Fast Structural Analysis for Deforming Objects Using Machine Learning 6.850 Project, Spring 2014

Maria Shugrina

MIT CSAIL

Abstract

The motivation of this work is customizing 3D models for fabrication. It is desirable to update the visualization of the model's problematic areas in real time during editing, but often these analysis algorithms are very time consuming. We investigate creating a fast update algorithm for per-vertex model fragility using machine learning. We propose and analyze a new feature vector formulation and present preliminary classification results.

Keywords: computational geometry, structural analysis

1 Introduction

Recently widely available 3D printing technology has made it possible to easily manufacture custom 3D models. Prior to manufacturing, it is desirable to analyze the digital design to ensure that it is unlikely to break. At the very least, such analysis should *detect* problems in the current design and provide a *visualization* of the problematic areas to the user, who can fix them. The inherent appeal of 3D printing is in the ability to customize manufactured 3D models. Because customization is usually an interactive process, a desirable property of all shape analysis algorithms is interactive performance.

Available shape analysis fall into two categories: simulation-based approaches estimating areas of high stress, and purely geometric approaches estimating minimum thickness of the object. Unfortunately, both categories of algorithms involve computationally expensive operations such as conversion to a discrete volumetric representation and medial axis transform, making them infeasible for interactive applications (See Section 2). We argue that for the purpose of interactivity, fast and approximate algorithm is needed, and explore a machine learning approach to this problem.

More concretely, we consider the problem of annotating each vertex of a deforming mesh with a color indicating its fragility in real time. Intuitively, at each deformation step that moves m vertices of the mesh, the complexity of the analysis algorithm should be equivalent to the complexity of the deformation itself: $O(m)$. In Section 3, we provide a machine learning formulation for the problem of updating fragility of a deforming mesh, and explain the query time algorithm. In Section 4, we provide details about the training process, including data used, feature vectors and the classifier. Section 5 concludes with the evaluation and discussion.

2 Related Work

Recent interest in 3D printing has inspired research on automatic analysis of 3D models for printability. A key question that such analysis should answer is whether there are features of the model that are too thin and are likely to break when manufactured.

Special-case Interactive Analysis: Interactive methods for design analysis have been proposed for limited classes of fabricable models. For example, Umetani et al. has proposed interactive methods for designing furniture [Umetani et al. 2012] and clothing [Umetani et al. 2011], while analyzing the design in real time. Recently, Umetani et al have also proposed a fast approximate structural analysis method using approximate cross-secitonal analysis [Umetani and Schmidt 2013]. Our approach takes a similar philosophy, but instead of simplifying analysis based on physically-inspired heuristics, we endeavor to learn the results of principled structural analysis using machine learning. The approaches are very different in spirit, but similar in motivation.

Simulation-based Analysis: Several existing approaches to this problem rely on physical simulation to predict fragile areas. Stava et al. solve for the stress through an FEM simulation of the discretized object volume under heuristically estimated loads and propose a method for correcting the areas of high stress [Stava et al. 2012]. Zhou et al. perform worst-case analysis of the model under load and estimate fragile areas by running modal analysis of the discretized model and maximizing stress [Zhou et al. 2013]. Although these methods produce accurate results, they require costly discretization of the object volume, making them prohibitively expensive to use during interactive mesh deformation.

Medial Axis Transform: Purely geometric methods for estimting minimum thickness have been proposed. The canoncial approach relies on the medial axis tramsform (MAT) [Blum et al. 1967], which approximates the shape with a set of tangent spheres. Computation of MAT is computationally expensive and unstable, but a number of approximate approaches have been proposed, such as [Foskey et al. 2003] and [Giesen et al. 2009]. While well-suited for extracting a shape skeleton, these approximations do not guarantee a good approximation of the MAT for the purpose of minimum thickness computation.

Other Geometric Approaches: Simpler geometric approaches have also been proposed. For example, Telea et al. propose voxelizing the mesh and analyzing the result of dilation and erosion operations for thin features [Telea and Jalba 2011]. Additionally, [Shapira et al. 2008] define a *shape diameter* metric closely linked to the minimum thickness. Though significantly simpler than simulation-based approaches, these methods have not been tested or analyzed for interactive performance. We note that voxelization is an expensive procedure if high precision is desired, and the complexity of the shape diameter metric scales at least linearly with the number of vertices in the mesh (exact complexity is implementation-dependent).

Interactive Deformation: In contrast to shape analysis, interactive deformation techniques have been developed for high-resolution 3D meshes, motivating fast minimum thickness computation. Typically, deformation techniques rely on a small number of control handles, but the computation complexity varies. For example, hierarchical approaches can achieve sublinear performance in the number of vertices, and other approaches such as LBS achieve linear complexity. Ideally, the complexity of minimum thickness computation for a deforming model should be comparable to the complexity of the deformation. In the ideal case, the complexity would vary with the number of deformation handles, not mesh resolution. Investigating reduced representations is the future work of this project, but its current aim is to reduce the complexity of the analysis algorithm to the complexity of the deformation.

Reduced Representation: There are a number of approaches for reducing the dimensionality of the deformation space. Modal analysis is commonly used in graphics to reduce the deformation space from $O(v)$ to $O(r)$, where v is the number of vertices in the mesh and r is the number of principal deformation directions extracted using PCA [Huang et al. 2009]. For instance, [Barbič and James 2010] exploit the structure of this *modal space* to reduce the lower bound for a related problem of self-collision detection from $\Theta(f)$ to $\Theta(r)$, where f is the number of faces and r is as above. In a similar vein, [Pan et al. 2013] use active learning in the *configuration space* (relative object translation, rotation) for computing interpenetration depth between two rigid objects. Another common reduced space is the so-called *pose space* or *shape space* [Lewis et al. 2000] that represents the parameters of the deformation handles which map to vertex displacements in the much larger object space.

3 Method

3.1 Problem

Consider a computationally expensive structural analysis algorithm A that computes a fragility estimate $f(v_i)$ at every vertex v_i of a 3D mesh. We define A in those terms, because the ultimate motivation of interactive stuctural analysis is to give desinger a visual cue of the problems in the mesh in real time during editing. Such visualization is often accomplished through vertex coloring.

For every new mesh G_{rest} that the user would like to deform we can use A to precompute $f_{rest}(v_i)$, the ground truth fragility estimate at every vertex v_i in the rest pose. Ideally, after every deformation $\mathcal{D}$ we would run A on the deformed mesh G_D, obtaining exact fragility $f_D(v_i)$ for every vertex. Because this computation is not feasible in real time, we define our goal as finding $\Delta f(v_i)$, the change in fragility compared to the rest pose at all affected vertices:

$$\Delta f_D(v_i) = f_D(v_i) - f_{rest}(v_i) \tag{1}$$

To come up with an approximation $\tilde{\Delta} f_D(v_i)$ for this value quickly, we formulate the problem in terms of machine learning.

3.2 Machine Learning Formulation

We formulate our problem as learning a regression classifier over a training set of samples $\{\mathbf{x}_j, y_j\}$ across multiple deformed meshes. In our formulation, $\mathbf{x}_j$ is a k dimensional vector defined for a particular vertex in some deformed mesh, and y_j is the ground truth change in fragility at that vertex relative to the rest pose.

After training a Support Vector Regression (SVR) [Drucker et al. 1997] classifier C on the training data, we can obtain an estimate for the change in fragility $\tilde{\Delta} f_D(v_i)$ by evaluating $C(\mathbf{x}(v_i))$ where $\mathbf{x}(v_i)$ is a feature vector for some new vertex v_i from a mesh that need not be in the training set, under a new deformation. We choose not to focus on the Machine Learning aspects of the problem, but treat SVR as existing method known to work. The first key to success of our approach is in selecting appropriate features and training data, not in the choice of the classifier.

3.3 Update Algorithm

Given a regression classifier C, the run time algorithm for updating the fragility values of mesh vertices after a single deformation step is as follows:

Precompute: $f_{rest}(vi), \mathbf{x}(v_i) \quad \forall v_i \in G_{rest}$
for deformation step $\mathcal{D}$ **do**
  for all v_i with changed feature vector $\mathbf{x}(v_i)$ **do**
    recompute $\mathbf{x}(v_i)$
    $\tilde{\Delta} f_D(v_i) \leftarrow C(\mathbf{x}(v_i))$
    fragility at $v_i \leftarrow f_{rest}(v_i) + \tilde{\Delta} f_D(v_i)$

Evaluating C takes $O(1)$, so the running time of this algorithm is $O(w \cdot t_x)$, where w is the number of vertices whose feature vector has changed and t_x is the time to recompute a single feature vector. Ideally, we would like a running time that is polynomial in m, the number of vertices moved by the deformation $\mathcal{D}$, but the actual running time depends on the feature vectors used. We analyze the running time for our particular vectors in Section 5.1.

4 Training

4.1 Data

We used four meshes of varying resolution for training, and two meshes for testing (See Table 1). Each mesh was manually annotated with deformation handles using a hand-built tool (See 4.4). Each training mesh was automatically deformed and sampled.

4.2 Features

We define a feature vector for each vertex v_i relative to its *support vertices*, determined for the rest pose during the precomputation step. During this precomputation, for each vertex v_i with normal vector $\mathbf{n}_i$ we run the following algorithm to compute $supports(v_i)$:

shoot ρ rays in a cone around $-\mathbf{n}_i$
for ray $\mathbf{r_j}$ **do**
  find closest intersected face f_j
$supports(v_i) \leftarrow$ sample k vertices from faces $\{f_1 ... f_\rho\}$
$\mathbf{d}_i \leftarrow$ vector of distances to support vertices

For each vertex, we shoot ρ random rays (used $\rho = 25$) by uniformly sampling the unit circle and then projecting it relative to the vertex position such that all the rays fall inside a 20° cone around the negative normal (See Fig. 1 a). For each ray, we find the closest intersected face and take the union of all the vertices in these faces for all rays. From this set of vertices we uniformly sample k vertices (used $k = 6$) to obtain the support vertices $supports(v_i)$ of vertex v_i (See Fig. 1 b, c). We stack distances from v_i to its supports in the rest pose into a vector $\mathbf{d}_i$.

To compute the actual feature vector at vertex v_i after deformation, we compute the differences in the distances to its support vertices relative to the rest pose (See Fig. 1 d, with deltas shown in red and rest pose distances shown in black). We stack the deltas in a vector and sort the vector from max delta to low delta to obtain a feature vector for vertex v_i in a deformed mesh. The hope is that

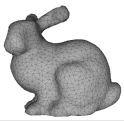
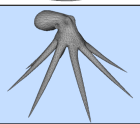
Mesh	vertices	handles	undersupp.	Precomp	samples
	1K	7	60	7min	100 × 30 poses
	1.8K	3	15	16min	200 × 10 poses
	1.8K	3	10	15min	200 × 10 poses
	4K	9	8	1hr 13min	20 × 40 poses
	40K	5	N/A	**too slow!**	N/A
	173K	8	N/A	**too slow!**	N/A

Table 1: *Training and test data used; meshes used only for testing are highlighted in blue. Meshes that I set up for use with Biharmonic weight handles, but which ended up too large for unoptimized ground truth computation are shown in red.*

the distribution of the changes to support vertices will carry information about the relative shift of the opposing faces.[1] Of course, the angle of the cone of rays, the parameter k and the number of sampled rays will affect performance, but rigorous experimentation with these parameters is outside the scope of this project.

Corner cases: It is possible to get no support vertices for vertices on sharp features of the object. We call such feature vectors *malformed* and skip training or evaluating the classifier on these features. It is also possible that the union of all the intersected faces (See Fig. 1 c) has fewer than k vertices; we call such features *undersupported*. In this case, we double-sample some of the vertices. The intuition for that is that if Δd_j for missing supports will be set to any arbitrary value (either zero or some maximum threshold), this will introduce noise into the system, because having Δd_j equal to 0 is very informative. Instead, if you visualize double sampling as simply having two support vertices very close to each other, double sampling strategy is well-justified.

Scale-invariance: To achieve scale-invariance when training on a variety of meshes, the actual feature vector contains *percent* changes, not raw deltas. The ground truth label also represents percent change in the fragility value.

Performance Issues: Simply running Möller–Trumbore intersection algorithm [Möller and Trumbore 1997] for every triangle to determine ray triangle intersections proved prohibitively expensive for meshes with more than 4K vertices. To make this feasible, I started implemented a naive grid-based acceleration structure to cull the faces that are unlikely to be intersected to be able to use the T-rex and armadillo models, but unfortunately ran out of time. Hence, the variety of training and test data leaves more to be desired.

4.3 Ground Truth

For this project, we used a variant of shape diameter [Shapira et al. 2008] as a proxy for shape fragility at a vertex (See Fig. 2). [2] Of course, using a more sophisticated ground truth function such as FEM simulation of the object under gravity would be more compelling.

Our shape diameter implementation is similar to the feature computation. To find shape diameter fragility f_i at a vertex v_i, we run the following algorithm:

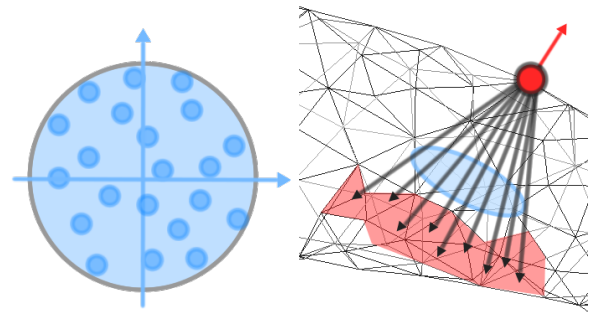
shoot ρ rays in a cone around $-\mathbf{n}_i$
for ray $\mathbf{r_j}$ **do**
    find distance D_j to the closest intersected face
$f_i \leftarrow$ weighted sum of D_j's

Instead of discarding the outliers and taking the median of the distances as in [Shapira et al. 2008], we take a weighted sum of the ray lengths. The weight is computed using a 2D standard normal distribution with the position of every sampled ray taken from the sample point in the unit circle used to sample it (See Fig. 1 a). Thus, distances closer to the normal direction take on more weight. Because this method for estimating ground truth mostly serves as a starting point, we have not looked into other weighing schemes. For future work, we plan to replace shape diameter with physical simulation.

Labeling the training data: The actual label we are learning is

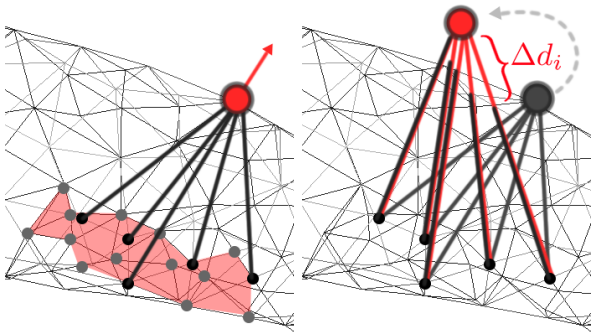
[1] This feature vector is my invention and is inspired by histogram-based shape discriptors. I hope to run more experiments using this feature vector in the future.

[2] Originally, I planned to use the open source version of shape diameter, but it was too difficult to get compiled and refactored into a usable form, so I have reimplemented it.



(a) *Sample rays in a cone around* $-\mathbf{n}$ *using unit circle.*

(b) *For each ray find the closest intersected face.*



(c) *Sample* k *vertices out of vertices of the intersected faces, and save the distances to support vertices for rest pose.*

(d) *After deformation, deltas* Δd *in distances to support vertices form the feature vector.*

Figure 1: *Computing the feature vector, steps a-c occur during precomputation for the rest pose, and step d occurs after deformation.*

the difference in fragility between the rest pose and deformed pose. Thus, every sample in the training data is a feature vector at a given vertex, and its label y is the signed difference between rest pose fragility and fragility at that vertex in the deformed mesh.

Visualization: The ultimate goal is to visualize fragility of the deforming mesh in real time. To visualize shape diameter, we normalize fragility at each vertex by the maximum fragility in the rest pose and map this value to hue in the HSV color system. We experimented with polynomial and sigmoid mappings of fragility to hue, but found that uniform mapping is most intuitive. Blue corresponds to low fragility (large shape diameter), green and yellow to medium values, and red corresponds to high fragility (small shape diameter) (See Fig. 3).

4.4 Deformation Model

Because sampling across the range of deformed models is part of the training process, the choice of the deformation model is significant. For example, if the deformations sampled involved randomly perturbing certain vertices, the classifier would not have any data to learn about more smooth and intuitive deformations.

For our work, we train and test on the same class of deformation. We have built a tool to place point controls inside a 3D mesh, and have manually added point controls to all the training and test meshes (See Table 1). Then, we used open source implementation of Bounded Biharmonic defomration weights [Jacobson et al. 2011] to calculate the influence of each point control on each vertex of the mesh (See Fig. 4). Each control has 6 digrees of freedom, and it effects a similar transformation on the neighboring vertices, proportional to its influence.

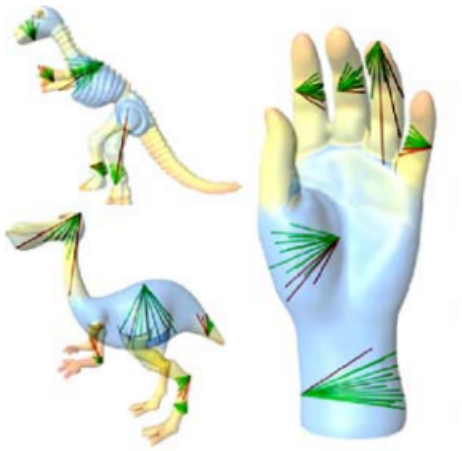
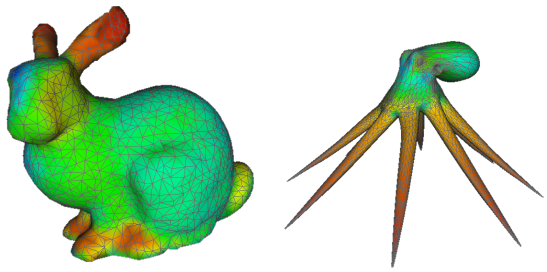


Figure 2: *Computing the shape diameter function; image from [Shapira et al. 2008].*



(a) *Bunny.* **(b)** *Octopus.*

Figure 3: *Visualization of the shape diameter computed using our implementation. Observe that the thin features such as ears and tentacles appear red.*

We have chosen this deformation method because it tends to provide smooth and intuitive results, and fragility transformations under such well-behaved deformation may be easier to learn.

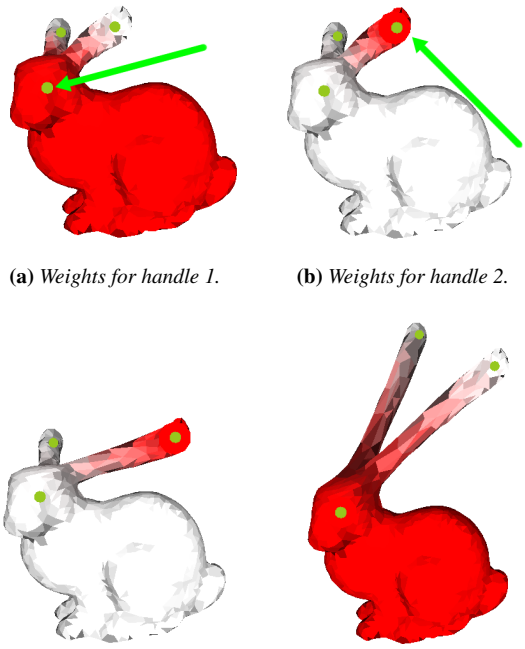
4.5 Classifier

We use SVR regression classifier available as a part of libSVM package [Chang and Lin 2011].

5 Evaluation

5.1 Complexity

In this section we analyze the complexity of the update algorithm in Section 3.3. The time to recompute the feature vector for a single vertex takes time $O(k \log k)$, because the k distance deltas need to be sorted. For a constant k, we are mostly concerned with bounding the number of vertices for which feature vectors need to be recomputed after a deformation step that moves m vertices. Thus, we need to bound the number of vertices with moved vertices as their supports.



(a) *Weights for handle 1.* **(b)** *Weights for handle 2.*

(c) *Deformation from handle 2.* **(d)** *Deformation from multiple handle transofrms.*

Figure 4: *Visualization of the biharmonic deformation weights for the bunny model with 3 deformation handles. The deformation handles were placed inside the object using a tool we built. Differnt deformation handles are selected in different screenshots (see green arrow), and red coloring corresponds to the influence of that handle on the vartices.*

5.1.1 Theoretical

In the worst case, one of the moved vertices could support all of the mesh's vertices, resulting in run time complexity $O(vk \log k)$. The expected complexity is much better, however. Consider a directed graph where an edge from vertex A to B means that B supports A (See Fig. 5). We are interested in the expected number of incoming edges for a randomly chosen set of m vertices. Although the graph has variable indegree, its outdegree is a constant k for every vertex. Thus, on average, the number of incoming edges for a set of m vertices will be the same as the number of outgoing edges: mk. This means that expected run time of the update algorithm is $O(mk^2 \log k) \approx O(m)$, meeting our initial goal of getting the same complexity as the complexity of the deformation itself, i.e. $O(m)$.

5.1.2 Empirical

In the previous section, we relied on the assumption that the set of moved vertices was chosen uniformly at random. This is certainly not the case for most deformations that occur in practice. To explore this dependance, we have sampled various numbers m of vertices in four test meshes and computed the number of feature vectors that would have to be recomputed if these m vertices moved. We selcted m vertices using three different deformation techniques:

1. Biharmonic handle deformation - see Section 4.4.
2. Random vertex deformation - m vertices are selected at random.

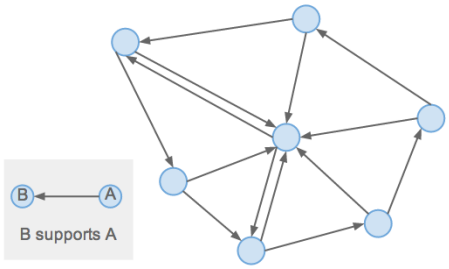


Figure 5: *Graph of vertices and their supports with a constant outdegree of k.*

3. Local deformation - m vertices are selected by picking a random vertex and growing selection region around it by following edges until m vertices are selected.

The plots of Number of Feature vector recomputations vs. m are in Fig. 6.

The number of feature vector recomputations scales linearly for Biharmonic weight deformation, as predicted assymptotically. We conjecture that this is likely because this deformation model is very smooth and tends to affect volumetrically contiguous regions of the mesh at the same time. Other two deformation models stray from the assymptotic predictions, as, of course, actual number of feature vector recomputations depends on k, which is non-negligible in practice. As expected, Random vertex deformation saturates the number of feature vector precomputations to the total vertex count more quickly, as all regions of the mesh are affected by this deformation uniformly at random (See Fig. 6 b). Local deformation tends to require fewer re-computations, but the plota are noise: as would be expected, the number of recomputations depends on which location is picked for the local deformation.

We note that although the number of feature vector recomputations may reach the total number of vertices, the complexity and run-time of the fragility update algorithm is still much lower than that of the structural analysis algorithms listed in Section 2.

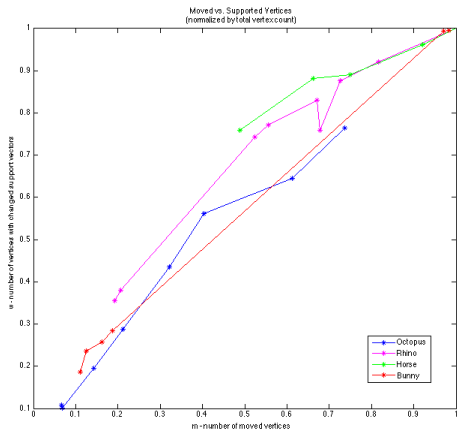
5.2 Accuracy

Evaluation set	Mean Abs Error	Stdev Abs Error
Training Set	8.723	14.86
Testing Set	5.7669	9.34

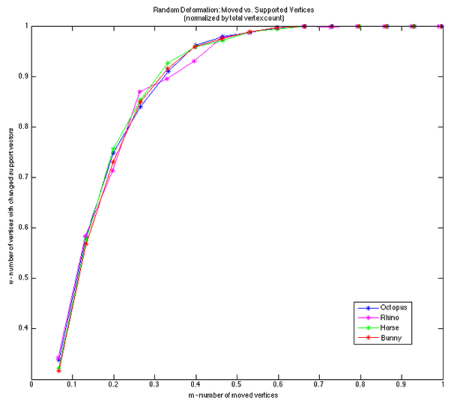
Table 2: *Error on the initial classifier.*

As discussed earlier, more training data is necessary to give credibility to these results. We also suspect that the shape diameter ground truth function itself is quite noisy, and a more deterministic ground truth function would be easier to learn. There are other aspects of this method that need attention:

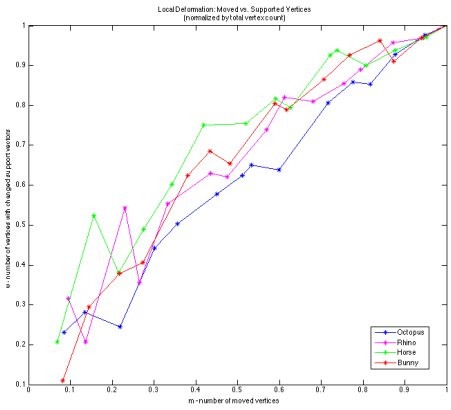
1. Training meshes must be diverse
2. Deformation sampling must be diverse and encompas representative deformations
3. Ground truth function should be less susceptible to noise (shape diameter is itself an approximation)
4. Vertex sampling for each pose should ensure that the deltas in fragility are well represented



(a) *Biharmonic Handle deformation.*



(b) *Random vertex deformation.*

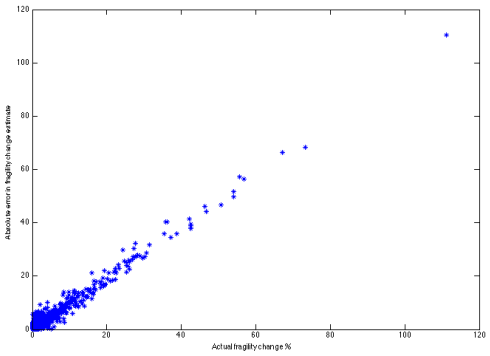


(c) *Local deformation.*

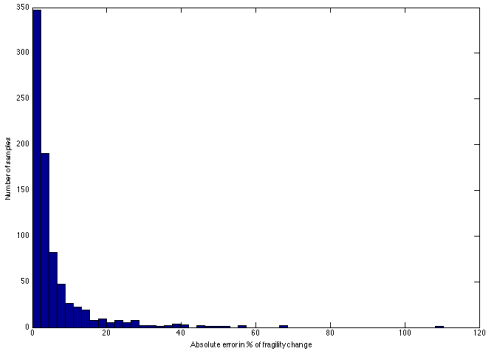
Figure 6: *Empirically measured w, number of vertices with changed support vectors, for various meshes and numbers of moved vertices m using three different deformation models.*

Therefore, all results need to be taken with a grain of salt, and there is certainly evidence indicating that something is not right.

Still, we have trained a classifier on the samples obtained from three training meshes in Table 1, and have evaluated the classifier on the



(a) *Absolute error vs. Ground truth.*



(b) *Distribution of absolute error.*

Figure 7: *Results of the initial classifier on the test set. In **a**, large errors tend to occur when the fragility changes by a really large percent. In **b**, we see the distribution of absolute errors: although most errors are small, some are extremely large.*

training data and on the test data from the octopus model. We evaluate our classifier in a scale-invariant manner. First we let ground truth y be the *percent by which the fragility actually changed* after deformation. The classifier predicts this value with a y' for each test sample.

In Table 2 we summarize statistics about absolute error $|y - y'|$ for test and train sets. Paradoxically, the error on the test set is lower. We hypothesize that this is due to abysmally small number of training meshes. Still, the mean error looks fairly small. To gage how useful such a classifier would be, we plot actual error against the ground truth estimate in Fig. 7 a. It turns out that larger errors tend to occur where the fragility changes a lot. Glancing at the training data, it seems that there are not enough samples where the fragility value is changing much, so there is probably not enough data to learn that dependance. Finally, Fig. 7 b shows the distribution of absolute errors. Although most errors are small, some are exteremely large, rendering classifier impractical without performance improvements.

5.3 Discussion

It is hard to gage the method's usefulness from these very preliminary results. The fact that the error distribution is skewed toward zero (Fig. 7) gives hope that this method may turn out positive results with more care devoted to preparing training set, sampling de-

formations and ensuring that the training data covers a wide range of fragility changes.

We believe that this approach is very promising in terms of its runtime and are enthusiastic about applying it to estimating results of physical simulation, which tends to be prohibitively slow for interactive applications.

References

BARBIČ, J., AND JAMES, D. L. 2010. Subspace self-collision culling. In *ACM Transactions on Graphics (TOG)*, vol. 29, ACM, 81.

BLUM, H., ET AL. 1967. A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form 19*, 5, 362–380.

CHANG, C.-C., AND LIN, C.-J. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2*, 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

DRUCKER, H., BURGES, C. J., KAUFMAN, L., SMOLA, A., AND VAPNIK, V. 1997. Support vector regression machines. *Advances in neural information processing systems 9*, 155–161.

FOSKEY, M., LIN, M. C., AND MANOCHA, D. 2003. Efficient computation of a simplified medial axis. *Journal of Computing and Information Science in Engineering 3*, 4, 274–284.

GIESEN, J., MIKLOS, B., PAULY, M., AND WORMSER, C. 2009. The scale axis transform. In *Proceedings of the 25th annual symposium on Computational geometry*, ACM, 106–115.

HUANG, Q.-X., WICKE, M., ADAMS, B., AND GUIBAS, L. 2009. Shape decomposition using modal analysis. In *Computer Graphics Forum*, vol. 28, Wiley Online Library, 407–416.

JACOBSON, A., BARAN, I., POPOVIC, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4, 78.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 165–172.

MÖLLER, T., AND TRUMBORE, B. 1997. Fast, minimum storage ray-triangle intersection. *Journal of graphics tools 2*, 1, 21–28.

PAN, J., ZHANG, X., AND MANOCHA, D. 2013. Efficient penetration depth approximation using active learning. *ACM TRANSACTIONS ON GRAPHICS 32*, 6.

SHAPIRA, L., SHAMIR, A., AND COHEN-OR, D. 2008. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer 24*, 4, 249–259.

STAVA, O., VANEK, J., BENES, B., CARR, N., AND MĚCH, R. 2012. Stress relief: improving structural strength of 3d printable objects. *ACM Transactions on Graphics (TOG) 31*, 4, 48.

TELEA, A., AND JALBA, A. 2011. Voxel-based assessment of printability of 3d shapes. In *Mathematical Morphology and Its Applications to Image and Signal Processing*. Springer, 393–404.

UMETANI, N., AND SCHMIDT, R. 2013. Cross-sectional structural analysis for 3d printing optimization. In *SIGGRAPH Asia 2013 Technical Briefs*, ACM, New York, NY, USA, SA '13, 5:1–5:4.

UMETANI, N., KAUFMAN, D. M., IGARASHI, T., AND GRINSPUN, E. 2011. Sensitive couture for interactive garment modeling and editing. *ACM Trans. Graph. 30*, 4, 90.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph. 31*, 4, 86.

ZHOU, Q., PANETTA, J., AND ZORIN, D. 2013. Worst-case structural analysis. *ACM Transactions on Graphics (TOG) 32*, 4, 137.