduction in computation time ceases to improve consistently. To elaborate, when simplifying, too much simplifying will make the shape unrecognizable. If there are no significant features left after simplification, the decomposition algorithm is going to make insignificant cuts that are not useful. In our research, we have not yet found this point of diminishing return. This is because, measuring the quality of a decomposition is not particularly simple to quantify.

[*]Computer Science, Collin College, Plano TX, 75074

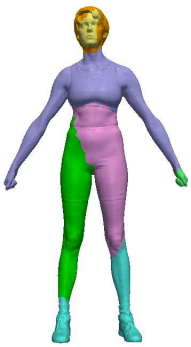
[†]Parasol Lab Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA

This research supported in part by NSF awards CRI-0551685, CCF- 0833199, CCF-0830753, IIS-096053, IIS-0917266 by THECB NHARP award 000512-0097-2009, by Chevron, IBM, Intel, Oracle/Sun and by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST).

(a) good decomposition



(b) not as good decomposition



(c) Bad decomposition

Figure 1: Quality of decomposition

Observe the figures in Figure 1. Figure (a)is what we would call good, (b), not as good, and (c), pretty bad. A decomposition is bad if it does not make cuts that adequately represent the shape of the model. Figure (b), does not have as many components, but that is not what makes it poorer quality. Observe how the blue leg does not end neatly at her hip and instead travels up to her waist. That is a bad cut. Despite the fact that a human being could roughly judge the quality of a decomposition, getting a computer to judge the quality of a cut may not be possible and is beyond the scope of this paper.

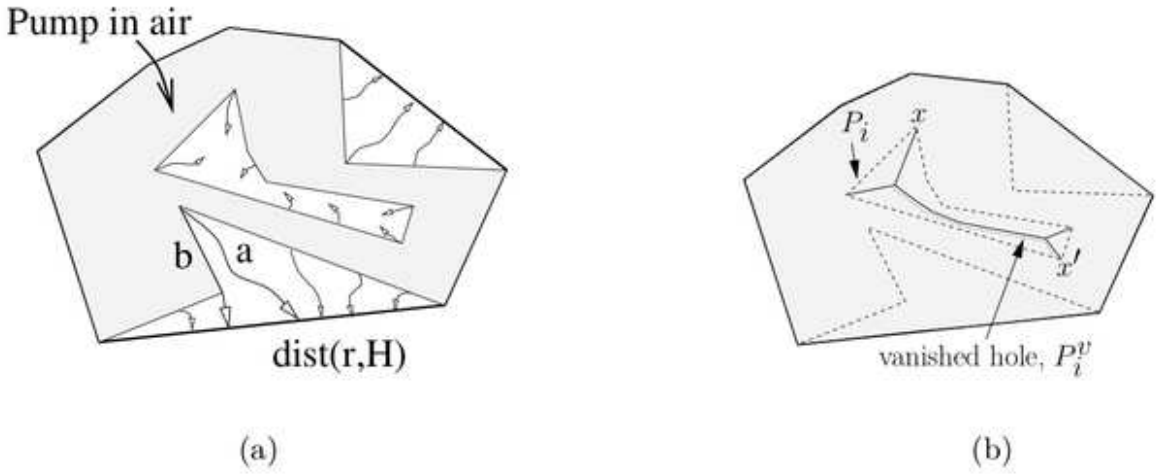
In conclusion, the results of our experiment have shown that when computation time is decreased, the quality of the decomposition also decreases. In some situations, the loss of quality may be negligible compared to the gains of decreased computation time.

2 Previous Work

A decomposition can be costly to construct. For minimum convex decomposition problems, there is O(n + r^3) time complexity for simple polygons without holes and NP-hard for simple polygons with holes, and the problem is NP-Hard for polygons with holes and polyhedra [4]. Exact composition may result in a decomposition with an unmanagable number of components [4]. To save time and get a more managable number of parts, one can decompose a model into approximately convex sub-models. A model P is said to be τ-approximate convex if concavity measurement of P is less than τ , i.e., concave(P) $< \tau$.

Concavity for a point x of P is measured as the distance to the surface of convex hull, H, of P, i.e., concave(x)=dist(x, H). [3]

Imagine P is a balloon is placed in a mold like in (a). Although the initial shape of this balloon is not convex, the balloon will become so if we keep pumping air into it. Then the trajectory of x to H can be defined as the path traveled by x from its position on the initial shape to the final shape of the balloon. [3] Although this intuition is simple, a path like that is not easy to define or compute [3].



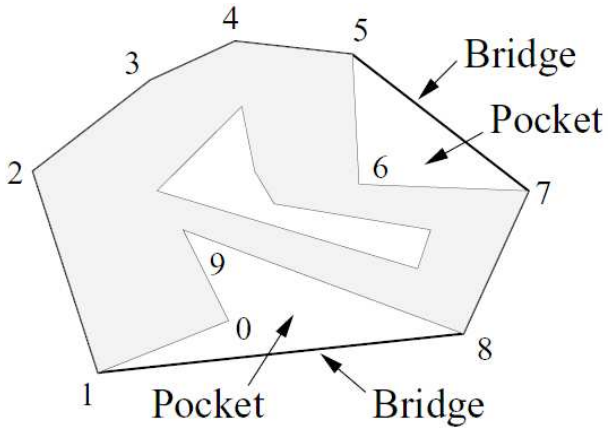


Figure 2: Bridge and Pocket in polygons

In essence, by sacrificing a little convexity, a model can be decomposed into fewer number of components in less time using approximate convex decomposition.

Another method we employ in our approach is the simplification of a model prior to decomposition. We employed a method very similar to the approach outlined in Micheal Garland's paper [1]. Basically, many computer graphics applications require complex, highly detailed models to maintain a convincing level of realism [1].

Consequently, models are often created or acquired at a very high resolution to accommodate this need for detail. However, the full complexity of such models is not always required, and since the computational cost of using a model is directly related to its complexity, it is useful to have simpler versions of complex models [1]. Naturally, we would like to automatically produce these simplified models.

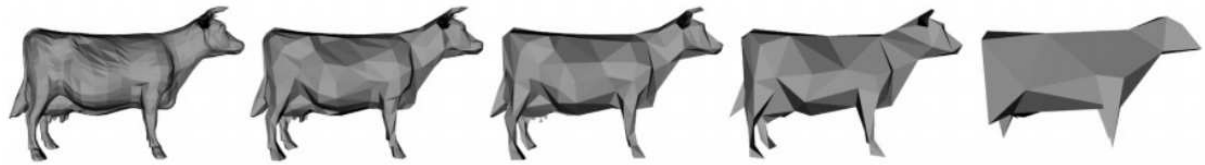


Figure 3: simplification of a cow model

Viewing the figures in Fig. 3 one can observe that the simplification of a model still preserves the model's basic features while reducing the number of vertices. During the simplification process, vertices are not removed at random, but are removed according to saliency, with the most salient vertices left

to remain in the final model. Although the simplification process is designed to retain the most salient features of a shape, there is a point where a model can be oversimplified. For example, observe how the very last simplified cow has lost it's horns. For optimal results, over simplification is not desirable.

To elaborate on the simplification process itself, in order to select a contraction to perform during a given iteration, we need some notion of the cost of a contraction. To define this cost, we attempt to characterize the error at each vertex. To do this, we associate a symmetric 44 matrix Q with each vertex. Our simplification algorithm is built around pair contractions and error quadrics. The current implementation represents models using an adjacency graph structure: vertices, edges, and faces are all explicitly represented and linked together. To track the set of valid pairs, each vertex maintains a list of the pairs of which it is a member. The algorithm itself can be quickly summarized as follows:

1. Compute the Q matrices for all the initial vertices.
2. Select all valid pairs.
3. Compute the optimal contraction target v for each valid pair v(v1 , v2). The error vT (Q1 + Q2) of this target vertex becomes the cost of contracting that pair.
4. Place all the pairs in a heap keyed on cost with the minimum cost pair at the top.
5. Iteratively remove the pair (v1 , v2) of least cost from the heap, contract this pair, and update the costs of all valid pairs involving v1.

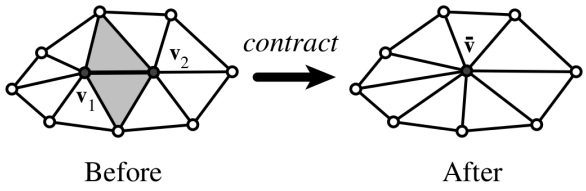


Figure 4: Edge Collapse operation used to decimate an edge

The process of removing a vertex pair can be observed in Figure 4.

2.1 Related Work

There are many decomposition methods that rely on using a simplified version of a polygon or model [5] An example of this, is the straight line skeleton. Basically, the straight line skeleton uses a skeleton generated by shrinking the model or polygon into it's center using a "wavefront". The skeleton can be used to identify the most visually striking parts of the model. Figure 5 illustrate the difference between a straight line skeleton and the medial axis of a polygon.

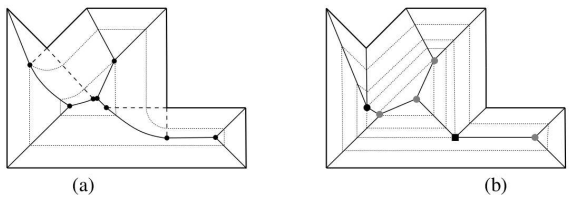


Figure 5: Comparison of straight line skeleton and medial axis of a polygon

Another method which incorporates simplification into the ACD process is explained in paper [2]. Their method differs from our method in that their method only does surface decomposition, while our method incorporates volumetric decomposition. Surface simplification is only capable of generating convex patches as opposed to convex sub models.

Our method is different because we use volumetric simplification and not surface simplification. As, well our method includes a mapping process for mapping a decomposition from the simple model to the original model.

3 Preliminaries

Our general strategy for computing ACD's follows the approach for polygons. Briefly, an ACD is generated by recursively removing (resolving) concave features in order of decreasing significance, i.e., concavity, until all remaining components have concavity less than some desired bound. Then some appropriate cuts are made at the points of highest concavity. Finding the point to make the cut is simple, but making the cut itself is much more complex.

Algorithm 4.1 Approx_CD(P, τ)

Input. A polygon, P, and tolerance, τ.

Output. A decomposition of P, $\{C_i\}$, such that $\max\{\text{concave}(C_i)\} \leq \tau$.

```
1: c = concave(P)
2: if c.value < τ then
3:    return P
4: else
5:    {C_i}=Resolve(P, c.witness).
6:    for Each component C ∈ {C_i} do
7:       Approx_CD(C,τ).
8:    end for
9: end if
```

Figure 6: ACD Pseudocode

The two main operations required for acd are: measuring the concavity of a feature(s), and resolving specified concave feature(s). The approach outlined above is the same strategy applied to compute acds for polygons [4]. For a given polygon P , the concavity of notches x of the polygon P are computed. Then, a notch x is resolved by adding a diagonal from x to P such that the dihedral angle of x is less than 180 .

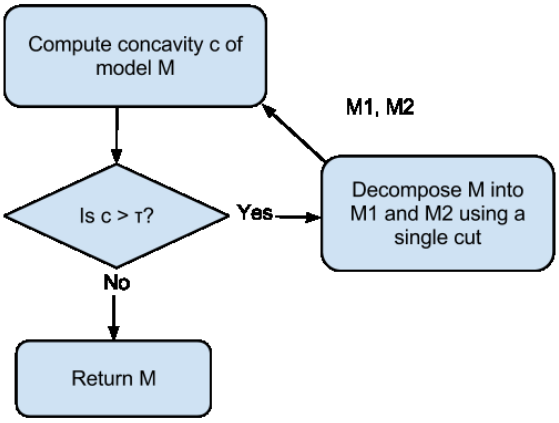


Figure 7: ACD Process

4 Our Method

The method in this paper is different from the typical ACD process in that their in an extra simplification and mapping step.

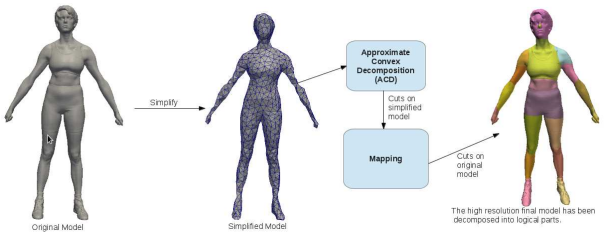


Figure 8: Our Method

For our particular implementation, we had to write a converter to translate between the ACD and simplifying processes.

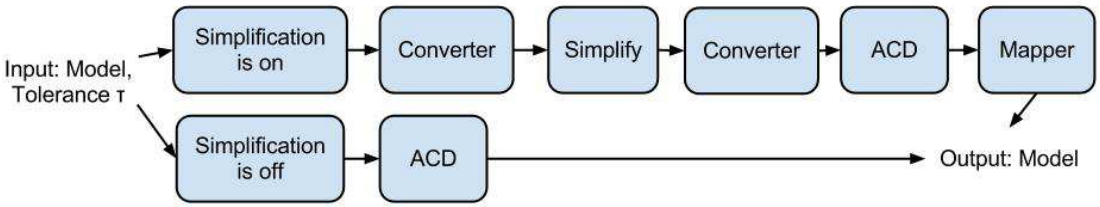


Figure 9: Our Method

You can see in Figure 9, the converter steps before and after the simplification process.

Since the ACD and simplification algorithms have different ways of storing a model, a converter had to be written for passing models to and from. The converter will transfer model data that in universal to both types, and also generate data that is not. There are certain things that the simplification algorithm uses that the ACD algorithm does not and vice versa.

After the model has been simplified and decomposed, it is given to a mapper. The mapper will take a path from a simplified model and create an equivalent path on the original model before simplification. The A* algorithm is used for this. A* is used to "fill in the gaps" between vertices. Vertices that exist on the simple model will also exist on the original model because the simplification process does not create new vertices, it only decimates existing unnecessary vertices.

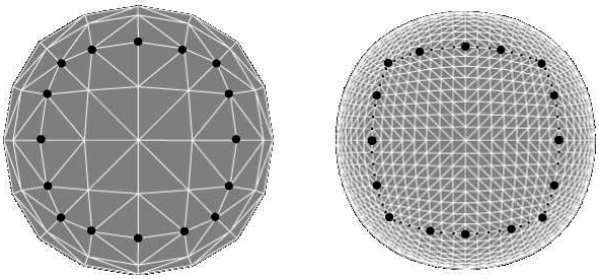


Figure 10: Mapping Process, left:simplified right:unsimplified

Looking at figure 10, it can be seen that the large black dots represent the vertices that exist on both models. Technically they are copies of the same vertex. Although they may not have exactly the same position cooridinates, they will have the same vertex id. The small black dots connecting the large dots are the "in between" vertices found by the A* pathfinding algorithm.

PsuedoCode:

Algorithm 1 Map(SC, OM)

Input: Original Model OM and Small Cuts SC.
Output: Cuts on the original model LC

```
for each cut C in SC do
    make a long cut, l for c with path(l) = ∅
    for each edge e(v1, v2) in path(C) do
        find corresponding path p(v1, v2) in OM
        path(l) ← path(l) ∪ p(v1, v2)
    end for
    insert l to LC
end for
return LC
```

5 Results and Future Work

For this experiment we used a dense model of a female with 234442 triangular faces. Future work will include more and different models. Total run times were recorded both with and without simplification.

Trial	Runtime (s)
1	1000.31
2	850.30
3	1369.00
4	537.22
5	896.67
6	746.21
7	1444.29
8	2191.01
9	399.31
10	2071.78

Parameters used were: tolerance = 0.1, error = 0.005, feature value = 0.0001
After 10 trials the average time was: 1250.61 seconds.

When testing the reduction in runtime with simplification, the reduction in runtime is dependent on the target number of faces. If a model begins with a 100,000 faces and we want to reduce it to 1,000 faces, then 1,000 is the target number of faces. In our experiment to determine target number of faces, instead of specifiying a specific number such a 1,000, we chose to make the target number based on some factor of the original number of faces. A target number of faces of Total/10 would result in a model that had 1/10th the number of faces.

Target	1	2	3	4	5	6	7	8	9	10	Average Time
Total/10	11.11	11.12	11.22	11.13	11.15	11.82	12.8	11.23	11.14	11.22	10.27
Total/25	6.89	6.90	6.96	6.88	7.39	6.91	6.94	7.16	6.88	6.94	6.29
Total/50	8.52	8.36	8.43	8.43	8.36	8.39	9.77	8.36	8.35	8.39	7.70
Total/75	6.45	5.42	5.48	5.43	5.56	6.07	5.47	5.46	5.65	5.48	5.10
Total/100	2.17	2.18	2.17	2.18	2.18	2.18	2.17	2.17	2.19	2.17	1.96
Total/125	4.26	3.26	3.29	3.26	3.28	3.35	3.30	3.26	3.25	4.26	3.05
Total/150	2.38	2.23	2.24	2.22	2.29	2.20	2.22	2.22	2.22	2.24	2.02
Total/175	1.27	1.27	1.27	1.27	1.26	1.40	1.28	1.27	1.26	1.27	1.16
Total/200	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.77	0.76	0.76	0.69
Total/225	1.16	1.15	1.16	1.14	1.16	1.15	1.16	1.15	1.16	1.14	1.04
Total/250	1.17	1.15	1.15	1.15	1.15	1.16	1.17	1.15	1.21	1.15	1.05

Parameters used were: tolerance = 0.1, error = 0.005, feature value = 0.0001
10 trials were done for each target number of faces. It can be seen from the table that more simplification of the model will consistently result in a lower computation time.

It can be seen from the bar graph that our results are nearly exponential.

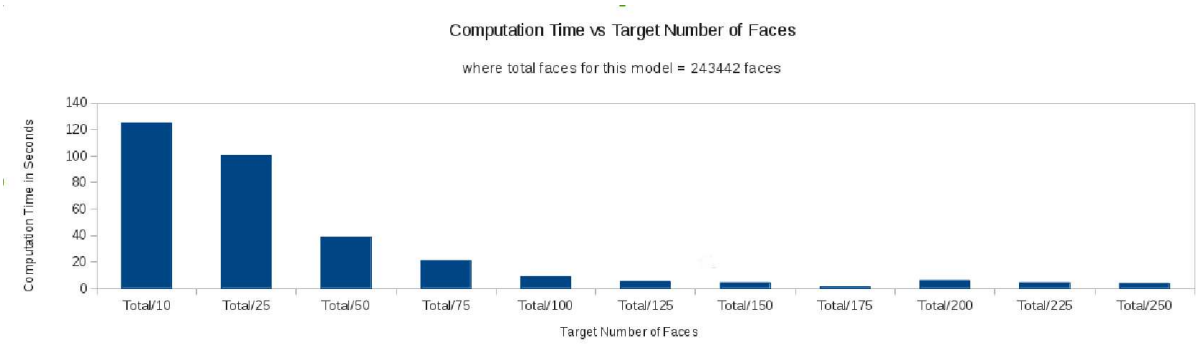


Figure 11: Computation Time vs Target Number of Faces

6 Discussion and Analysis

By reducing the number of faces, the overall computation time is drastically reduced. When simplifying a model prior to decomposition, there is a point of diminishing return when simplifying where the model loses it's important features. If a model does not retain it's shape during the simplification process, then the decompositions made upon it will not be meaningful.

7 Summary and Conclusions

By reducing the number of faces, the overall computation time is drastically reduced. Although computation time is greatly reduced, decomposition quality goes down with computation time.

Future work will concern bringing up the quality with decreasing computation time, and automating the process of finding the best decomposition. To find the best decomposition many paramaters must be tested. Automating this process would allow greater precision in finding the best decomposition because it would not be necessary for a programmer to manually test many many parameter values. This is particularly useful because every model requires different parameters to produce the best possible decomposition. These parameters cannot be known, although they can be narrowed down to a particular range, prior to adequate testing.

References

[1] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics, 1997.

[2] K. Mamou and F. Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3501 –3504, nov. 2009.

[3] Jyh ming Lien and Nancy M. Amato. Approximate convex decomposition of polygons. In *In Proc. 20th Annual ACM Symp. Computat. Geom. (SoCG*, pages 17–26, 2004.

[4] Jyh ming Lien, Nancy M. Amatodepartment, and Computer Science. Approximate convex decomposition. In *In Proc. 20th Annual ACM Symp. Computat. Geom. (SoCG*, pages 457–458. ACM Press, 2004.

[5] Mirela Tanase and Remco C. Veltkamp. Polygon decomposition based on the straight line skeleton. In *Proceedings of the nineteenth annual symposium on Computational geometry*, SCG '03, pages 58–67, New York, NY, USA, 2003. ACM.