NBD-Tree: Neural Bounded Deformation Tree for Collision Culling of Deformable Objects

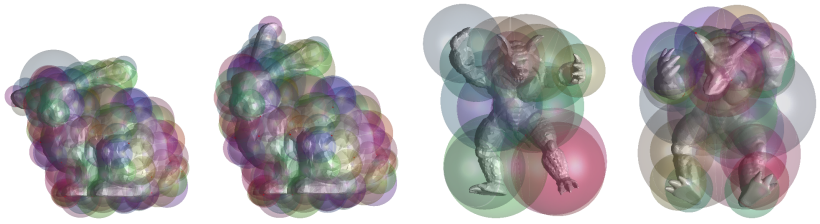
Ryan S. Zesch[1](✉), Bethany R. Witemeyer[1], Ziyan Xiong[1], David I.W. Levin[2], and Shinjiro Sueda[1]

[1] Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA
rzesch@tamu.edu

[2] Department of Computer Science, University of Toronto, Toronto, Ontario, Canada

Abstract. We propose a novel machine learning-based approach for accelerating the broad phase of 3D collision detection for deformable objects. Our method, which we call the neural bounded deformation tree (NBD-Tree), allows us to cull away primitives for full-space deformable objects quickly. Unlike its classic, non-neural counterpart, the NBD-Tree is not limited to deformable objects that are constrained to work within the space of low-dimensional deformation modes, and instead works with an arbitrary set of deformations. With our approach, when the shape of the object changes at runtime, we use the low-dimensional deformation modes of the object only as the input to a neural network that calculates the necessary updates to the NBD-Tree. To further improve efficiency, we approximate these low-dimensional modes efficiently through clustering, which allows us to avoid going through every vertex of the mesh. We then rely on the network to overcome the potential errors stemming from these approximations. The NBD-Tree paves the way for interactive collision culling of large-scale, full-space deformable objects.

Keywords: Collision Detection · Neural Network · Broad Phase · Bounding Volume Hierarchy · Sphere Tree



(a) (b) (c) (d)

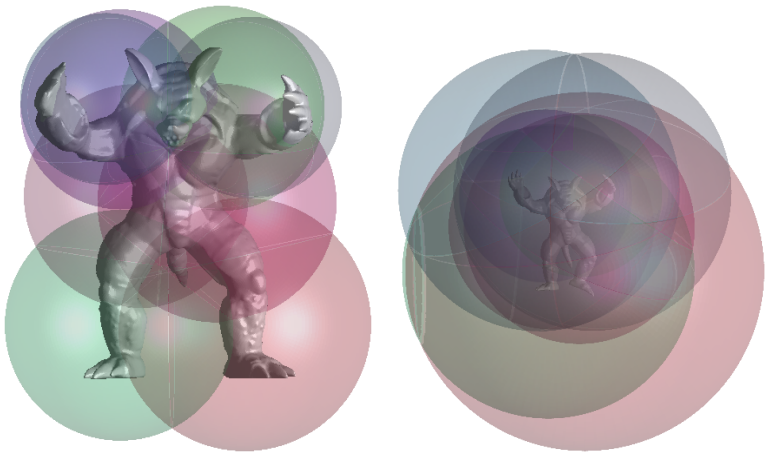
Fig. 1: The Neural Bounded Deformation Tree allows us to quickly compute a sphere tree for full-space deformable objects.

1 Introduction

3D collision detection is an important component of various applications in such fields as computer graphics, computer vision, and robotics, but an efficient approach for deformable objects remains a challenge [22]. In most applications, the collision detection pipeline is usually divided into two parts: the broad-phase to quickly cull away primitives that are far from being in contact, and the narrow-phase to go through the remaining primitives to compute the actual collisions [4]. In this paper, we focus on broad-phase collision detection. In particular, we work with a sphere tree, which is one of the most commonly-used techniques for broad-phase collision detection. With a sphere tree, we surround the 3D object with a hierarchical set of spheres. If the root sphere of a 3D object, which contains all of the primitives of the 3D object, does not intersect the root sphere of another 3D object, then we know that the two 3D objects are not intersecting; otherwise, we recursively check the children spheres of the two sphere trees.

For objects that are constrained to deform linearly based on a set of modes [21], the classic bounded deformation Tree algorithm (BD-Tree) is still arguably the best choice for the broad-phase [9]. The BD-Tree algorithm starts with a sphere tree built from the rest pose of the deformable object, and then during runtime, it updates the sphere centers and radii in time linear in the number of modes, regardless of the number of vertices/triangles in the mesh. In some situations, however, using the full space, as opposed to the linear modes, of deformations is desired or necessary, especially when hard constraints are present, as they can cause locking with reduced deformations.

Unfortunately, BD-Trees cannot be used for full-space deformations, since its update equations depend on the linearity of the deformations—large full-space deformations cause BD-Tree bounds to be extremely conservative. Therefore, we propose the neural bounded deformation tree (NBD-Tree), which uses neural networks to update the sphere tree for full-space deformable objects. At runtime, we compute the low-dimensional modes of the deformation, which are passed through a network to compute the corrections to be applied to the rest pose sphere tree. We use a small multi-layer perceptron (MLP) for each sphere of the tree, making the network evaluation very fast. However, the process of computing the low-dimensional modes can then become a bottleneck for a large mesh because: first, we need to compute the rigid alignment of the mesh to go to the local transformed space of the trained network, and then we need to perform a matrix-vector multiplication between the modal matrix and the transformed vertices to compute the low-dimensional modes. Unfortunately, both of these operations are linear, requiring us to go through every vertex of the mesh. Therefore, we compute the rigid alignment and the matrix-vector multiplication via a clustering approach to quickly compute the approximate low-dimensional mode, and then rely on the network to overcome the potential errors stemming from the approximations. This allows us to compute the modes quickly, thereby making the whole pipeline highly efficient.



(a) (b)

Fig. 2: Even at a mild deformation (b) from the rest pose (a), a standard BD-Tree is vastly overly conservative when using a small number of modes of deformation.

2 Related Work

3D collision detection has been an active area of research in many fields including graphics, robotics, and vision, with several survey papers spanning multiple decades [8, 11, 22]. We refer the reader to these excellent surveys for an overview of various techniques. One of the most popular approaches for deformable objects collisions is a bounding volume hierarchy (BVH). If the modes of deformation are known *a priori*, then the individual bounding volumes in a BVH can be updated very efficiently [9]. However, as mentioned in the introduction, these fast updates do not work on full-space models. Image-based methods work well with deformable objects and naturally run on the GPU [5, 23], but these methods cannot be readily incorporated into other simulation frameworks. Another approach is to deform a signed distance field (SDF) based on the object's mesh [6, 12, 13]. However, with these methods, a BVH is still required to find the region or the cell that contains the query point. Deformed SDFs have also been used for deformed sphere tracing and simple collision detection [20], but such methods have limited applicability to general collision detection because they cannot evaluate the underlying implicit surface at an arbitrary point in deformed space.

Recently, approaches based on neural fields have become extremely popular [24]. Of these, implicit shape representation through occupancy or signed distance fields are highly relevant to collision detection. Park et al. [16] showed that, with their Coded Shape DeepSDF approach, they can build a highly effective implicit representation of non-rigid 3D geometry. Concurrent work by Mescheder et al. [14] and Chen and Zhang [2] used neural networks for occupancy fields. All of these works use neural approaches for various visual applications, such as shape completion, interpolation, and 3D reconstruction. There are also neural

approaches that are specialized for articulated characters [1, 3, 19], including implicit collision handling with posed characters. Our work is orthogonal to these approaches, focused on general volumetric collision handling.

In the BD-Tree algorithm [9], which we base our work upon, a sphere tree for a deformable model is updated based on its current reduced space deformation. A rest-pose sphere tree is precomputed using a wrapped hierarchy, in which each sphere contains the enclosed geometry of its children, but not necessarily the bounding spheres of its children. At runtime, a sphere tree node has its center and radius updated using precomputed quantities derived from the model's displacement field and the current set of reduced space coordinates. This method is output-sensitive in that a sphere has its center and radius updated only if its parent sphere is in collision, reducing the number of computations needed. A central drawback of this method, however, is that large deformations and deformations not captured in the reduced space have very conservative bounds, as seen in Fig. 2.

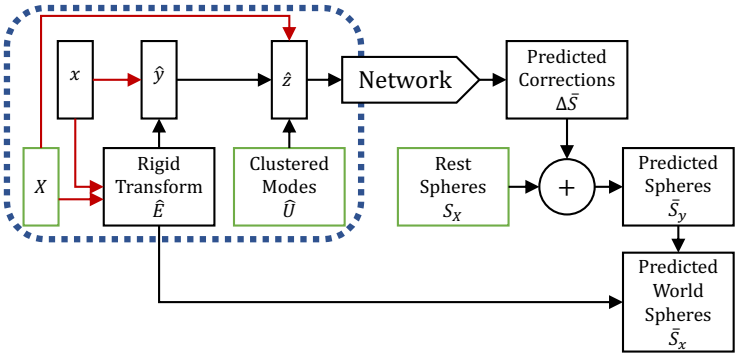
3 NBD-Tree Algorithm

3.1 BD-Tree Overview

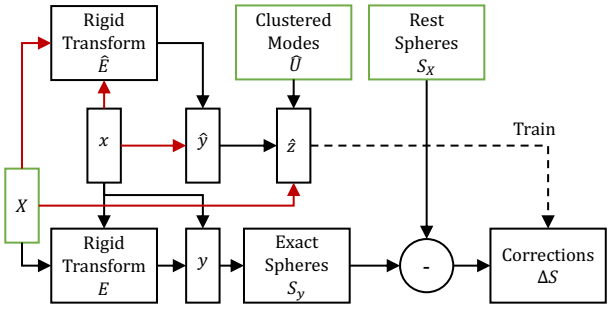
With a standard sphere tree, whenever the 3D object changes its shape, the sphere centers and radii are updated by checking all the primitives of the 3D object. If we constrain the deformation to only linear modes, we can instead use the Bounded Deformation Tree (BD-Tree) algorithm [9]. This work is based upon the fact that a deformed model $x \in \mathbb{R}^{3n}$ which has rest pose $X \in \mathbb{R}^{3n}$ can be approximated as $x = X + Uq$, where $U \in \mathbb{R}^{3n \times m}$ is the model's displacement field and $q \in \mathbb{R}^m$ is a vector of reduced space coordinates (*i.e.,* linear modes) for the current deformation. In the BD-Tree, each node of a sphere tree can be updated to the model's current deformation independently by using precomputed values for that model along with the linear modes for the current deformation. In particular, a sphere's center and radius are updated as $c' = c + \bar{U}q$ and $r' = r + \Delta r^T \text{abs}(q)$, where $\bar{U}$ and Δr are precomputed matrices derived from U. The updated spheres computed via these matrices are guaranteed to contain the deformed model, but if deformations are not well approximated by U, BD-Tree bounds become extremely conservative. In our method, we address these overly conservative bounds by using a learning based approach, which we call the Neural Bounded Deformation Tree (NBD-Tree).

3.2 NBD-Tree Overview

An overview of the online portion of the NBD-Tree pipeline of is shown in Fig. 3a. At every frame of the simulation, given the current vertex positions x, our goal is to update the predicted (binary) sphere tree $\bar{S}_x$. (We use a bar above S to indicate that this is the *predicted* quantity computed by evaluating the network.) This is done through a sequence of steps, including rigid alignment ($\hat{y}$), code



(a) Online pipeline



(b) Training pipeline

Fig. 3: Training and online pipelines. Green boxes are the quantities precomputed in the preprocessing pipeline, shown in Fig. 4. Red arrows imply that a subset of the quantity is used to generate a reduced quantity. In the online pipeline, the portion inside the dashed blue rectangle is performed once for the entire tree, whereas the other portions are evaluated multiple times to update the spheres as necessary.

generation ($\hat{z}$), and network evaluations. (We use the hat notation to indicate quantities that result from clustering, which we describe later.) The output of the network is a correction $\Delta\bar{S}$ to be applied to S_X, the sphere tree created during the preprocessing stage with the rest pose vertex positions X, shown in Fig. 4. We follow the original BD-Tree approach of using a *wrapped*, instead of *layered*, hierarchy [7, 9]. Like the BD-Tree algorithm, our method is output-sensitive in that we update the sphere tree nodes only if necessary. For instance, if the top-level sphere does not return a collision, we do not update any of the descendant spheres. In Fig. 3a, this is indicated by the dotted rectangle in dark blue. The portion of the pipeline within this rectangle is performed once for the entire tree, whereas the remaining portions are evaluated for a sphere if its parent sphere reported a collision.

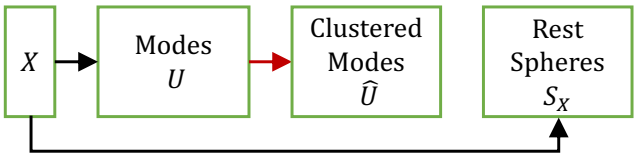


Fig. 4: Preprocessing pipeline. The red arrow implies that a subset of rows of U is used to generate the reduced modal matrix $\hat{U}$.

To compute the predicted corrections, we use a neural network trained for each sphere. The data generation pipeline for training is shown in Fig. 3b. We take a set of training poses x, and, following a series of steps described in more detail below, we generate the code $\hat{z}$ and the corresponding sphere corrections ΔS. This mapping between $\hat{z}$ and ΔS is learned by the network and is used in the online pipeline.

In the rest of this section, we describe these steps in more detail.

3.3 Rigid Alignment

The first step in both the training and online pipelines (Fig. 3) is to rigidly align the current vertex positions x to best match the rest vertex positions X. We will first consider the exact rigid transform $E \in \mathrm{SE}(3)$ in Fig. 3b, ignoring the red arrows and the approximate rigid transform $\hat{E} \in \mathrm{SE}(3)$, which we will describe shortly. Given a training pose x, we compute the rigid transform E using the method described by Müller et al. [15]. This rigid transform then allows us to compute the aligned vertex positions:

$$y = Ex. \tag{1}$$

Using these aligned vertices, we construct the exact spheres S_y in the local aligned space. Using the local space is helpful for training the network, since then the network would not need to learn the rigid transforms.

For a large mesh, computing the best rigid transform can become a bottleneck, since we must go through every vertex of the mesh. We therefore use a pre-selected subset of vertices to efficiently compute the approximate rigid transform $\hat{E}$. (In Figs. 3 and 4, we indicate all of the steps that use the pre-selected subset with red arrows. All the arrows share the same subset.) We then use the same subset of vertices from x to form $\hat{x}$ and transform them by $\hat{E}$:

$$\hat{y} = \hat{E}\hat{x}. \tag{2}$$

In the next subsection, we discuss how we choose this subset of vertices.

3.4 Clustered Modes

Along with the rest sphere tree S_X, we construct the modal matrix U in the preprocessing stage (Fig. 4). This matrix can be constructed in a number of

ways, as described by Sifakis and Barbic [21], but in our implementation, we use linear modes based on the mass and stiffness matrices of the volumetric object [18]. The modal basis matrix U, regardless of how it was constructed, is a tall and skinny matrix; if there are n vertices, then U is a $3n \times m$ matrix, where $m \ll n$. (In our implementation, we use $m = 128$ columns.) We can use $U^\top$ to transform from the (aligned) full space y to the modal space z:

$$z = U^\top (y - X). \tag{3}$$

Unfortunately, these operations again require us to iterate over all of the vertices.

Therefore, as a preprocessing step, we cluster the rows of U to form a clustered modal matrix $\hat{U}$ (red arrow in Fig. 4). First, we reshape U so that a single row corresponds to a 3D vertex, rather than a single coordinate, which makes U be $n \times 3m$. We then cluster the rows of this reshaped U via k-means using the standard L^2 metric. After clustering, we find the representative vertex in each cluster that is closest to each centroid. The indices of these k representative vertices become the *pre-selected subset* for efficiently computing the rigid alignment and the modes. Taking the rows corresponding to these k vertices and reshaping, we form the $3k \times m$ matrix $\hat{U}$. To compute the clustered modal space vector $\hat{z}$, we apply a per-cluster weight before multiplying by $\hat{U}^\top$:

$$\hat{z} = \hat{U}^\top \hat{w} \odot (\hat{y} - \hat{X}), \tag{4}$$

where $\hat{w}$ is a weight vector composed of the number of elements in each cluster, and $\odot$ denotes a component-wise multiplication between two vectors.

Since the result of k-means depends on initialization, we run the clustering algorithm multiple times and choose the result that gives the smallest average L^2 distance between z and $\hat{z}$ across all training poses. The inevitable error that comes from the clustered alignment and modes will be remedied by the network, which we describe next.

3.5 Training

The training poses are generated by running an FEM simulation of the volumetric object with various initial and dynamic conditions. As shown in Fig. 3b, for each pose x, we compute the full space rigid alignment y, the clustered rigid alignment $\hat{y}$, and the code $\hat{z}$ as described in §3.3 and §3.4. For each tree node, we compute the target bounding sphere $S_y = \{c_y, r_y\}$ around y, where the sphere is defined by its center c_y and radius r_y. We then subtract the rest sphere S_X from S_y to compute the corrections ΔS:

$$\Delta c = c_y - c_X, \quad \Delta r = r_y - r_X. \tag{5}$$

Our neural network will learn a mapping from the clustered modal codes to the sphere corrections: $\hat{z} \mapsto (\Delta c, \Delta r)$. This allows us to quickly reconstruct the sphere without having to visit every vertex of the mesh.

As a preprocessing step, we normalize the range of $\hat{z}, \Delta c$, and Δr to ± 1 element-wise. For each tree node, we train a small MLP. Each MLP uses two

Table 1: Meshes used for our experiments.

Mesh	# Nodes	# Faces	# train	# test
Bunny	5988	6244	10k	2.7k
Arma	10518	16204	10k	2k
ArmaHD	32410	64816	6k	1k
ArmaUHD	129634	259264	850	100

fully connected layers with 32 neurons and ReLU activation functions between layers. We use a simple L^2 loss as our loss function. We train all networks in parallel as a single model, and extract the weights and biases for each tree node as a post-processing step.

3.6 Runtime

The runtime pipeline (Fig. 3a) has two stages: code computation and sphere updates. In each timestep, we first compute the code $\hat{z}$ as described in §3.3 and §3.4. Next, we recursively update, starting from the root node. Each node's two children are updated only if the node is itself in collision. Finally, if a leaf node is found to be in collision, it passes its stored list of triangles back to be handled by a narrow phase collision detector.

The update procedure is the same for all spheres. We first normalize $\hat{z}$ component-wise and pass it through the network for the sphere. The output of the network is then unnormalized component-wise, which gives us the predicted corrections $\Delta\bar{S} = \{\Delta\bar{c}, \Delta\bar{r}\}$. These predicted corrections are applied to the rest sphere $S_X = \{c_X, r_X\}$ to compute the predicted sphere $\bar{S}_y$ in the local aligned space:

$$\bar{c}_y = c_X + \Delta\bar{c}, \quad \bar{r}_y = r_X + \Delta\bar{r}. \tag{6}$$

Finally, this sphere is transformed via $\hat{E}$ to give us the world space sphere $\bar{S}_x$.

4 Results

All networks in our pipeline were trained in PyTorch [17] on dual Titan-RTX graphics cards. We use the Adam optimizer with a learning rate of 5×10^{-3} with plateau decay [10]. We found that lower batch sizes yield better results, and used a batch size of 128. All networks are trained to convergence. For each tree node, we use an MLP with 2 hidden layers and 32 neurons per layer. We found the benefits of any network larger than this size to be marginal. For all our experiments, we use 128 linear modes, and in our modal clustering, we use 512 cluster points. All of our online code is written in C++ with Eigen, including MLP evaluations. We test our method on four meshes, as listed in Table 1.

All networks for a tree are trained in parallel with an L^2 loss. We experimented with other loss functions, including a loss which highly penalized when predictions do not contain the ground truth nodes. In practice, we found that

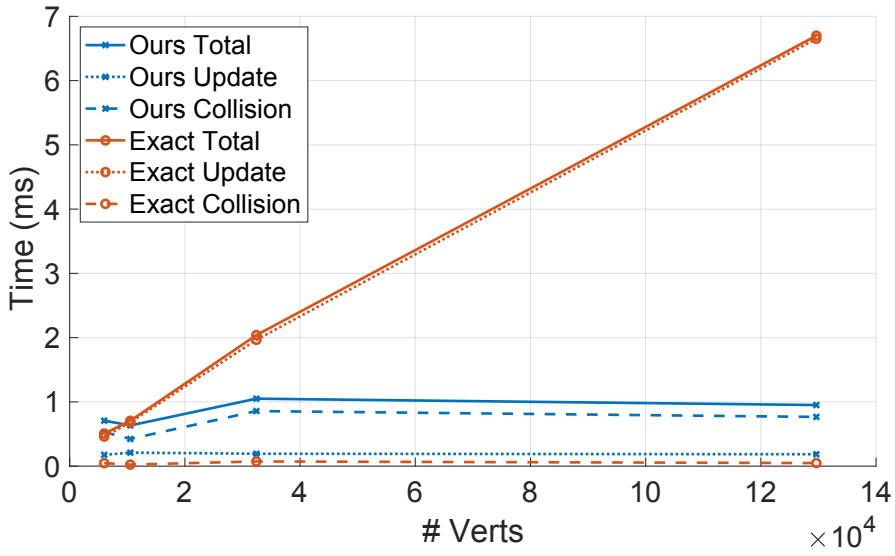


Fig. 5: Comparison of performance between Our method and Exact sphere trees. Total: the amount of time it takes to run the online pipeline for two colliding objects. Update: the amount of time to compute the code by our methods or to update the spheres by the exact. Collision: the amount of time spent computing sphere-sphere collisions.

the performance benefit of such a loss was no better than simply increasing predicted radii by 1%, in terms of percentage of vertices protruding from their predicted nodes.

When we cluster the modal matrix U, we experience a code reconstruction error of at most 5%. Because this error is systematic, our networks are able to overcome it and produce results on par with unclustered methods. We additionally find that increasing the radii of a clustered model by 1-2% will result in vertex containment better than its unclustered counterpart.

Fig. 5 shows the timing results for colliding two objects. For these tests, we used a collision scenario that occurs often in practice with physics based modeling: with the two objects touching at a few places but with no deep interpenetrations. We compare our method against a traditional bottom-up 'exact' sphere tree. With the exact sphere tree, we need to scan the whole mesh to update the tree (red dotted line). On the other hand, with our method (blue dotted line), we are able to quickly compute the code $\hat{z}$ using the clustering approach—no matter how large the mesh is, as long as $\hat{U}$ is the same size, it takes the same amount of time to compute the code (blue rectangle in Fig. 3a). This is a reasonable assumption when working with different resolutions of the same object, since the modes are likely to be similar, and the vertex clusters

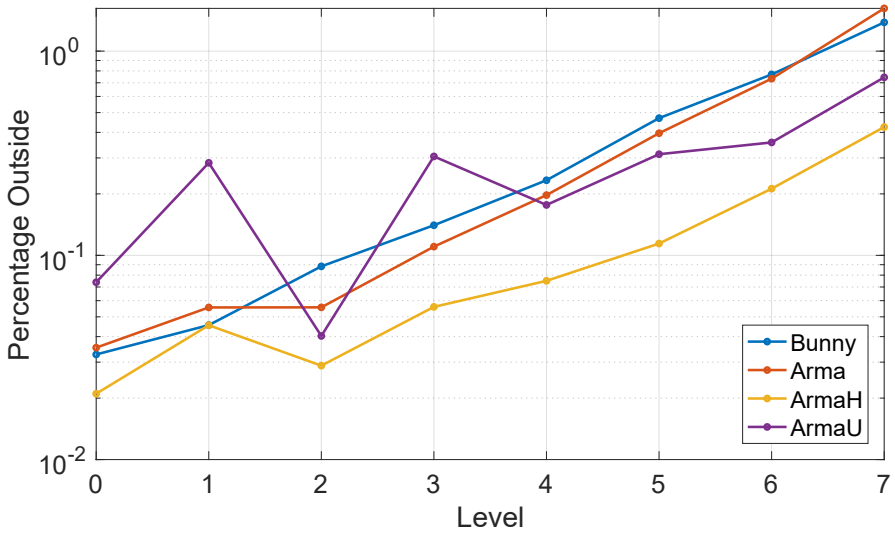


Fig. 6: The percentage of vertices that are outside the spheres at each level (*i.e.*, $10^{-1} = 0.1\%$).

are also likely to be similar. The actual colliding of the spheres (dashed lines) is faster with the exact approach, because once the spheres are constructed, the collision check only involves the cheap distance check between spheres. Our method is required to evaluate an MLP to compute the center and the radius, and so it is relatively more expensive to perform these checks. In essence, the long-term trajectory of our approach does not depend on the vertex count but on the modes and the clusters, all the while being able to account for full-space deformations due to the neural corrections.

We find that our trained networks learn bounds which contain almost all vertices across our test sets (Fig. 6). At the root level, almost every vertex is contained, with at most 0.1% protruding from the nodes. Furthermore, we find that almost all vertices protruding from their respective bounding spheres are very near the surface of the predicted bounding sphere. As we consider nodes deeper in the trees, we see that at a depth of 7, only around 1% of vertices protrude from their spheres. Various predicted tree depths are shown on a test set deformation in Fig. 9.

In Fig. 7, we demonstrate that across our test set, we learned the correct radii within at most 1-2%, independent of tree depth. Notably, this percentage error is very tight as compared to a BD-Tree, which often has radial error of over 10-20% across our test sets.

In Fig. 10, we visually compare the results of our method against the BD-Tree method. We find that, even when deformations are small, the BD-Tree method

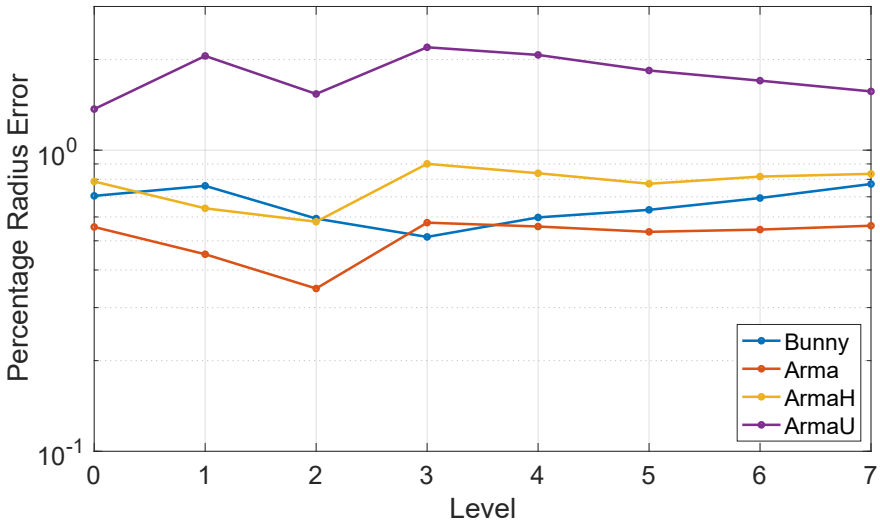


Fig. 7: The percentage error of the sphere radius, measured with respect to the exact spheres (*i.e.*, $10^{-1} = 0.1\%$).

is very conservative with it's bounds, while our method produces bounds much closer to the ground truth.

Fig. 8 shows how the percentage of vertices of Arma outside the spheres decreases as we increase the radii by adding a percentage margin. We vary the added percentage from 1% to 25%. The errors for top-level spheres quickly reach zero, whereas the errors for lower-level spheres go down more slowly. However, we note that even for Level 7 spheres, the decrease is exponential.

5 Conclusion & Future Work

We presented NBD-Tree, a neural network-based approach for collision culling of full-space deformable objects. NBD-Trees allows for interactive collision culling for large-scale, full-space deformable objects. It is ideal for collision detection with a highly detailed mesh, possibly driven by lower resolution physics. The performance depends not on the discretization of the mesh but on the number of modes and clusters. Like the classic BD-Tree algorithm, our NBD-Tree algorithm uses low-dimensional modes, but our NBD-Tree algorithm is not limited to reduced-space deformations because we use a neural network to learn the corrections needed to update the spheres to enclose full-space deformations that are in the training set. We compute these low-dimensional modes efficiently and approximately, without going through every vertex of the mesh, relying on the network to overcome the potential errors stemming from these approximations.

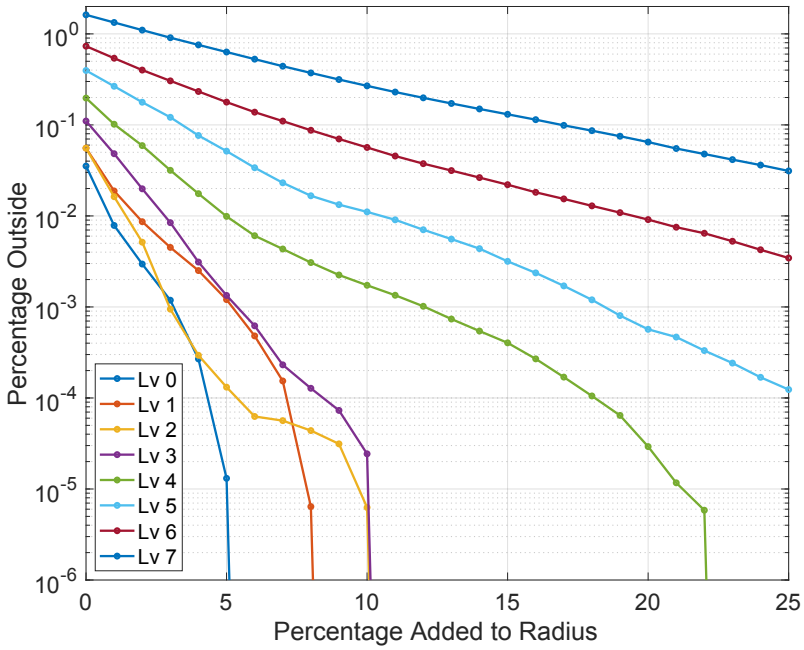


Fig. 8: The percentage of vertices of ARMA that fall outside their predicted node after adding a percentage offset to the predicted radius. The errors fall exponentially.

Like other learning-based methods, our approach does not work well for extrapolating outside the training set. For extreme deformations that were not seen before, our spheres will inevitably produce more and more false negatives. We also need to train the networks for every new object to be simulated. It would be interesting to see if different discretizations of the same object (*e.g.,* ARMA, ARMAHD, and ARMAUHD) can share the same trained network. This is likely to be the case if the modes are compatible, such as linear modal modes.

If there are many deep inter-penetrations, a standard sphere tree that naively updates its spheres can eventually become cheaper, since their sphere checks are extremely fast. Conversely, our method is extremely efficient if there are few collisions with deep inter-penetrations. Our method is fast at computing the code for the network, but compared to the standard sphere tree, actually carrying out sphere-sphere collisions is more expensive because an MLP needs to be evaluated to compute the center and the radius. Our current implementation based on Eigen can be improved significantly to speed up this process, for example, by using the GPU.

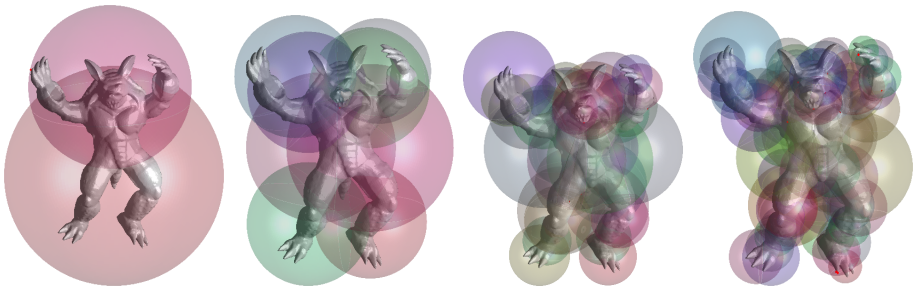
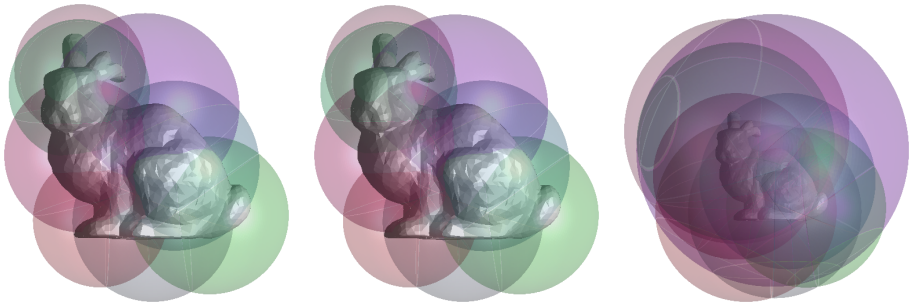


Fig. 9: Predicted bounding spheres for the ArmaHD mesh, at a test set deformation. We show the results of depths 1, 3, 5 and 7.



(a) (b) (c)

Fig. 10: We compare the results of our method (a) and the traditional BD-Tree method (c) against the ground truth (b) using 128 modes of deformation. Our method closely matches the ground truth, while the BD-Tree method is vastly over conservative with its bounds, even at a mild deformation.

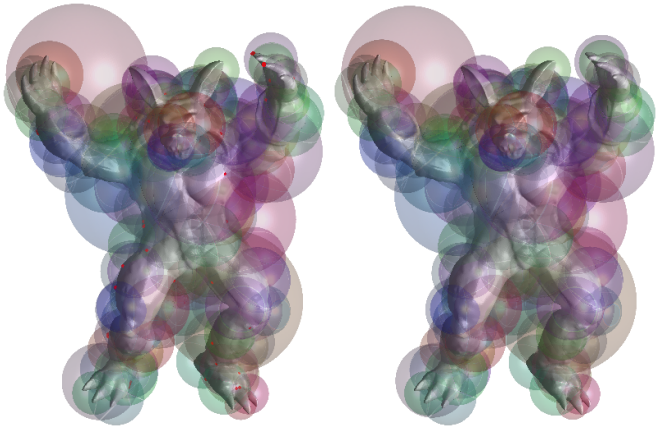


Fig. 11: The predicted bounding spheres at a depth of 7 for a test set deformation of the ArmaUHD model (left), compared to the ground truth (right).

Bibliography

[1] Alldieck, T., Xu, H., Sminchisescu, C.: imGHUM: Implicit generative models of 3d human shape and articulated pose. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 5461–5470 (2021)

[2] Chen, Z., Zhang, H.: Learning implicit fields for generative shape modeling. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5939–5948 (2019)

[3] Deng, B., Lewis, J.P., Jeruzalski, T., Pons-Moll, G., Hinton, G., Norouzi, M., Tagliasacchi, A.: NASA neural articulated shape approximation. In: European Conference on Computer Vision, pp. 612–628, Springer (2020)

[4] Ericson, C.: Real-time collision detection. Crc Press (2004)

[5] Faure, F., Allard, J., Falipou, F., Barbier, S.: Image-based collision detection and response between arbitrary volumetric objects. In: ACM Siggraph/Eurographics Symposium on Computer Animation, pp. 155–162, Eurographics Association (2008)

[6] Fisher, S., Lin, M.C.: Deformed distance fields for simulation of non-penetrating flexible bodies. In: Computer Animation and Simulation 2001, pp. 99–111, Springer (2001)

[7] Guibas, L., Nguyen, A., Russel, D., Zhang, L.: Collision detection for deforming necklaces. In: Proceedings of the Eighteenth Annual Symposium on Computational Geometry, p. 33–42, SCG '02 (2002), ISBN 1581135041

[8] Haddadin, S., De Luca, A., Albu-Schäffer, A.: Robot collisions: A survey on detection, isolation, and identification. IEEE Transactions on Robotics **33**(6), 1292–1312 (2017)

[9] James, D.L., Pai, D.K.: BD-Tree: Output-sensitive collision detection for reduced deformable models. ACM Trans. Graph. **23**(3), 393–398 (Aug 2004), ISSN 0730-0301

[10] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)

[11] Lin, M., Gottschalk, S.: Collision detection between geometric models: A survey. In: Proc. of IMA conference on mathematics of surfaces, vol. 1, pp. 602–608, Citeseer (1998)

[12] Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S., Corse, Z.: Local optimization for robust signed distance field collision. Proceedings of the ACM on Computer Graphics and Interactive Techniques **3**(1), 1–17 (2020)

[13] McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J., Sifakis, E.: Efficient elasticity for character skinning with contact and collisions. ACM Trans. Graph. **30**(4) (Jul 2011), ISSN 0730-0301

[14] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3D reconstruction in function space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4460–4470 (2019)

[15] Müller, M., Heidelberger, B., Teschner, M., Gross, M.: Meshless deformations based on shape matching. ACM Trans. Graph. **24**(3), 471–478 (Jul 2005), ISSN 0730-0301

[16] Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: DeepSDF: Learning continuous signed distance functions for shape representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 165–174 (2019)

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035, Curran Associates, Inc. (2019)

[18] Pentland, A., Williams, J.: Good vibrations: Modal dynamics for graphics and animation. vol. 23, p. 207–214, ACM, New York, NY, USA (Jul 1989), ISSN 0097-8930

[19] Santesteban, I., Otaduy, M.A., Casas, D.: SNUG: Self-supervised neural dynamic garments. arXiv preprint arXiv:2204.02219 (2022)

[20] Seyb, D., Jacobson, A., Nowrouzezahrai, D., Jarosz, W.: Non-linear sphere tracing for rendering deformed signed distance fields. ACM Trans. Graph. **38**(6) (nov 2019), ISSN 0730-0301

[21] Sifakis, E., Barbic, J.: FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction. In: ACM SIGGRAPH 2012 Courses, SIGGRAPH '12, Association for Computing Machinery, New York, NY, USA (2012), ISBN 9781450316781

[22] Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.P., Faure, F., Magnenat-Thalmann, N., Strasser, W., et al.: Collision detection for deformable objects. In: Computer graphics forum, vol. 24, pp. 61–81, Wiley Online Library (2005)

[23] Wang, B., Faure, F., Pai, D.K.: Adaptive image-based intersection volume. ACM Trans. Graph. **31**(4) (jul 2012), ISSN 0730-0301

[24] Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V., Sridhar, S.: Neural fields in visual computing and beyond. arXiv preprint arXiv:2111.11426 (2021)