Machine Learning and Optimization for 3D data

Shape Retrieval with Spherical Bounding Volume Hierarchy

Sang Hyun Son
Chang Woon Choi

About Spherical Bounding Volume Hierarchy(BVH)

Problem Formulation

We present a 3D object search engine that takes 3D mesh data as input and shows the most similar 3D models in our database. Most of current shape retrieval algorithms need to compare the input model with each model in their database(by their own metrics) in linear fashion, which takes $O(N)$ times. To reduce this time complexity to $O(logN)$, here we propose to use hierarchical structure for shape retrieval. To be specific, we will show that Bounding Volume Hierarchy(BVH) could give precise and stable results as other methods, with much lower cost.

A Bounding Volume Hierarchy(BVH) is a tree structure of a set of geometric objects (typically, triangle mesh) that are wrapped in bounding volumes which form the nodes of the tree. In this project, we selected sphere for bounding volume. This is because various distance measure operations (e.g. minimum distance) run very fast with spheres, and are easy to implement.

BVH for single 3D Object : Query

The first part of our project is about building spherical BVH on a single model. In the search process, we build this BVH for the query model and use it for accelerating entire algorithm. We had tried to start from medial axis of 3D mesh, since it preserves valuable topological features of the given model. However, we found out that many models in our database have significant flaws for extracting medial axis. Thus, we took naive approach – we sampled points on mesh model uniformly. The spheres centered on those sampled points comprise leaf nodes of our BVH.

From the leaf nodes, we can build inner nodes in bottom-up fashion. We merge spheres in lower level to create those in upper level. Since we aim to build efficient BVH, we have to find best-matching nodes in the lower level that would incur minimum increase in upper level spheres' volume. We used KD tree for finding such pairs.

BVH for multiple 3D Objects : DB

Next, we have to build spherical tree structure of entire 3D models in our database in order to run our retrieval algorithm efficiently.

When the tree structure for DB is first created and the first model is inserted, we can use BVH for the first model itself as our initial tree structure. After that, when the user inserts a new 3D model to our database, we can consider it as a query model in the first place. Once constructing BVH for the input model is finished, we can use it to search for the most similar model in our database and find the most appropriate place to insert our new model. After we find the best node to branch out, we make a new branch starting from the node and ends in the leaf node that contains our newly inserted model.

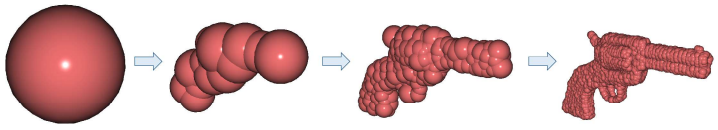

Figure 1. BVH for single 3D object

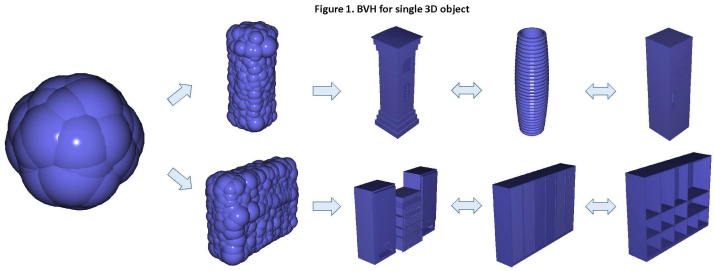

Figure 2. BVH for multiple 3D objects

Three Error Metrics for Comparing Two Sphere Sets

Residual Distance (RD)

For given two sphere sets X and Y, we can find subset of each sphere set that is not included in the other sphere set. We can use volume sum of such subsets for our error metric, and we named it *Residual Distance*(RD). Formally, RD for sphere sets X and Y is defined as :

$$RD(X,Y) = volume(X \cup Y) - volume(X \cap Y)$$

Since exact value of RD is hard to compute, we approximate it by using collision detection algorithm. If we find collision between spheres from each set, we use the collision information to exclude those parts that are included in both sphere sets. After that, we can use only residual parts for our RD measure.

Hausdorff Distance (HD)

For given two geometric entities X and Y, *Hausdorff Distance* (HD) represents the maximum value among minimum distances between the points on those entities. We can apply this concept to sphere sets without any modification. Formally, HD for sphere sets X and Y is defined as :

$$HD(X,Y) = \max\left\{\max_{x\in X}\left\{\min_{y\in Y} d(x,y)\right\}, \max_{y\in Y}\left\{\min_{x\in X} d(x,y)\right\}\right\}$$

In our setting, it is easy to notice that HD never occurs between spheres that intersect with each other. Therefore, we can use residual parts from RD computation for HD computation. To be specific, we can iterate through residual parts of each sphere set and update HD value.

Chamfer's Distance (CD)

For given two point sets $X = \{x_1, \cdots, x_n\}$, $Y = \{y_1, \cdots, y_m\}$, *Chamfer's Distance* (CD) is defined as :

$$CD(X,Y) = \sum_{i=1}^{n} \min_j d(x_i, y_j) + \sum_{i=1}^{m} \min_j d(y_i, x_j)$$

Since data structure we are dealing with is sphere set, we plugged in the set of center points of each sphere set for the point set above. Since this approach does not take account radius of sphere, it may not be a good error metric to use in our context. However, we tested it because we wanted to compare our results against other approach using only random sampling of points on models.

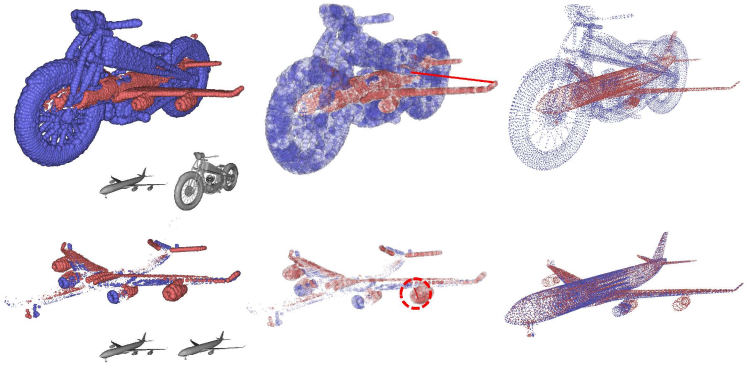

Figure 3. Illustration of error metrics applied to a pair of objects : Residual Distance, Hausdorff Distance, Chamfer's Distance from left to right

Search Algorithm and Example Results

Search Algorithm

When user inputs a 3D model, first we build BVH from it using algorithm described above. After the build is done, we iterate through the DB's tree structure from the root node and find the most similar model. Using BVH in this process can greatly accelerate the entire computation time, because following properties hold :

- We can get concrete upper bound of CD or HD between the query model and the models we are searching for by computing those values between the query model and a certain model (not node) in the DB.
- We can get concrete lower bound of CD or HD between the query model and a subtree rooted at a certain node by computing those values in only one direction : From query model to the node.
- As we go to the lower levels in the DB's tree structure, the lower bound of CD and HD between the query model and the subtree rooted at the node monotonically increase.

By using above properties, we can also infer following property :

- If the lower bound of CD or HD between the query model and a certain node is greater than the upper bound of CD or HD, we do not have to test entire subtree of the node.

Using priority queue and the last property, we can accelerate our entire algorithm much faster.

Figure 4. Upper bound of error can be obtained by computing it with some real models in the database

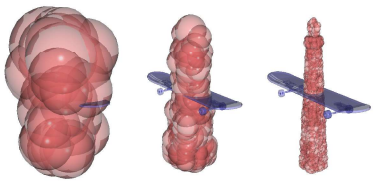

Figure 5. Lower bound of error between query model and a subtree increases as the algorithm goes down the search tree

Search Results

For testing our search algorithm, we used ShapeNetCore dataset. It is composed of 50,000 models from 55 categories. However, due to the lack of time, we used only 10 models from each category and built search tree. We found that our algorithm gives models from same category as the query model in high priorities for many cases in average 2-3 seconds :

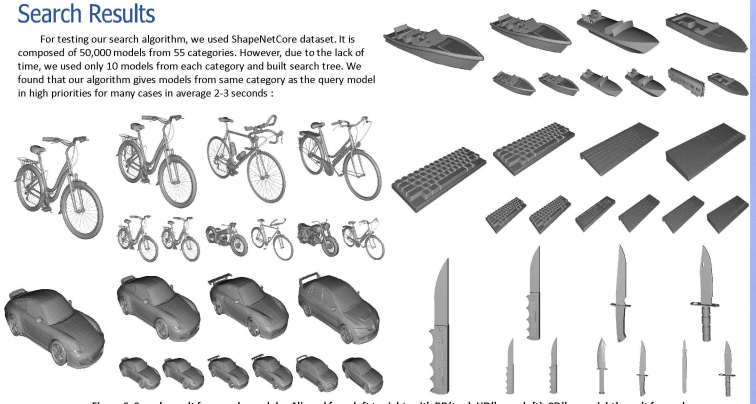

Figure 6. Search result for sample models : Aligned from left to right, with RD(top), HD(lower left), CD(lower right) result for each

Conclusion and Future Work

Algorithm Assessment

From the above results, we can conclude that our algorithm gives accurate results as other algorithms in many cases. Also, we like to point out following advantages that our algorithm has :

- Robust to local perturbation : Even if there is some local perturbation, as long as there is not so much difference in BVH, our algorithm gives same results.
- Intuitive search result : Since our algorithm is based on very intuitive error metrics, search results are also easy to understand. This means that even if our algorithm does not give desired result, we can find out reason why by investigating errors.
- Scalability : As we pointed out, we have theoretical guarantee that this algorithm runs in $O(logN)$, since we use tree structure for entire process. Therefore, we can say that our algorithm will run faster than other algorithms as dataset grows.

However, we also found following problems :

- Needs global alignment : As intuitive as our approach is, it is very weak to transformations. Even though two models are same, if they are in different configurations, our algorithm would fail.
- Limit of static hierarchy construction : For simplicity, we did not implement dynamic update of our search tree. This ended up in severe error accumulation. We found out that current search tree suffers from this problem very much. Therefore, it does not fully utilize power of tree structure.
- Does not consider semantic meaning : In our dataset, categories are defined by semantic meaning. Since our algorithm does not care about such meaning, it gives the model from other category than the query model if it shows smaller error than those from the same category.

What to do next ?

We are planning to complement our research by doing following works :

- Spheres centered on Medial Axis : At the beginning of our project, we aimed at using medial axis of 3D models for efficient and precise representation of BVH. We are going to do this work with flawless models that admit medial axis computation.
- Global Alignment using BVH : We expected that we could also use BVH for global alignment. Even though we found some working examples for this assumption, we realized that this problem itself is another independent problem. If we succeed in solving this problem, we expect to overcome the first limitation stated above.
- Rebalancing search tree : Every time we insert a new model to our search tree, we can detect if there is abnormal error accumulation in some node in the tree. If there is, we have to solve it by inserting intermediate node between those levels or making new branches to lessen the error. This work will help us fully utilize the potential power of tree algorithm.

Also, based on above works, we are going to test our algorithm against different kinds of datasets.

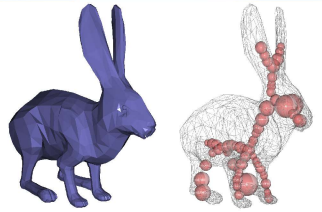

Figure 7. Medial axis of 3D object

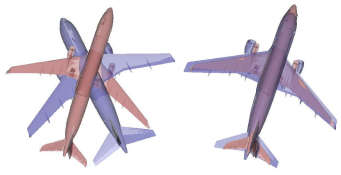

Figure 8. Global alignment of perturbed models